

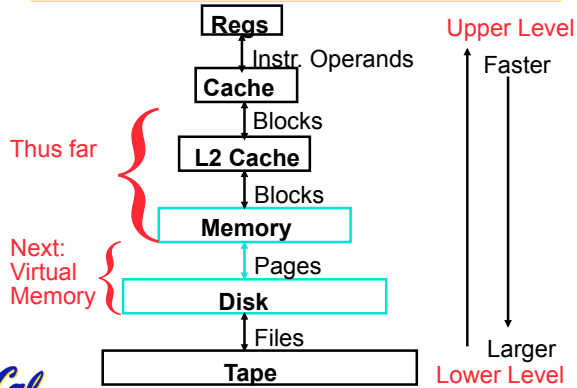


**Review**

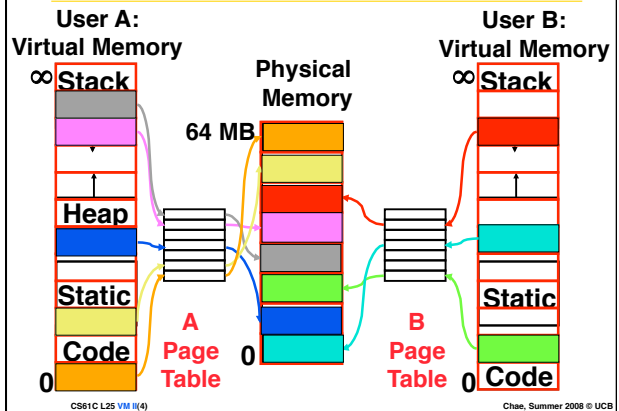
- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each user vs. tag/data in cache
  - TLB is cache of Virtual→Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well



**Another View of the Memory Hierarchy**

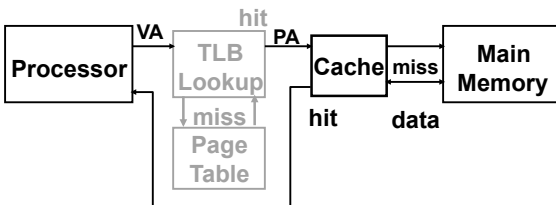


**Paging/Virtual Memory Multiple Processes**



**Translation Look-Aside Buffers (TLBs)**

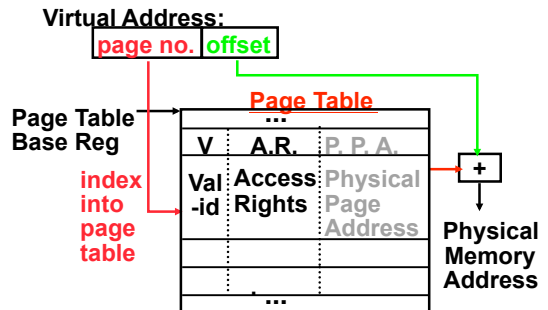
- TLBs usually small, typically 128 - 256 entries
- Like any other cache, the TLB can be direct mapped, set associative, or fully associative



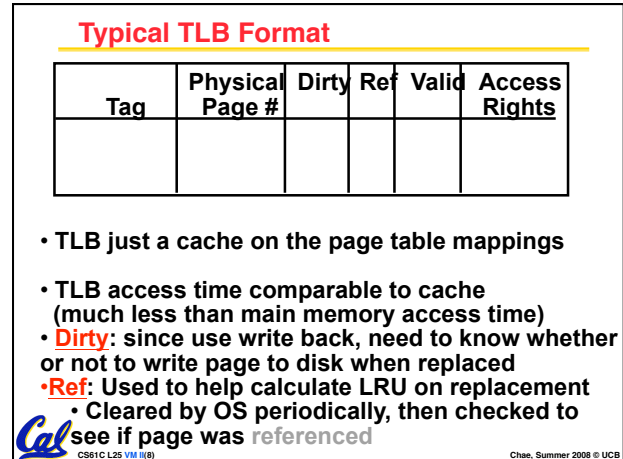
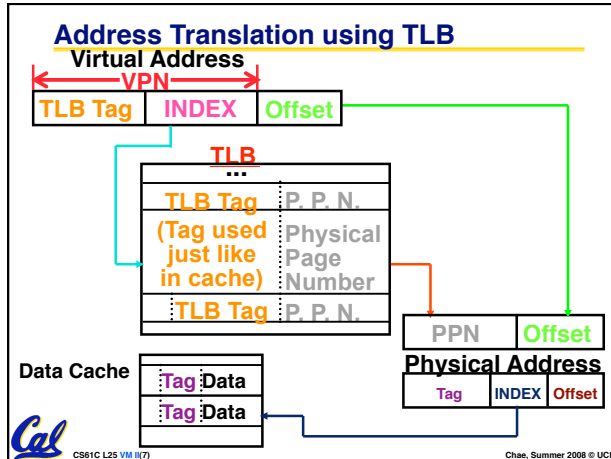
On TLB miss, get page table entry from main memory



