Modern Virtual Memory Systems

Krste Asanovic
Laboratory for Computer Science
M.I.T.

http://www.csg.lcs.mit.edu/6.823

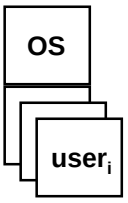
Modern Virtual Memory Systems

illusion of a large, private, uniform store

Protection & Privacy

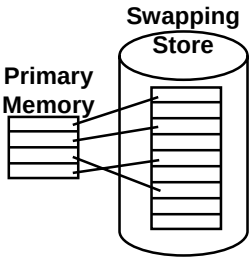
- several users, each with their private address space and one or more shared address spaces

  page table ≡ name space



Demand Paging

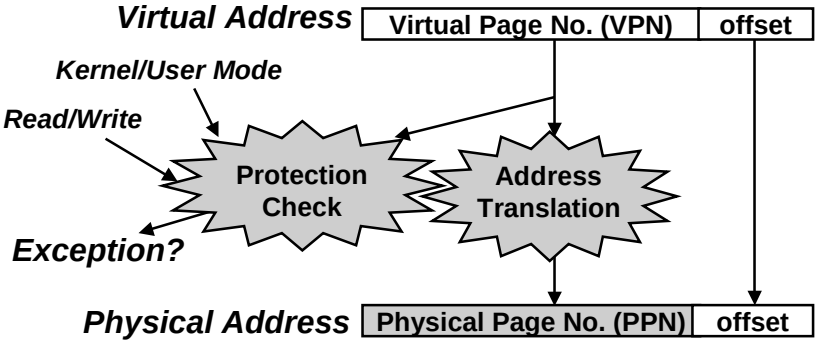
- ability to run a program larger than than the primary memory



⇒ portability on machines with different memory configurations

The price is address translation on each memory reference

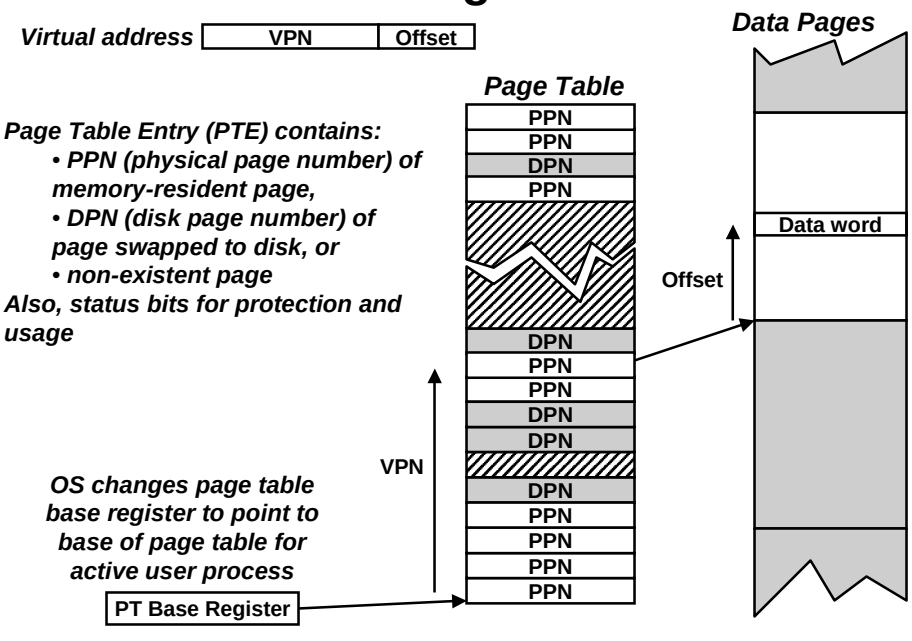
Address Translation and Protection



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

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

Linear Page Table



Page Table Entry (PTE) contains:

- *PPN (physical page number) of memory-resident page,*
- *DPN (disk page number) of page swapped to disk, or*
- *non-existent page*

Also, status bits for protection and usage

OS changes page table base register to point to base of page table for active user process

Size of Linear Page Table

With 32-bit addresses, 4-KB pages, and 4-byte PTEs:

- ⇒ 2^{20} PTEs, i.e, 4 MB page table per user
- ⇒ 4 GB of swap needed to back up full virtual address space

Larger pages?

