



CS 152 Computer Architecture and Engineering

Lecture 10 - Virtual Memory

Krste Asanovic

Electrical Engineering and Computer Sciences
University of California at Berkeley

<http://www.eecs.berkeley.edu/~krste>
<http://inst.eecs.berkeley.edu/~cs152>

February 23, 2010

CS152, Spring 2010



Last time in Lecture 9

- Protection and translation required for multiprogramming
 - Base and bounds, early simple scheme
- Page-based translation and protection avoids need for memory compaction, easy allocation by OS
 - But need to indirect in large page table on every access
- Address spaces accessed sparsely
 - Can use multi-level page table to hold translation/protection information
- Address space access with locality
 - Can use “translation lookaside buffer” (TLB) to cache address translations (sometimes known as address translation cache)
 - Still have to walk page tables on TLB miss, can be hardware or software talk
- Virtual memory uses DRAM as a “cache” of disk memory, allows very cheap main memory

February 23, 2010

CS152, Spring 2010

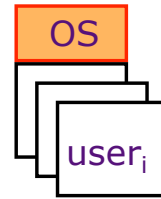


Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

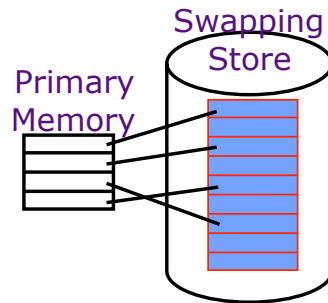
several users, each with their private address space and one or more shared address spaces
page table = name space



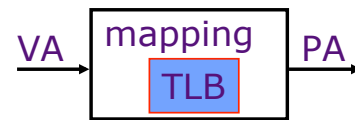
Demand Paging

Provides the ability to run programs larger than the primary memory

Hides differences in machine configurations



The price is address translation on each memory reference



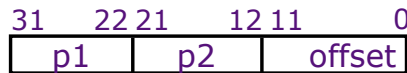
February 23, 2010

CS152, Spring 2010



Hierarchical Page Table

Virtual Address



10-bit L1 index
10-bit L2 index

Root of the Current Page Table

(Processor Register)

p1

Level 1 Page Table

p2

Level 2 Page Tables

offset

- page in primary memory
- page in secondary memory
- PTE of a nonexistent page

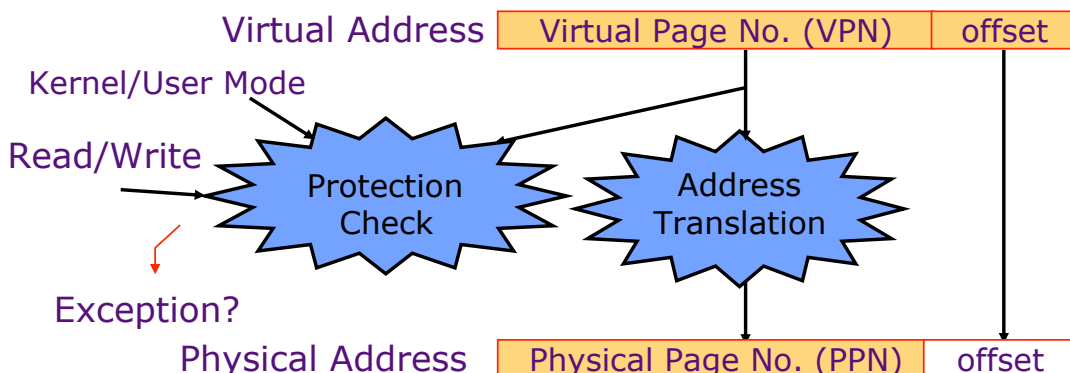
Data Pages

February 23, 2010

CS152, Spring 2010



Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

February 23, 2010

CS152, Spring 2010



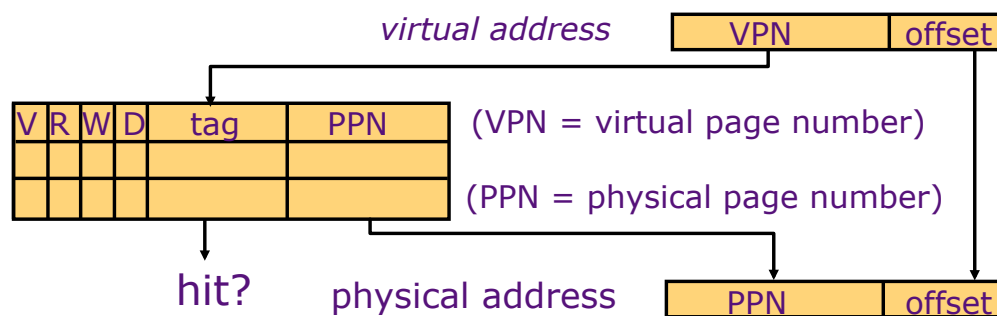
Translation Lookaside Buffers

Address translation is very expensive!