**Review Address Mapping: Page Table**



Page Table located in physical memory



- ### What if not in TLB?
- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
  - Option 2: Hardware **traps** to OS, up to OS to decide what to do
    - MIPS follows Option 2: Hardware knows nothing about page table
- Cal CS61C L25 VM (9) Chaе, Summer 2008 © UCБ

- ### What if the data is on disk?
- We load the page off the disk into a free block of memory, using a DMA transfer (Direct Memory Access – special hardware support to avoid processor)
    - Meantime we switch to some other process waiting to be run
  - When the DMA is complete, we get an **interrupt** and update the process's page table
    - So when we switch back to the task, the desired data will be in memory
- Cal CS61C L25 VM (10) Chaе, Summer 2008 © UCБ

- ### What if we don't have enough memory?
- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
    - If clean (disk copy is up-to-date), just overwrite that data in memory
    - We chose the page to evict based on replacement policy (e.g., LRU)
  - And update that program's page table to reflect the fact that its memory moved somewhere else
  - If continuously swap between disk and memory, called **Thrashing**
- Cal CS61C L25 VM (11) Chaе, Summer 2008 © UCБ

- ### Why Translation Lookaside Buffer (TLB)?
- Paging is most popular implementation of virtual memory (vs. base/bounds)
  - Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection / indirection
  - Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast
- Cal CS61C L25 VM (12) Chaе, Summer 2008 © UCБ

### Three Advantages of Virtual Memory

#### 1) Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program (“**Working Set**”) must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later



### Three Advantages of Virtual Memory

#### 2) Protection:

- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more “viruses” under Microsoft Windows
- Special Mode in processor (“Kernel mode”) allows processor to change page table/TLB

#### 3) Sharing:

- Can map same physical page to multiple users (“Shared memory”)



### Peer Instruction

- A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM
- B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS
- C. VM helps both with security and cost

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT



### Peer Instruction Answer

- A. Locality is important yet different for cache and virtual memory (VM). Temporal locality for caches but spatial locality for VM
- B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS
- C. VM helps both with security and cost

**FALSE**

**TRUE**

**TRUE**

- A. No. Both for VM and cache
- B. Yes. TLB SW (MIPS) or HW (\$ HW, Page table OS)

- C. Yes. Protection and a bit smaller memory

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT



### Administrivia

- Proj3 out now, due Wednesday 8/6
  - Will be hand graded in person, signups will be posted tonight
    - Maybe early checkoffs starting tomorrow
- Proj4 out soon. Find a partner.
- Final 8/14 – 9:30-12:30pm in 105 North Gate



We're done with new material

Let's now review w/ Questions



## 4 Qs for any tier in the Memory Hierarchy

- Q1: Where can a block be placed?
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- Q2: How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)
- Q3: Which block is replaced on a miss?
  - Least recently used (LRU)
  - Random
- Q4: How are writes handled?
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be "dirty", must have dirty bit)

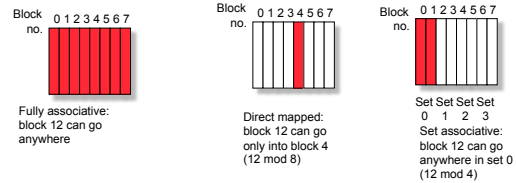


CS61C L25 VM (19)

Chae, Summer 2008 © UCB

## Q1: Where is block placed? (upper level)

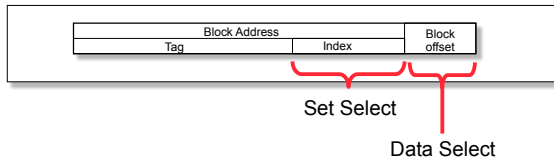
- Block #12 placed in 8 block cache:
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping =  $\text{Block \#} \bmod \# \text{ of Sets}$



CS61C L25 VM (20)

Chae, Summer 2008 © UCB

## Q2: How is a block found? (upper level)



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag



CS61C L25 VM (21)

Chae, Summer 2008 © UCB

## Q3: Which block replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

### Miss Rates

Size	Associativity: 2-way		4-way		8-way	
	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%



CS61C L25 VM (22)

Chae, Summer 2008 © UCB

## Q4: What to do on a write hit?

- **Write-through**
  - update the word in cache block and corresponding word in memory
- **Write-back**
  - update word in cache block
  - allow memory word to be "stale"
  - => add 'dirty' bit to each line indicating that memory is updated when block is replaced
  - => OS flushes cache before I/O !!!
- Performance trade-offs?