- more internal fragmentation (don't use all memory in page)
- larger page fault penalty (more to read from disk)

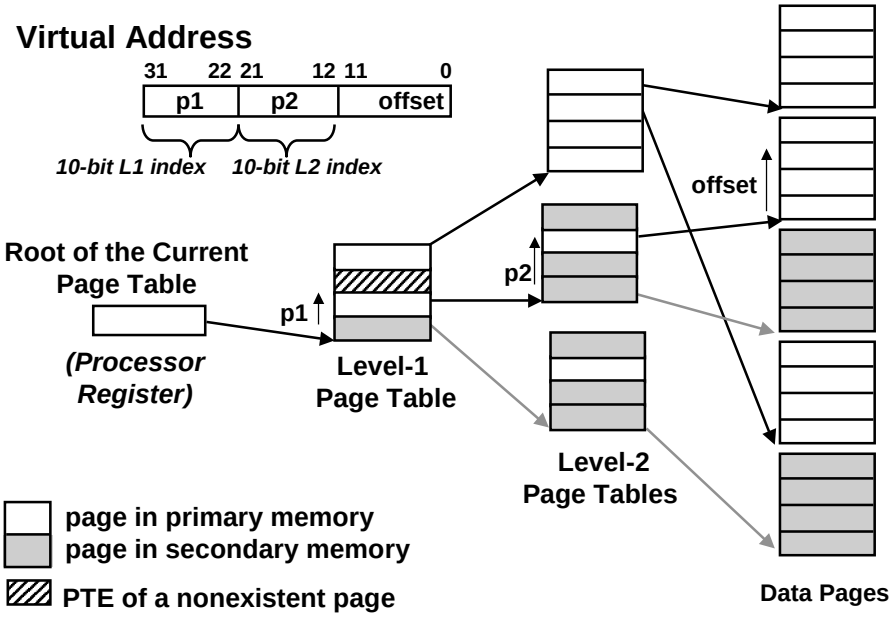
What about 64-bit virtual address space???

- Even 1MB pages would require 2^{42} 8-byte PTEs (35 TB!)

Virtual address space is large but only a small fraction of pages are populated

⇒ ***Use a sparse representation of page table***

Hierarchical Page Table



Translation Lookaside Buffers

Address translation is very expensive!

In a two-level page table, each reference becomes

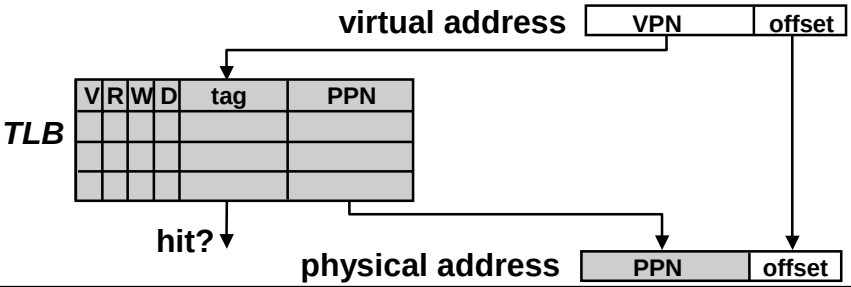
the best case - 3 memory references

the worst case - 2 page faults (disk accesses) + ...

Solution: *Cache translations in TLB*

TLB hit ⇒ *Single Cycle Translation*

TLB miss ⇒ *Page Table Walk to refill*



TLB Designs

Typically 32-128 entries

Usually fully associative

- **Each entry maps a large page, hence less spatial locality across pages => more likely that two entries conflict**
- **Sometimes larger TLBs are 4-8 way set-associative**

Random or FIFO replacement policy

Typically only one page mapping per entry

TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = _____?

Handling A TLB Miss

Software (MIPS, Alpha)

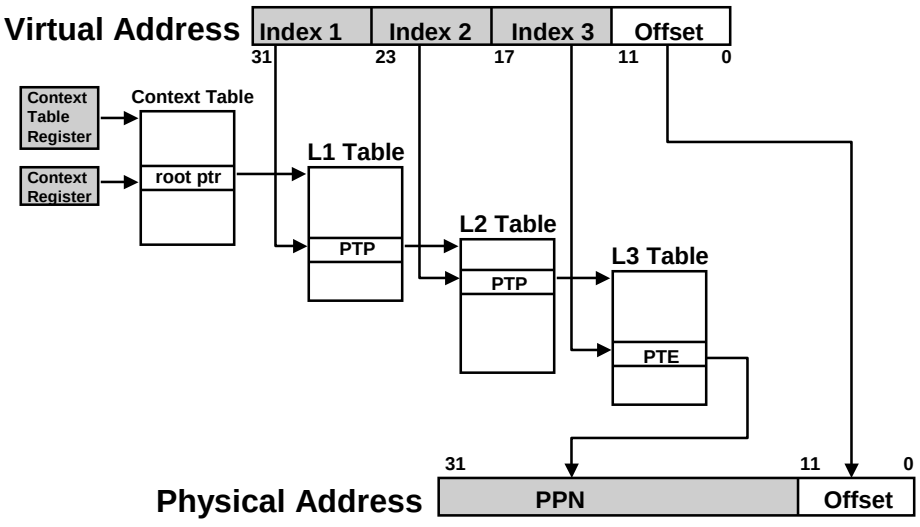
TLB miss causes an exception and the operating system walks the page tables and reloads TLB
privileged "untranslated" addressing mode used for walk

Hardware (SPARC v8, x86, PowerPC)