In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*

TLB hit ⇒ *Single Cycle Translation*
 TLB miss ⇒ *Page Table Walk to refill*



February 23, 2010

CS152, Spring 2010



Handling a TLB Miss

Software (MIPS, Alpha)

TLB miss causes an exception and the operating system walks the page tables and reloads TLB. A *privileged "untranslated" addressing mode used for walk*

Hardware (SPARC v8, x86, PowerPC)

A memory management unit (MMU) walks the page tables and reloads the TLB

If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

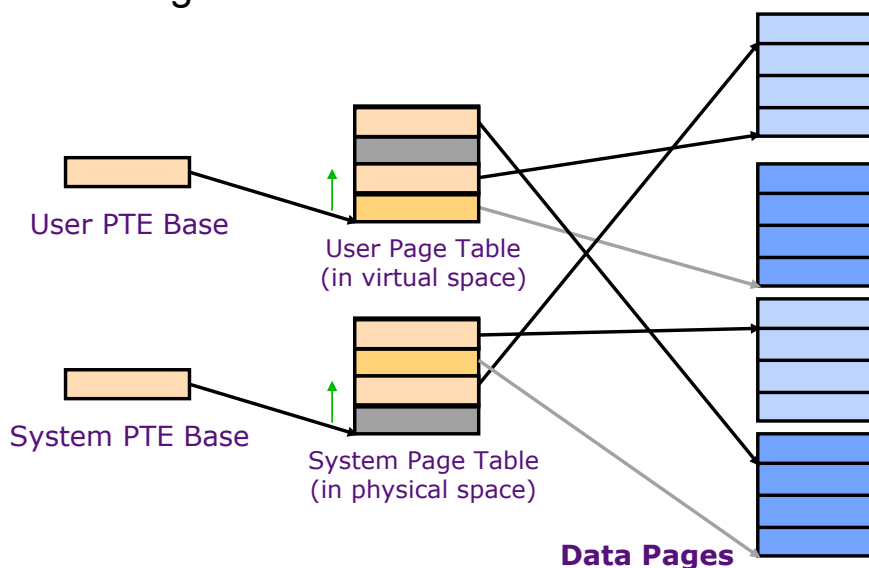
February 23, 2010

CS152, Spring 2010



Translation for Page Tables

- Can references to page tables cause TLB misses?
- Can this go on forever?

