
CS 162

Virtual Memory

Instructor: Jun Yang
Computer Science and Engineering Department

A Real Problem

- ❑ What if you wanted to run a program that needs more memory than you have?

Virtual Memory (and Indirection)



□ Finally, we get to Virtual Memory!

- We'll talk about the motivations for virtual memory
- We'll talk about how it is implemented
- Lastly, we'll talk about how to make virtual memory fast: Translation Lookaside Buffers (TLBs).

A Real Problem

- ❑ **What if you wanted to run a program that needs more memory than you have?**
 - **You could store the whole program on disk, and use memory as a cache for the data on disk. This is one feature of virtual memory.**
 - **Before virtual memory, programmers had to manually manage loading “overlays” (chunks of instructions & data) off disk before they were used. This is an incredibly tedious, not to mention error-prone, process.**

More Real Problems

- ❑ **Running multiple programs at the same time brings up more problems.**
- 1. Even if each program fits in memory, running 10 programs might not.**
- 2. Multiple programs may want to store something at the same address.**
- 3. How do we protect one program's data from being read or written by another program?**

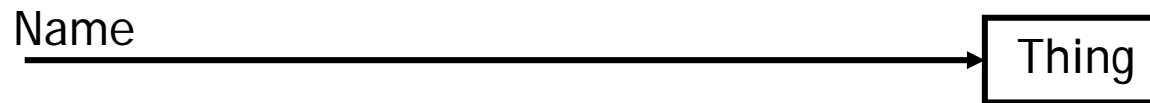
More Real Problems

- ❑ Running multiple programs at the same time brings up more problems.
- 1. Even if each program fits in memory, running 10 programs might not.
 - This is really the same problem as on the previous slide.
- 2. Multiple programs may want to store something at the same address.
 - I.e., what if both Program A and B want to use address 0x10000000 as the base of their stack?
 - It is impractical (if not impossible) to compile every pair of programs that could get executed together to use distinct sets of addresses.
- 3. How do we protect one program's data from being read or written by another program?

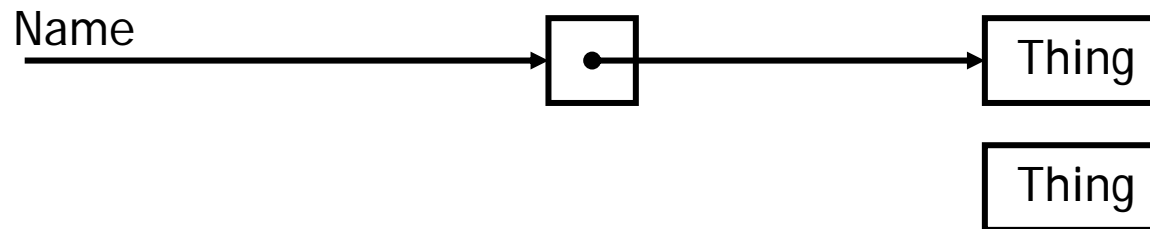
Indirection

- ❑ “Any problem in CS can be solved by adding a level of indirection”

- ❑ **Without Indirection**



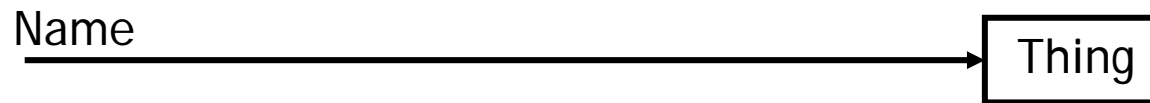
- ❑ **With Indirection**



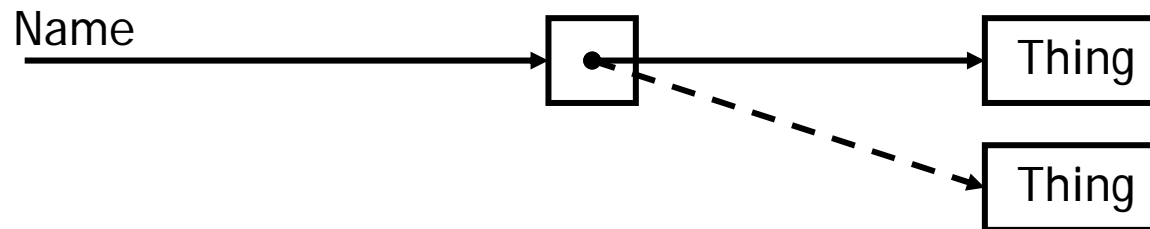
Indirection

- ❑ “Any problem in CS can be solved by adding a level of indirection”