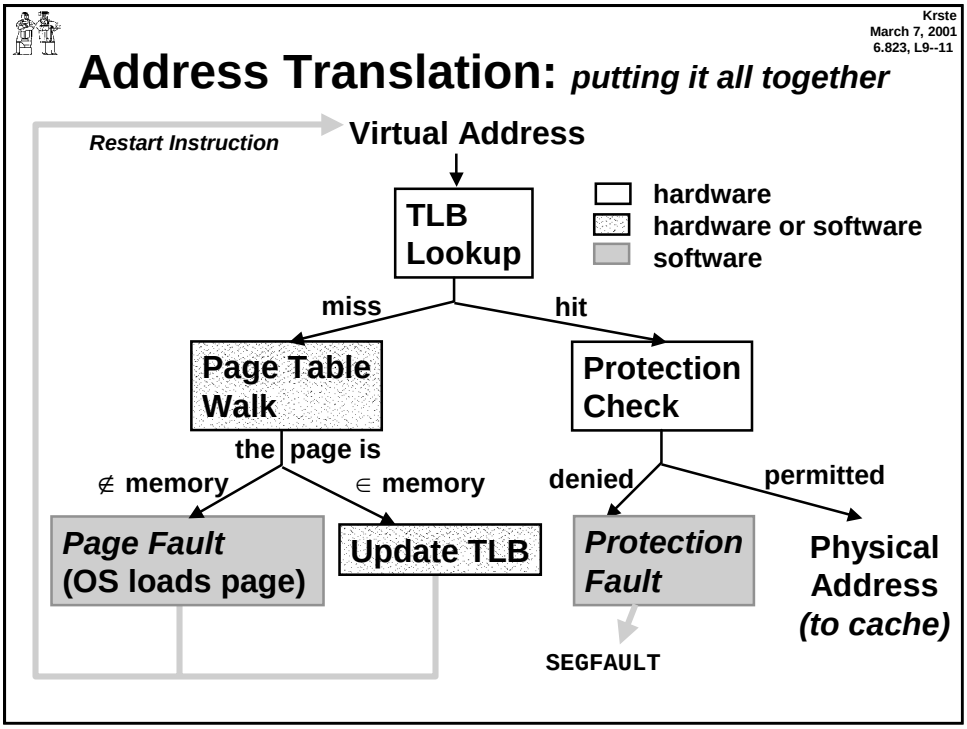
A memory management unit (MMU) walks the page tables and reloads the TLB

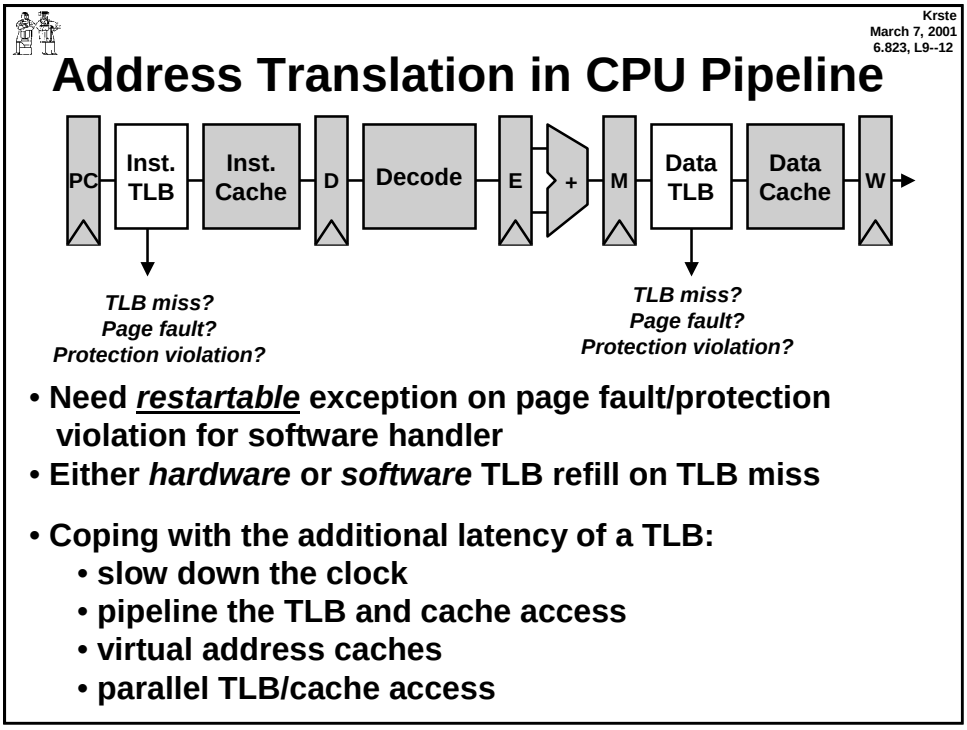
If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