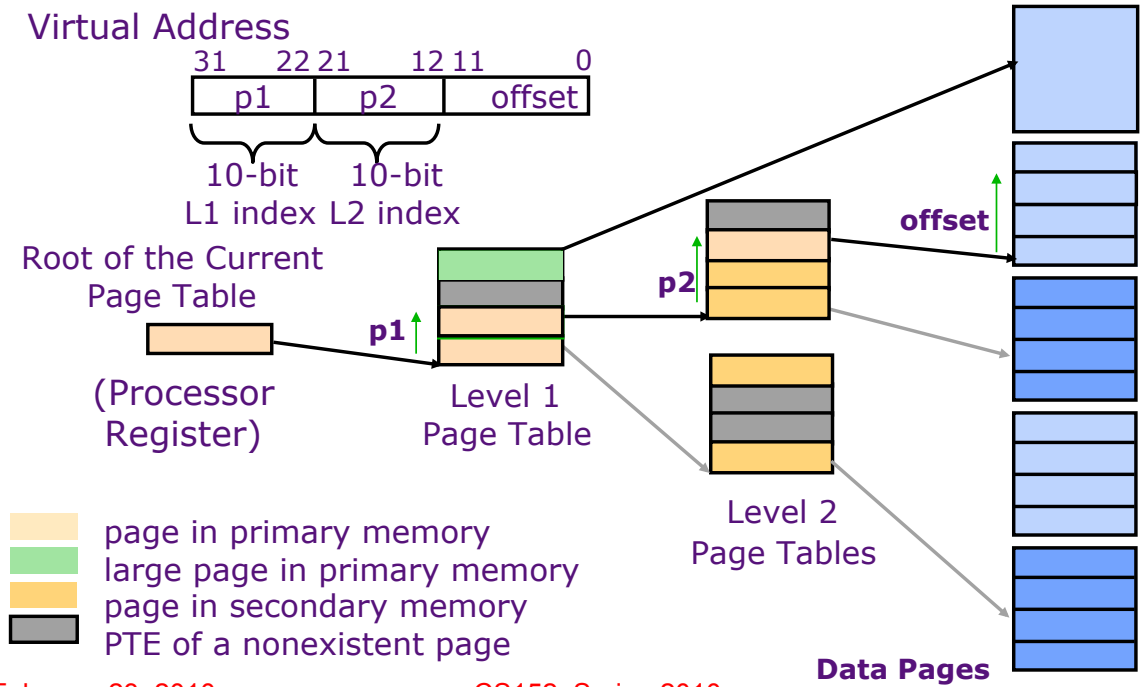


February 23, 2010

CS152, Spring 2010



Variable-Sized Page Support



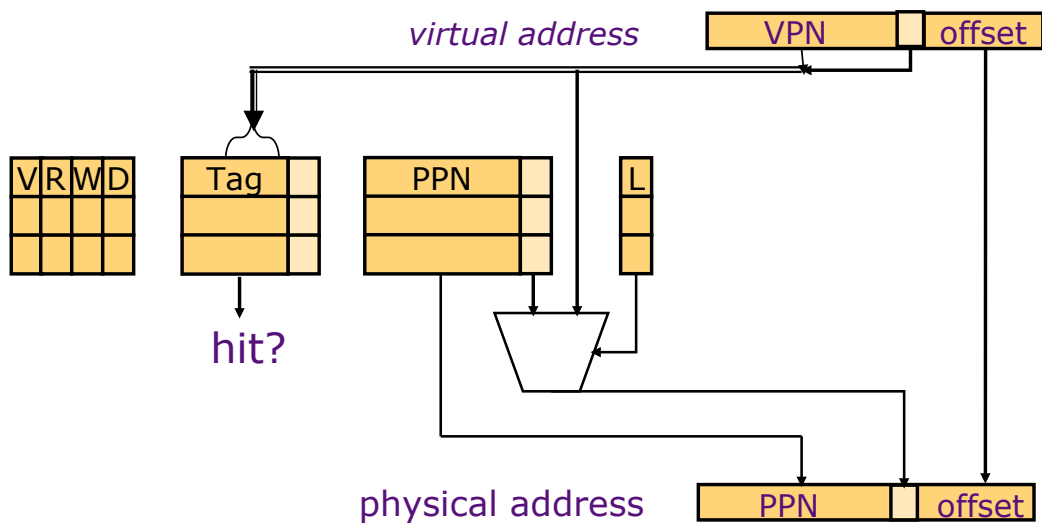
February 23, 2010

CS152, Spring 2010



Variable-Size Page TLB

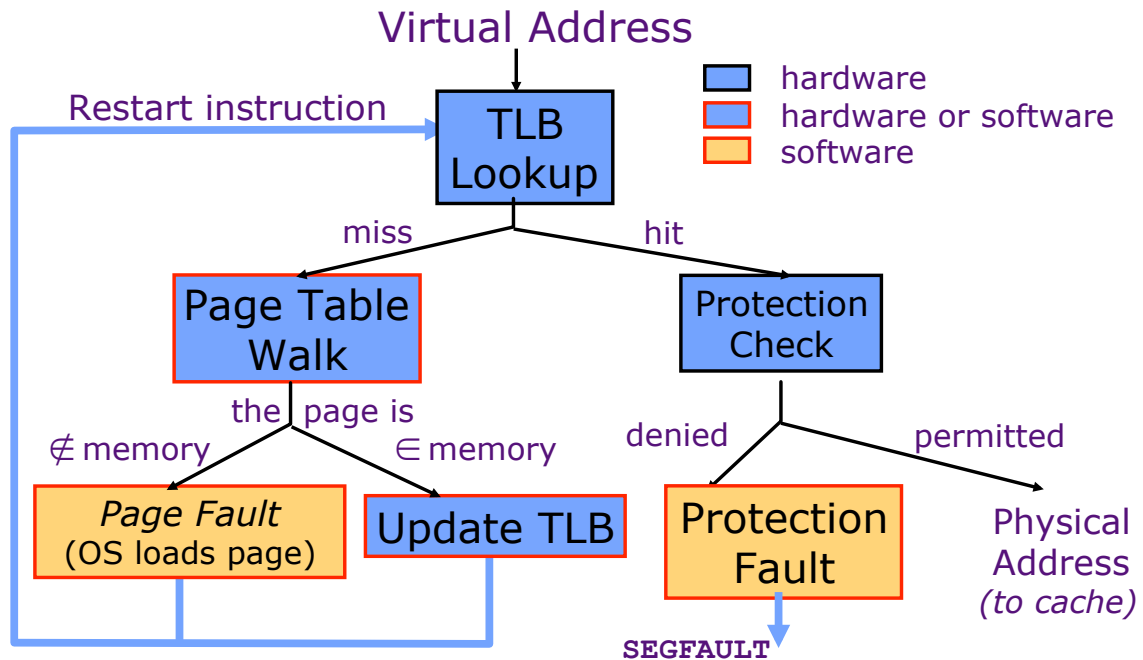
Some systems support multiple page sizes.



February 23, 2010

CS152, Spring 2010

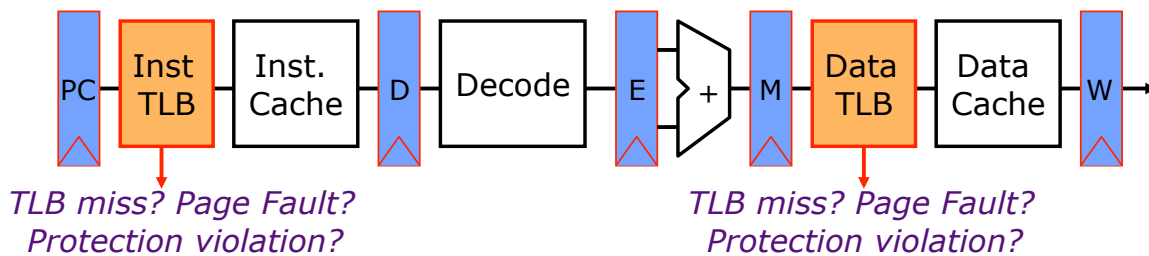
Address Translation: putting it all together