CS61C L25 VM (23)

Chae, Summer 2008 © UCB

WT: read misses cannot result in writes  
WB: no writes of repeated writes

## Virtual Memory Overview (1/4)

- User program view of memory:
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program
- Reality:
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time



CS61C L25 VM (24)

Chae, Summer 2008 © UCB

Chae, Summer 2008 © UCB

### Virtual Memory Overview (2/4)

- Virtual memory provides:
  - illusion of contiguous memory
  - all programs starting at same set address
  - illusion of ~ infinite memory ( $2^{32}$  or  $2^{64}$  bytes)
  - protection



### Virtual Memory Overview (3/4)

- Implementation:
  - Divide memory into “chunks” (pages)
  - Operating system controls page table that maps virtual addresses into physical addresses
  - Think of memory as a cache for disk
  - TLB is a cache for the page table



### Virtual Memory Overview (4/4)

- Let’s say we’re fetching some data:
  - Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check page table (in memory)
      - Page table hit: fetch translation
      - Page table miss: page fault, fetch page from disk to memory, return translation to TLB
  - Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory

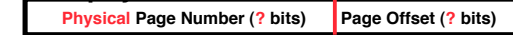


### Question (1/3)

- 40-bit virtual address, 16 KB page



- 36-bit physical address



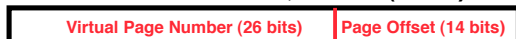
- Number of bits in Virtual Page Number/ Page offset, Physical Page Number/Page offset?

- 1: 22/18 (VPN/PO), 22/14 (PPN/PO)
- 2: 24/16, 20/16
- 3: 26/14, 22/14
- 4: 26/14, 26/10
- 5: 28/12, 24/12

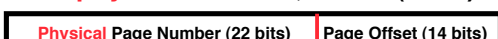


### (1/3) Answer

- 40- bit virtual address, 16 KB ( $2^{14}$  B)



- 36- bit physical address, 16 KB ( $2^{14}$  B)



- Number of bits in Virtual Page Number/ Page offset, Physical Page Number/Page offset?

- 1: 22/18 (VPN/PO), 22/14 (PPN/PO)
- 2: 24/16, 20/16
- 3: 26/14, 22/14
- 4: 26/14, 26/10
- 5: 28/12, 24/12

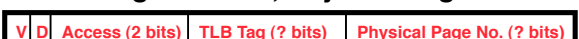


### Question (2/3): 40b VA, 36b PA

- 2-way set-assoc. TLB, 512 entries, 40b VA:



- TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Virtual Page Number, Physical Page Number



- Number of bits in TLB Tag / Index / Entry?

- 1: 12 / 14 / 38 (TLB Tag / Index / Entry)
- 2: 14 / 12 / 40
- 3: 18 / 8 / 44
- 4: 18 / 8 / 58



### (2/3) Answer

- 2-way set-associative data cache, 256 ( $2^8$ ) “sets”, 2 TLB entries per set  $\Rightarrow$  8 bit index



Virtual Page Number (26 bits)

- TLB Entry: Valid bit, Dirty bit, Access Control (2 bits), Virtual Page Number, Physical Page Number



1: 12 / 14 / 38 (TLB Tag / Index / Entry)

2: 14 / 12 / 40

3: 18 / 8 / 44

4: 18 / 8 / 58



CS61C L25 VM II (31)

Chae, Summer 2008 © UCB

### Question (3/3)

- 2-way set-associative 64KB data cache, 64B block



Physical Page Address (36 bits)

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + ? bits of Data



- Number of bits in Data cache Tag / Index / Offset / Entry?

1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)

2: 20 / 10 / 6 / 86

3: 20 / 10 / 6 / 534

4: 21 / 9 / 6 / 87

5: 21 / 9 / 6 / 535



CS61C L25 VM II (32)

Chae, Summer 2008 © UCB

### (3/3) Answer

- 2-way set-associative data cache, 64K/1K ( $2^{10}$ ) “sets”, 2 entries per sets  $\Rightarrow$  9 bit index



Physical Page Address (36 bits)

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + 64 Bytes of Data



1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)

2: 20 / 10 / 6 / 86

3: 20 / 10 / 6 / 534

4: 21 / 9 / 6 / 87

5: 21 / 9 / 6 / 535



CS61C L25 VM II (33)

Chae, Summer 2008 © UCB

### And in Conclusion...

- Virtual memory to Physical Memory Translation too slow?

- Add a cache of Virtual to Physical Address Translations, called a **TLB**

- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

- Virtual Memory allows protected sharing of memory between processes with less swapping to disk



CS61C L25 VM II (34)

Chae, Summer 2008 © UCB