Hierarchical Page Table Walk: *SPARC v8*



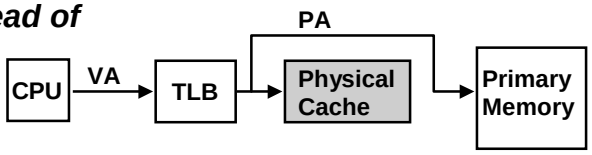
MMU does this table walk in hardware on a TLB miss


Krste
March 7, 2001
6.823, L9--11
Address Translation: putting it all together
Restart Instruction
Virtual Address
TLB Lookup
hardware
hardware or software
software
miss
hit
Page Table Walk
Protection Check
the page is
∉ memory
∈ memory
denied
permitted
Page Fault (OS loads page)
Update TLB
Protection Fault
Physical Address (to cache)
SEGFAULT

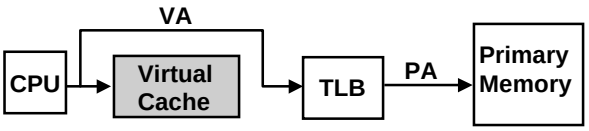

Krste
March 7, 2001
6.823, L9--12
Address Translation in CPU Pipeline
PC
Inst. TLB
Inst. Cache
D
Decode
E
+
M
Data TLB
Data Cache
W
TLB miss?
Page fault?
Protection violation?
TLB miss?
Page fault?
Protection violation?
• Need restartable exception on page fault/protection violation for software handler
• Either hardware or software TLB refill on TLB miss
• Coping with the additional latency of a TLB:
• slow down the clock
• pipeline the TLB and cache access
• virtual address caches
• parallel TLB/cache access

Virtual Address Caches

Instead of



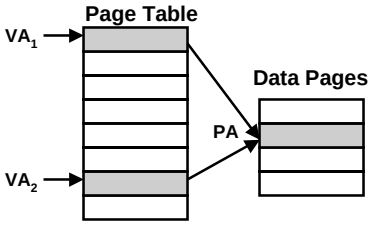
place the cache before the TLB



(StrongARM)

+ 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

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

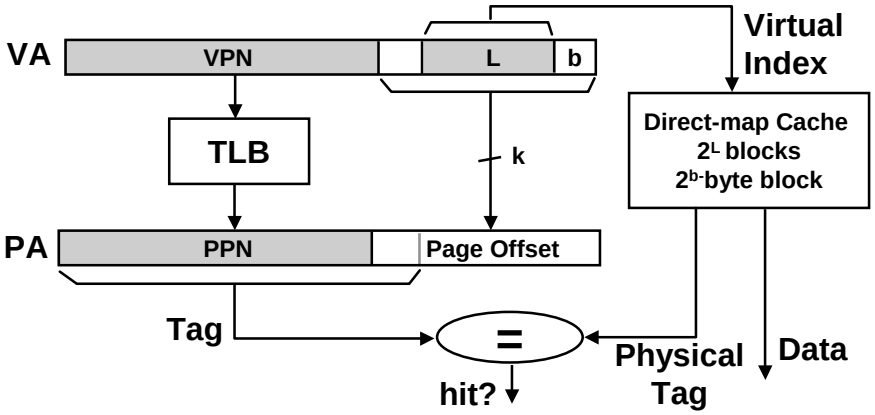
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

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 cache (used by early SPARCs)

Concurrent Access to TLB & Cache



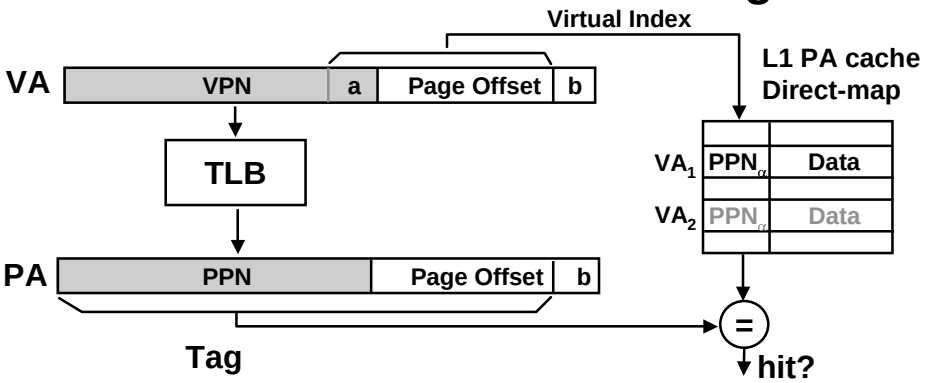
Index L is available without consulting the TLB

⇒ ***cache and TLB accesses can begin simultaneously***

Tag comparison is made after both accesses are completed

What if the cache is larger than the page size? ***(L+b > k)***

Concurrent Access to TLB & Large L1



Suppose VA_1 and VA_2 both map to PA and VA_1 and VA_2 differ in the lower *'a'* bits