February 23, 2010

CS152, Spring 2010

Address Translation in CPU Pipeline



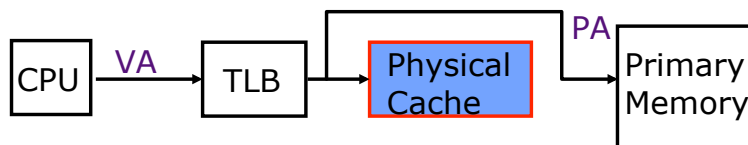
- Software handlers need *restartable* exception on page fault or protection violation
- Handling a TLB miss needs a *hardware* or *software* mechanism to refill TLB
- Need mechanisms to cope with the additional latency of a TLB:
 - slow down the clock
 - pipeline the TLB and cache access
 - virtual address caches
 - parallel TLB/cache access

February 23, 2010

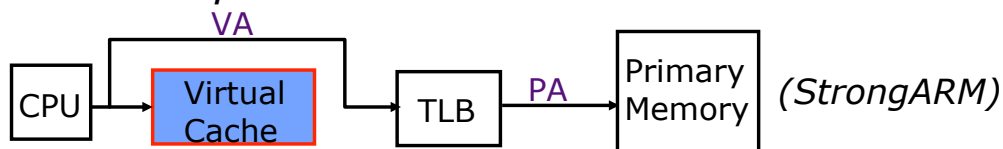
CS152, Spring 2010



Virtual Address Caches



Alternative: place the cache before the TLB



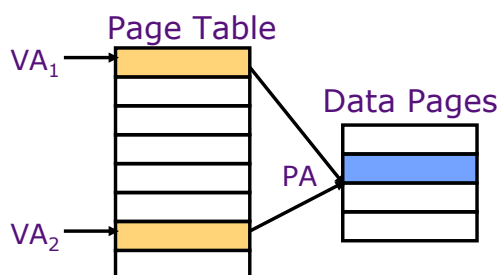
- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- *aliasing problems* due to the sharing of pages (-)
- maintaining cache coherence (-) (see later in course)

February 23, 2010

CS152, Spring 2010



Aliasing in Virtual-Address Caches



Two virtual pages share one physical page

Tag	Data
VA ₁	1st Copy of Data at PA
VA ₂	2nd Copy of Data at PA

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Disallow aliases to coexist in cache*

Software (i.e., OS) solution for direct-mapped cache

VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)

February 23, 2010

CS152, Spring 2010



CS152 Administrivia

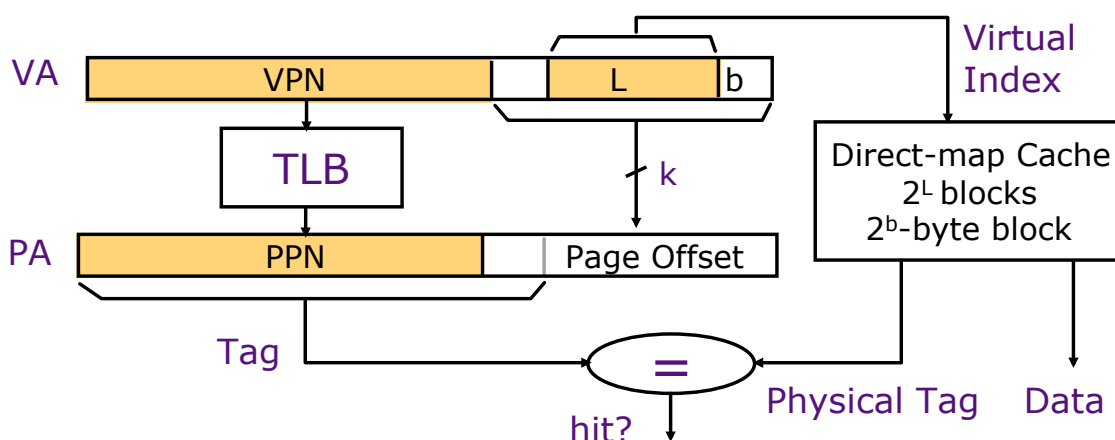
- Quiz 2: Tuesday March 3. Covers Lectures 6-8, PS2, and Lab 2

February 23, 2010

CS152, Spring 2010



Concurrent Access to TLB & Cache



Index L is available without consulting the TLB
 ⇒ *cache and TLB accesses can begin simultaneously*
 Tag comparison is made after both accesses are completed