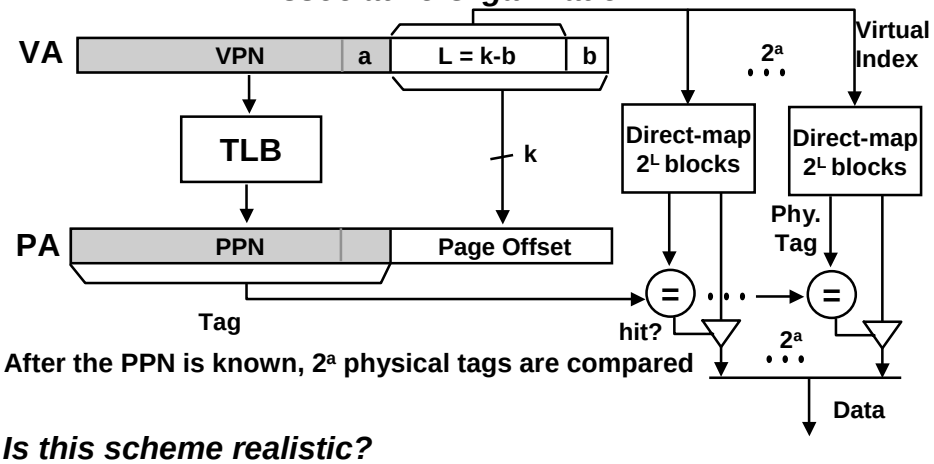
⇒ ***aliasing problems***

Aliases can be avoided in hardware by using

1. **Set Associative L1 or**
2. **Cache mechanisms of L2**

Virtual-Index Physical-Tag Caches

Associative Organization



After the PPN is known, 2^a physical tags are compared

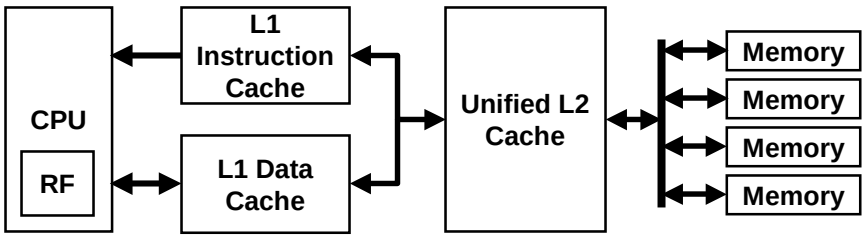
Is this scheme realistic?

Consider 4-Kbyte pages and caches with 32-byte blocks

32-Kbyte cache $\Rightarrow 2^a = 8$

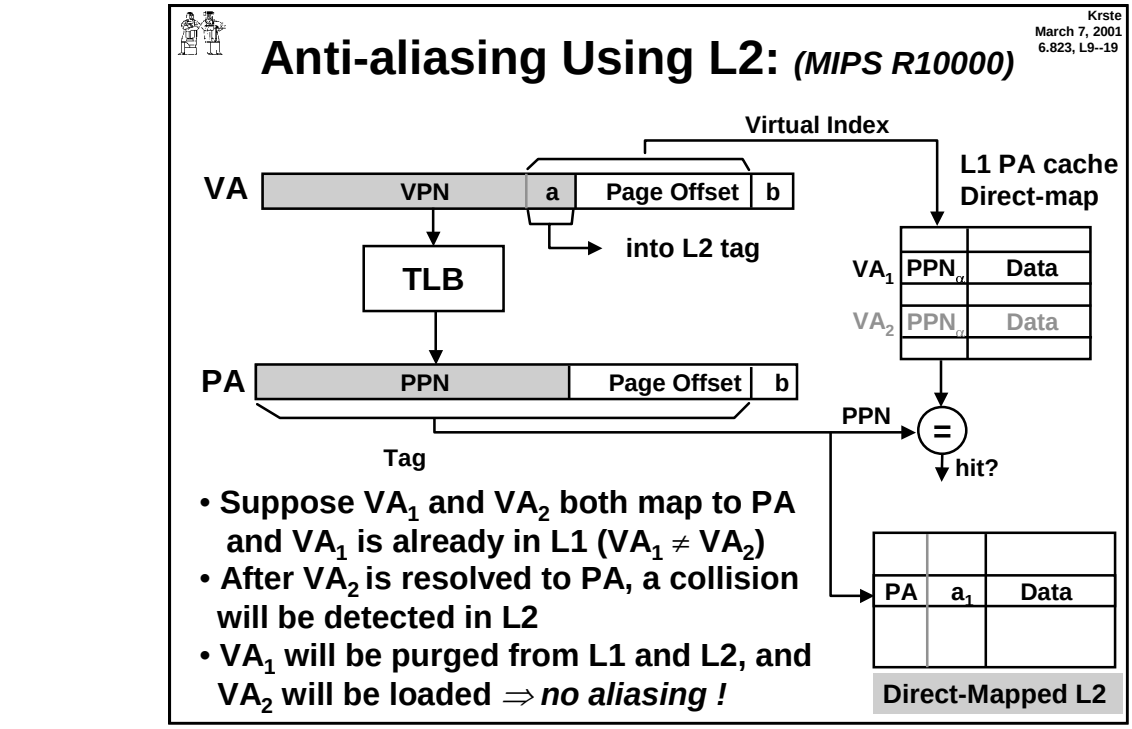
4-Mbyte cache $\Rightarrow 2^a = 1024$ ***No !***

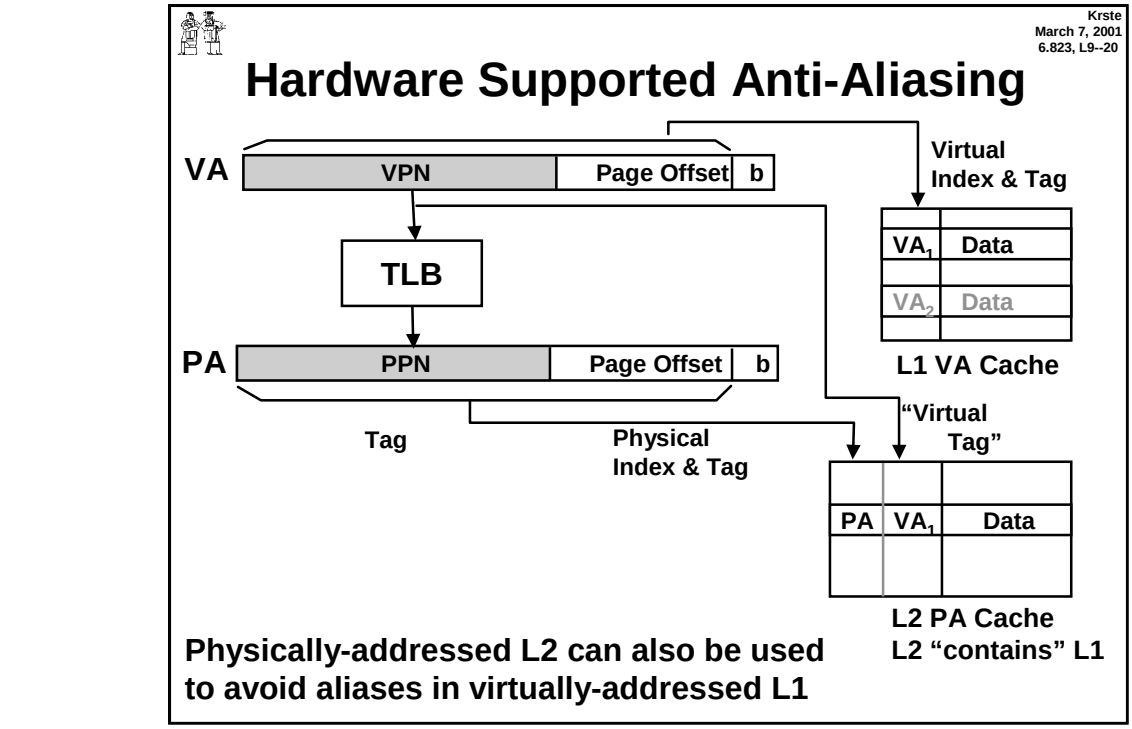
Second Level Cache



Usually a common L2 cache backups both Instruction and Data L1 caches

L2 is "inclusive" of both Instruction and Data caches


Krste
March 7, 2001
6.823, L9--19
Anti-aliasing Using L2: (MIPS R10000)
Virtual Index
L1 PA cache
Direct-map
VA
VPN
a
Page Offset
b
into L2 tag
TLB
PA
PPN
Page Offset
b
Tag
VA1
PPNa
Data
VA2
PPNa
Data
PPN
=
hit?
PA
a1
Data
Direct-Mapped L2
• Suppose VA1 and VA2 both map to PA and VA1 is already in L1 (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 !


Krste
March 7, 2001
6.823, L9--20
Hardware Supported Anti-Aliasing
VA
VPN
Page Offset
b
Virtual
Index & Tag
TLB
PA
PPN
Page Offset
b
VA1
Data
VA2
Data
L1 VA Cache
Tag
Physical
Index & Tag
"Virtual
Tag"
PA
VA1
Data
L2 PA Cache
L2 "contains" L1
Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1

Page Fault Handler

When the referenced page is not in DRAM:

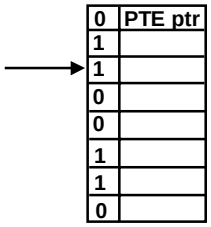
- **The missing page is located (or created)**
- **It is brought in from disk, and page table is updated**
  (A different job may start running on the CPU while the first job waits for the requested page to be read from disk)
- **If no free pages are left, a page is swapped out**
  - generally, pseudo-*Least Recently Used (LRU) page replacement* policy is used

Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the operating system *(untranslated addressing mode is essential to allow kernel to access page tables)*

Implementation of Pseudo-LRU

A page-frame table for the whole system is maintained in software

1: *recently referenced*
0: *not recently referenced*



The table is searched sequentially for a page to be replaced. The search begins from where the last one had ended.