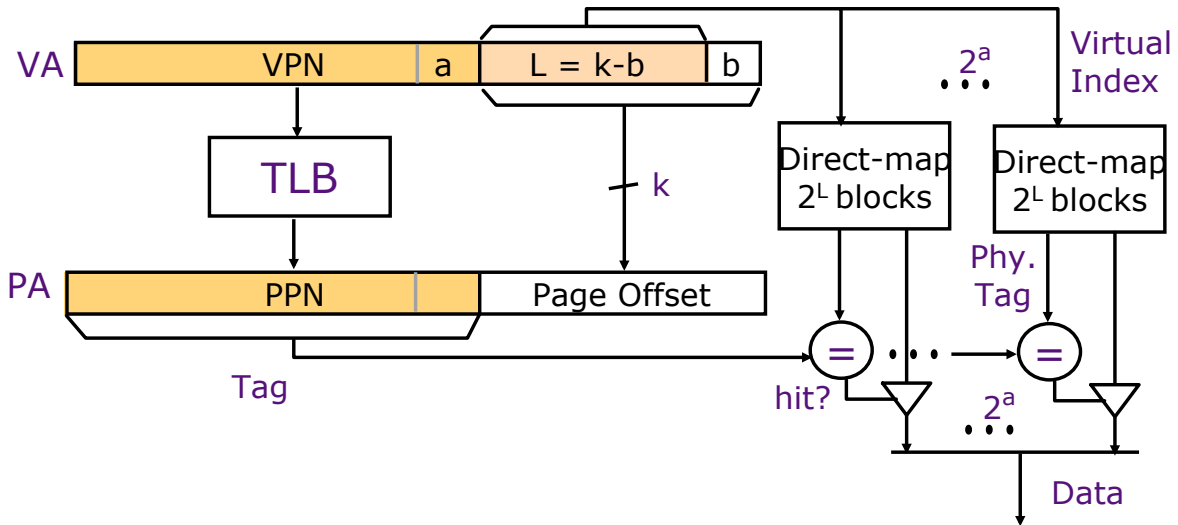
Cases: $L + b = k$, $L + b < k$, $L + b > k$

February 23, 2010

CS152, Spring 2010



Virtual-Index Physical-Tag Caches: Associative Organization



After the PPN is known, 2^a physical tags are compared
How does this scheme scale to larger caches?

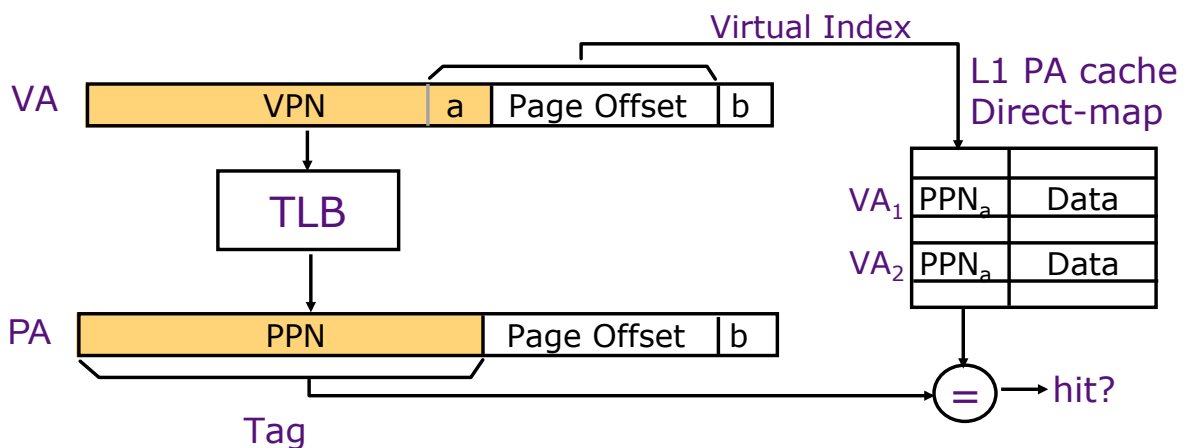
February 23, 2010

CS152, Spring 2010



Concurrent Access to TLB & Large L1

The problem with $L1 > \text{Page size}$



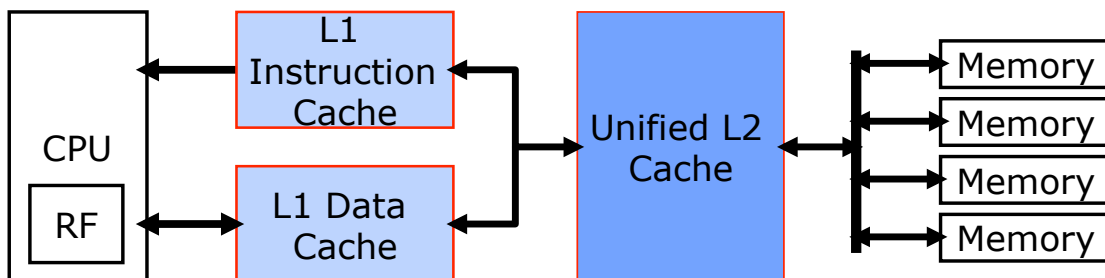
Can VA_1 and VA_2 both map to PA ?

February 23, 2010

CS152, Spring 2010



A solution via **Second Level Cache**

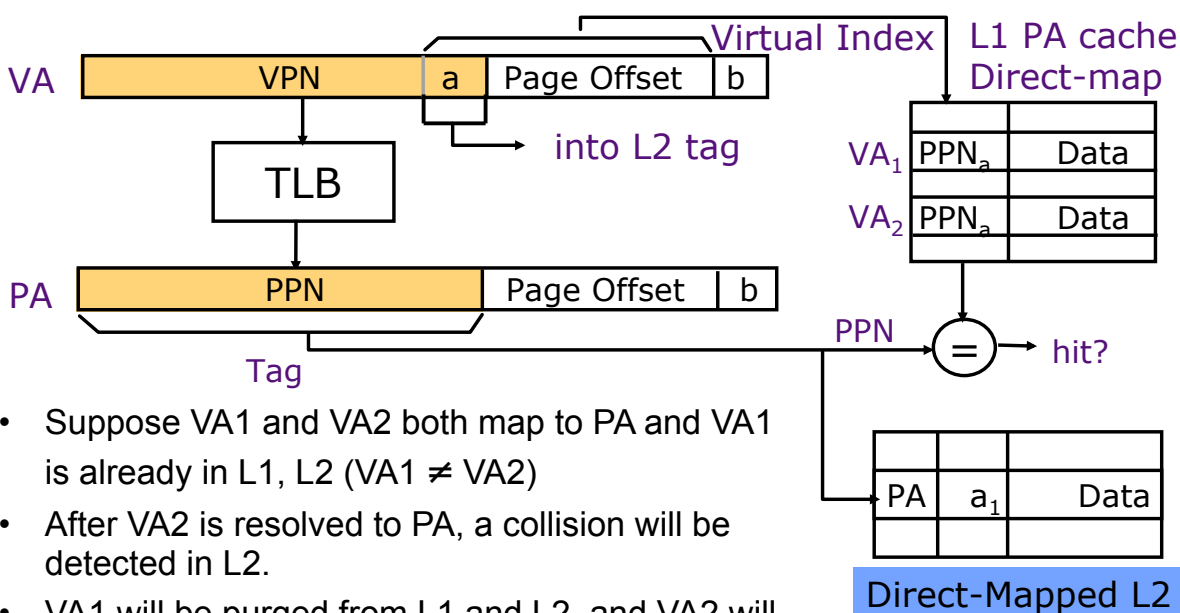


Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is "inclusive" of both Instruction and Data caches



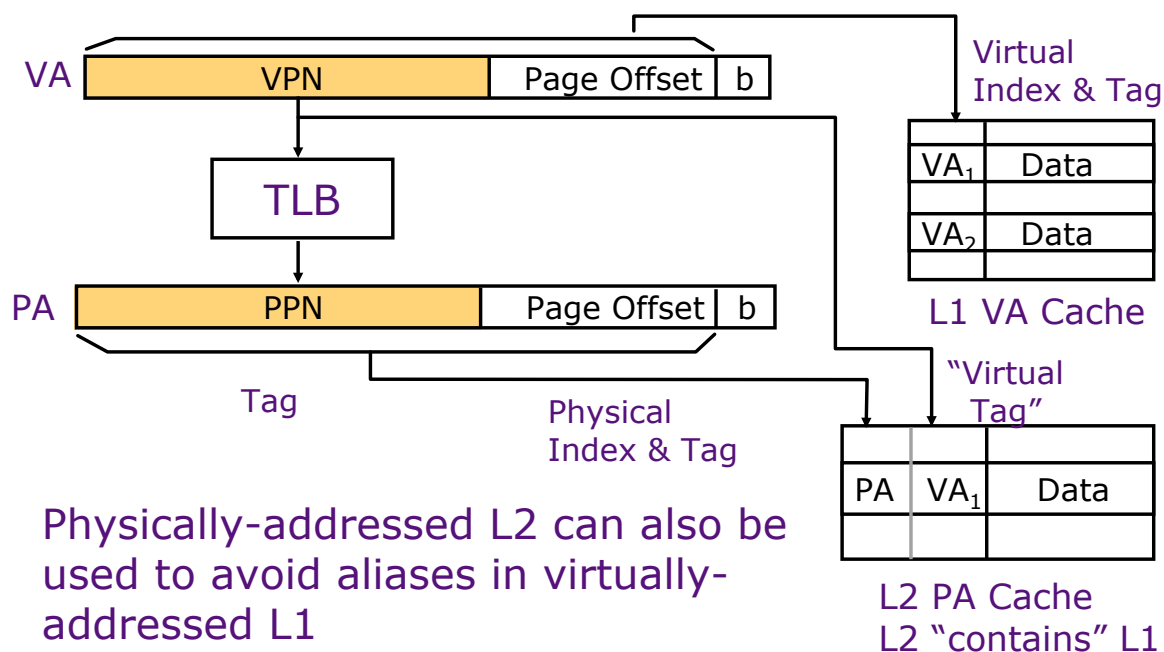
Anti-Aliasing Using L2: *MIPS R10000*



- Suppose VA₁ and VA₂ both map to PA and VA₁ is already in L1, L2 (VA₁ ≠ VA₂)
- After VA₂ is resolved to PA, a collision will be detected in L2.
- VA₁ will be purged from L1 and L2, and VA₂ will be loaded ⇒ *no aliasing!*