If the entry is

1 ⇒ mark it 0, mark the page as inaccessible in the PTE
(0 is changed to 1 by the protection-fault handler on the next reference)

0 ⇒ the search terminates and returns the PPN
(each new entry starts with a marking of 1)

Pages are automatically taken away from inactive jobs

Swapping a Page of a Page Table

A PTE in primary memory contains primary or secondary memory addresses

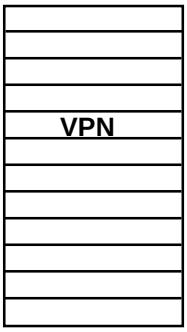
A PTE in secondary memory contains *only* secondary memory addresses

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

Why?_____

ATLAS Revisited

PAR's



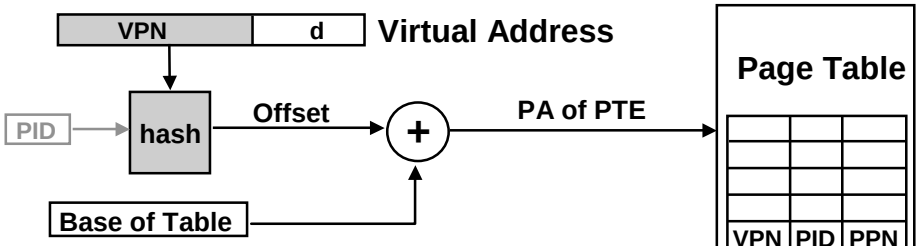
- **One PAR for each physical page**
- **PAR's contain the VPN's of the pages resident in primary memory**

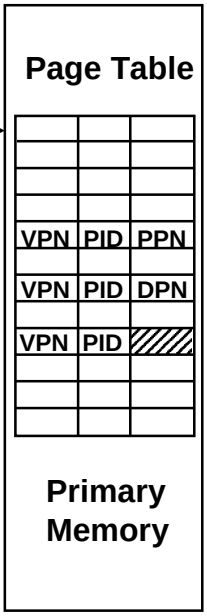
Advantage: **The size is proportional to the size of the primary memory**

Disadvantage: **The table needs to be searched for a given virtual address**

⇒ *associative addressing (can be approximated by hashing)*

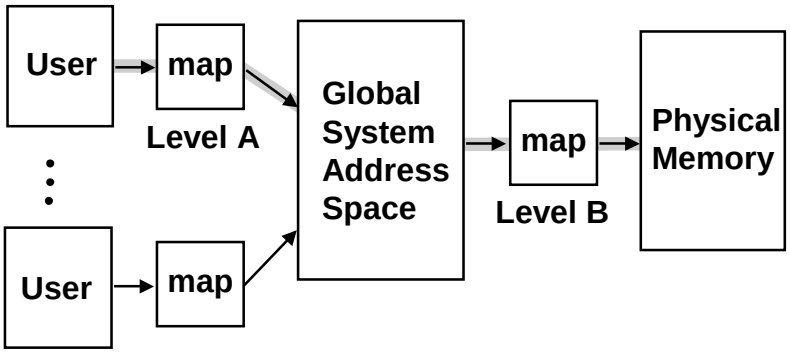
Hashed Page Table



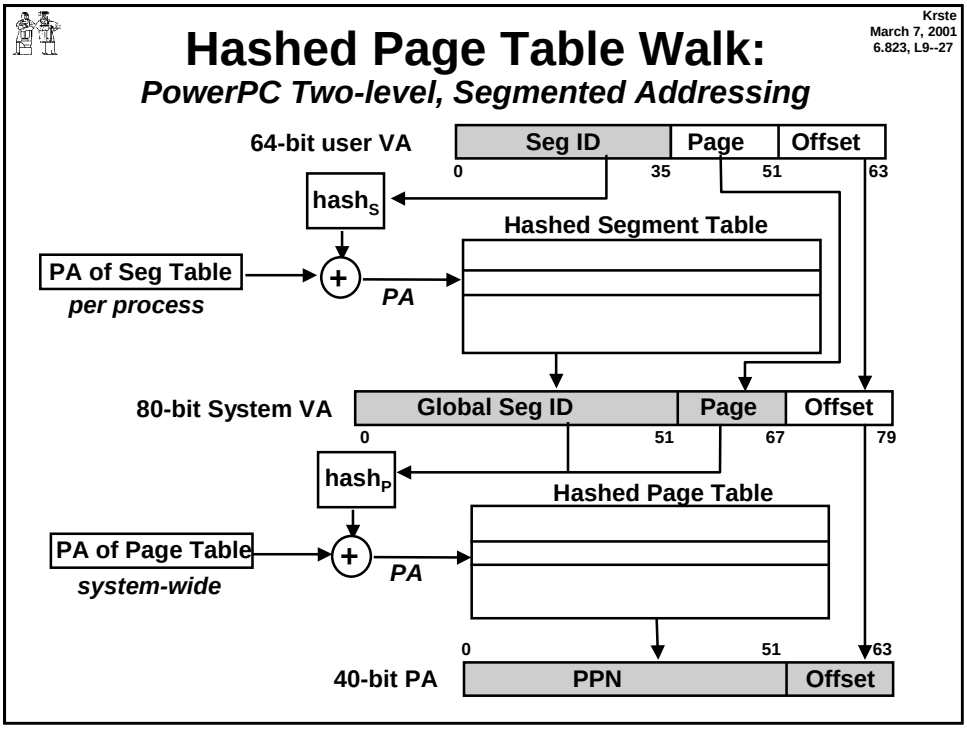


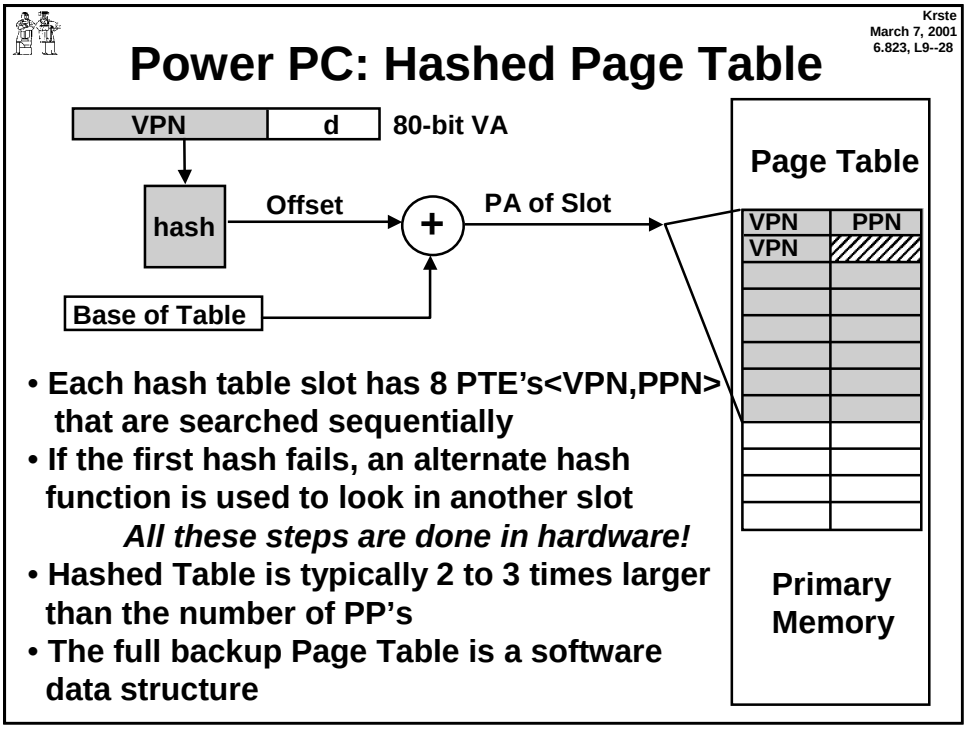
- Hashed Page Table is typically 2 to 3 times larger than the number of PPN's to reduce collision probability
- It can also contain DPN's for some non-resident pages *(not common)*
- If a translation cannot be resolved in this table then the software consults a data structure that has an entry for every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

Global System Address Space



- Level A maps users' address spaces into the global space providing privacy, protection, sharing etc.
- Level B provides demand-paging for the large global system address space
- Level A and Level B translations may be kept in separate TLB's


Krste
March 7, 2001
6.823, L9--27
Hashed Page Table Walk:
PowerPC Two-level, Segmented Addressing
64-bit user VA
Seg ID
Page
Offset
0
35
51
63
hash$_S$
Hashed Segment Table
PA of Seg Table
per process
+
PA
80-bit System VA
Global Seg ID
Page
Offset
0
51
67
79
hash$_P$
Hashed Page Table
PA of Page Table
system-wide
+
PA
0
51
63
40-bit PA
PPN
Offset


Krste
March 7, 2001
6.823, L9--28
Power PC: Hashed Page Table
VPN
d
80-bit VA
hash
Offset
+
PA of Slot
Base of Table
Page Table
VPN
PPN
VPN
Primary Memory
• Each hash table slot has 8 PTE's<VPN,PPN> that are searched sequentially
• If the first hash 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 PP's
• The full backup Page Table is a software data structure

Virtual Memory Use Today

- **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 (Crays: base&bound, Japanese: 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)**
  - **More difficult to implement restartable vector instructions**
- **Most embedded processors and DSPs provide only absolute addressing**
  - **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**