Virtually-Addressed L1: Anti-Aliasing using L2



February 23, 2010

CS152, Spring 2010



Page Fault Handler

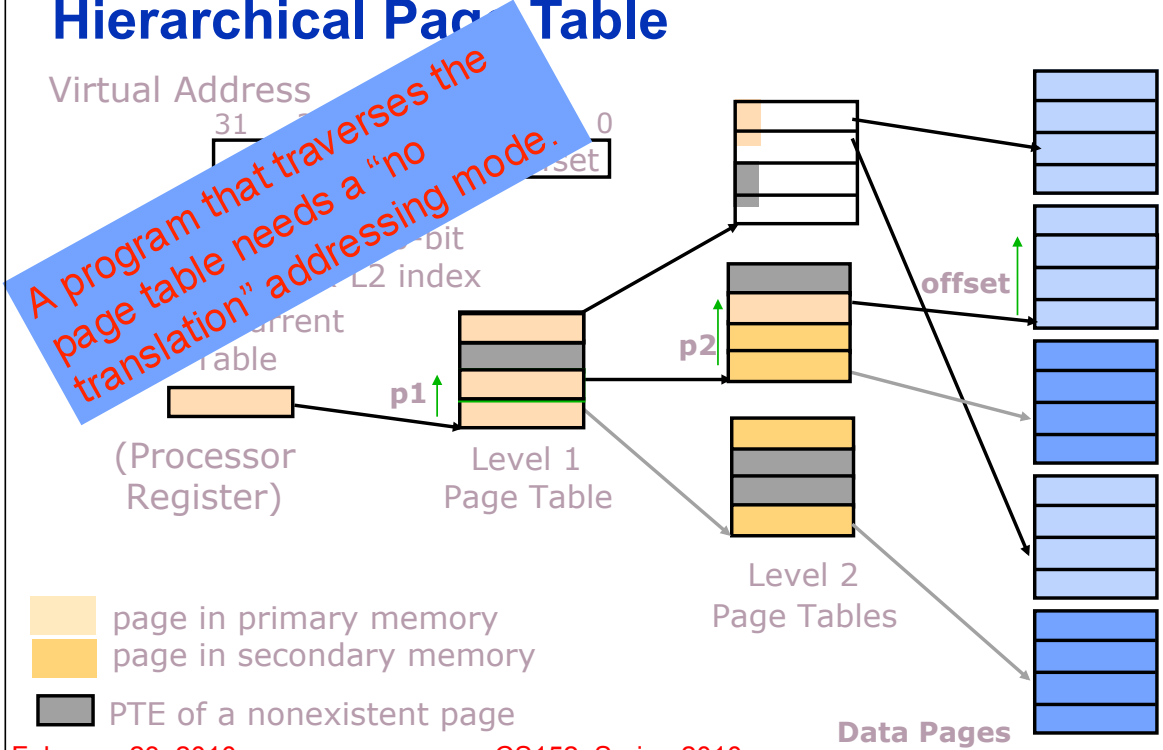
- When the referenced page is not in DRAM:
 - The missing page is located (or created)
 - It is brought in from disk, and page table is updated
 - Another job may be run on the CPU while the first job waits for the requested page to be read from disk*
 - If no free pages are left, a page is swapped out
 - Pseudo-LRU replacement policy*
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
 - Untranslated addressing mode is essential to allow kernel to access page tables

February 23, 2010

CS152, Spring 2010



Hierarchical Page Table



February 23, 2010

CS152, Spring 2010



Swapping a Page of a Page Table



A PTE in primary memory contains primary or secondary memory addresses



A PTE in secondary memory contains *only* secondary memory addresses

⇒ a page of a PT can be swapped out only if none its PTE's point to pages in the primary memory

Why? _____

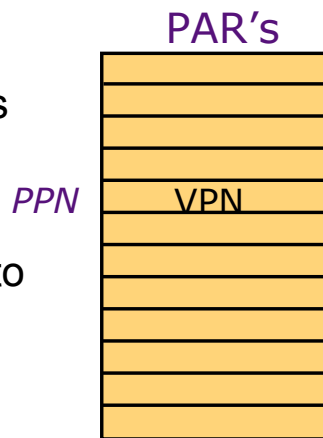
February 23, 2010

CS152, Spring 2010

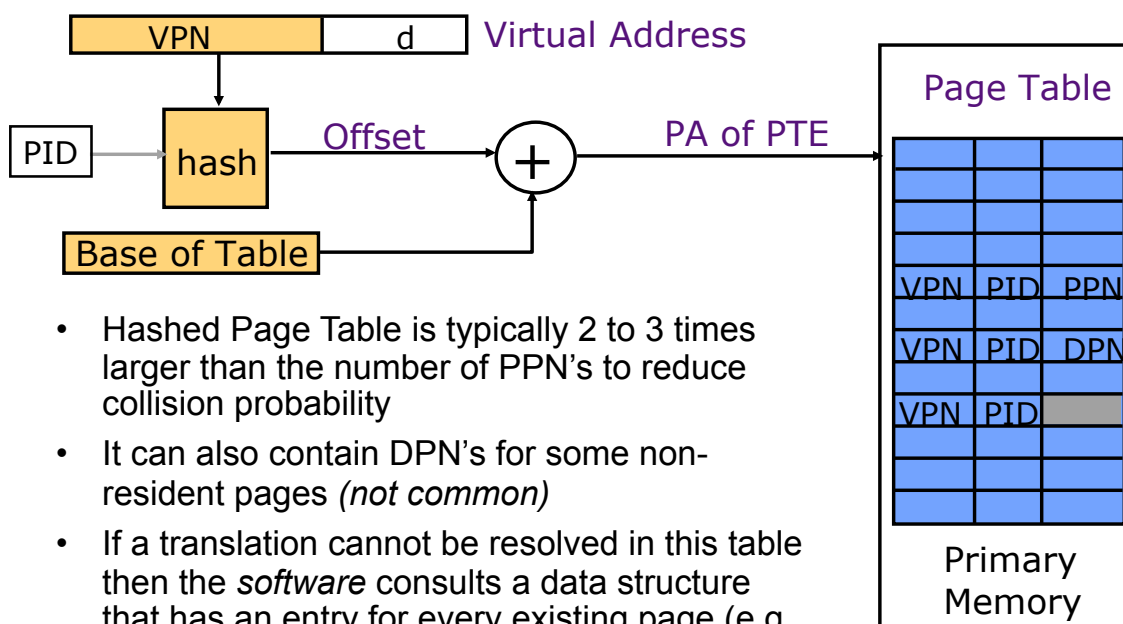


Atlas Revisited

- One PAR for each physical page
- PAR's contain the VPN's of the pages *resident in primary memory*
- *Advantage:* The size is proportional to the size of the primary memory
- *What is the disadvantage ?*



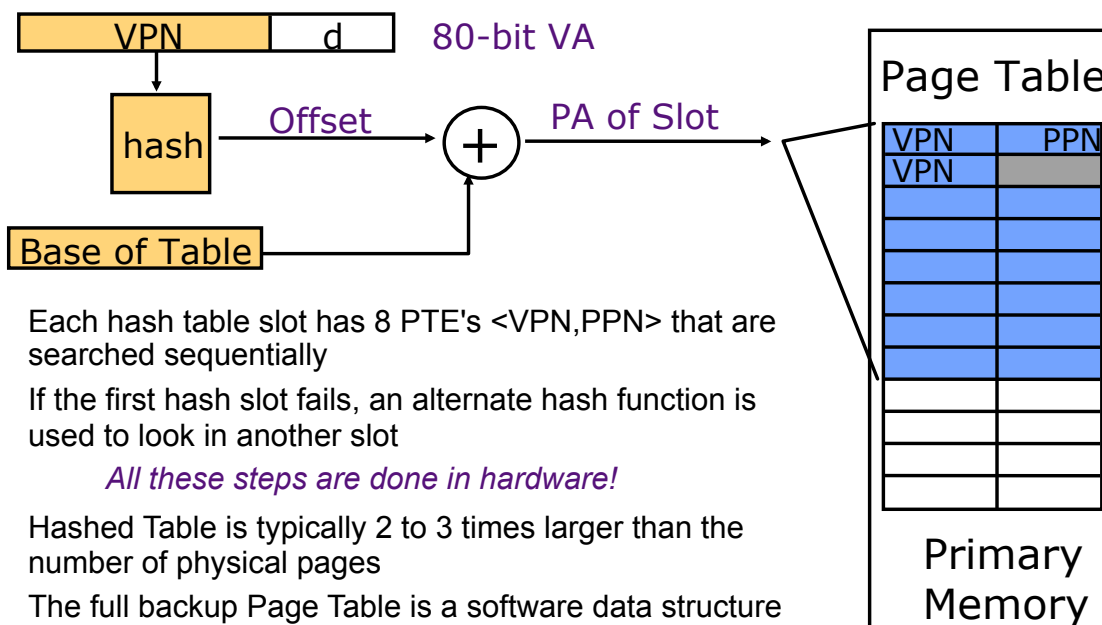
Hashed Page Table: Approximating Associative Addressing



- Hashed Page Table is typically 2 to 3 times larger than the number of PPN's to reduce collision probability
- It can also contain DPN's for some non-resident pages (*not common*)
- If a translation cannot be resolved in this table then the *software* consults a data structure that has an entry for every existing page (e.g., full page table)



Power PC: Hashed Page Table



- Each hash table slot has 8 PTE's <VPN,PPN> that are searched sequentially
- If the first hash slot fails, an alternate hash function is used to look in another slot
 - All these steps are done in hardware!*
- Hashed Table is typically 2 to 3 times larger than the number of physical pages
- The full backup Page Table is a software data structure

February 23, 2010

CS152, Spring 2010



Virtual Memory Use Today - 1

- Desktops/servers have full demand-paged virtual memory
 - Portability between machines with different memory sizes
 - Protection between multiple users or multiple tasks
 - Share small physical memory among active tasks
 - Simplifies implementation of some OS features
- Vector supercomputers have translation and protection but not demand-paging
- (Older Crays: base&bound, Japanese & Cray X1/X2: pages)
 - Don't waste expensive CPU time thrashing to disk (make jobs fit in memory)
 - Mostly run in batch mode (run set of jobs that fits in memory)
 - Difficult to implement restartable vector instructions

February 23, 2010

CS152, Spring 2010



Virtual Memory Use Today - 2

- Most embedded processors and DSPs provide physical addressing only
 - Can't afford area/speed/power budget for virtual memory support
 - Often there is no secondary storage to swap to!
 - Programs custom written for particular memory configuration in product
 - Difficult to implement restartable instructions for exposed architectures

February 23, 2010

CS152, Spring 2010



Acknowledgements

- These slides contain material developed and copyright by:
 - Arvind (MIT)
 - Krste Asanovic (MIT/UCB)
 - Joel Emer (Intel/MIT)
 - James Hoe (CMU)
 - John Kubiatowicz (UCB)
 - David Patterson (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252

February 23, 2010

CS152, Spring 2010