- ❑ **Without Indirection**

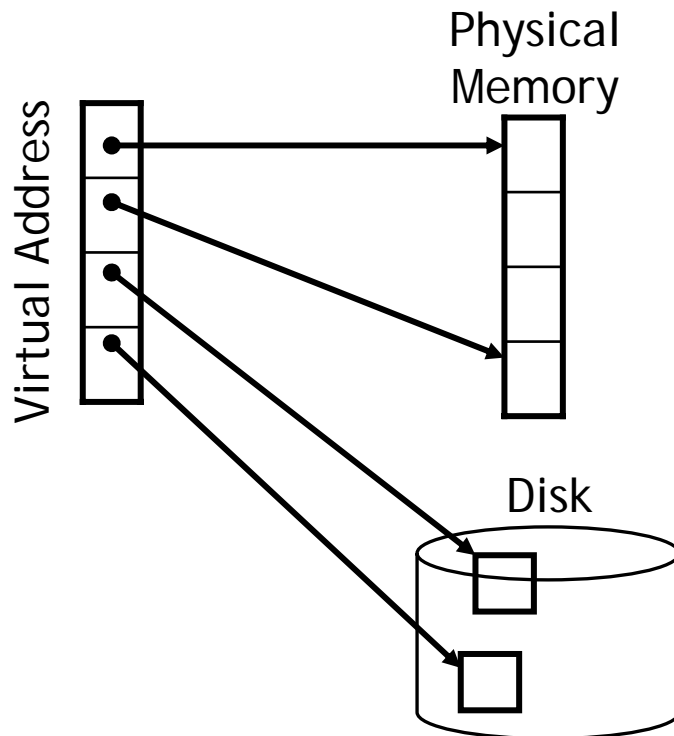


- ❑ **With Indirection**



Virtual Memory

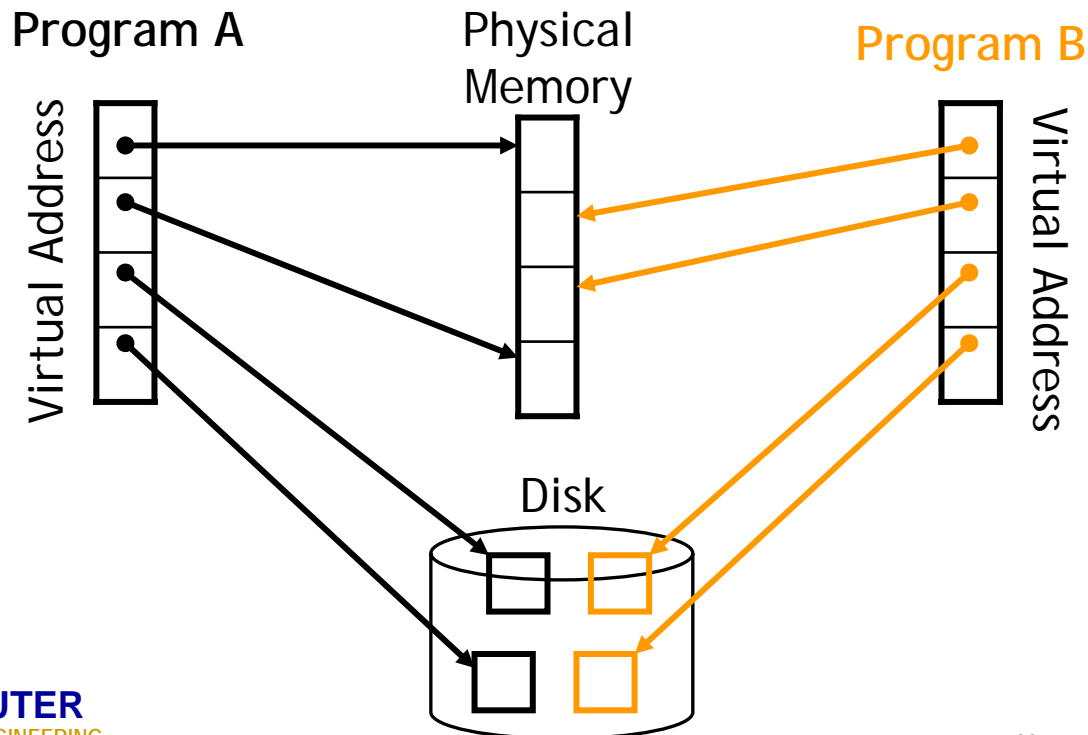
- We translate “virtual addresses” used by the program to “physical addresses” that represent places in the machine’s “physical” memory.
 - The word “translate” denotes a level of indirection



A virtual address can be mapped to either physical memory or disk.

Virtual Memory

- ❑ Because different processes will have different mappings from virtual to physical addresses, two programs can freely use the same virtual address.
- ❑ By allocating distinct regions of physical memory to A and B, they are prevented from reading/writing each others data.



Caching revisited

- ❑ Once the translation infrastructure is in place, the problem boils down to caching.
 - We want the size of disk, but the performance of memory.

- ❑ The design of virtual memory systems is really motivated by the high cost of accessing disk.
 - While memory latency is **~100** times that of cache, disk latency is **~100,000** times that of memory.
 - i.e., the miss penalty is a real whopper.

- ❑ Hence, we try to minimize the miss rate:
 - VM “pages” are much larger than cache blocks. Why?
 - A fully associative policy is used.
 - With approximate LRU

- ❑ Should a write-through or write-back policy be used?

Finding the right page

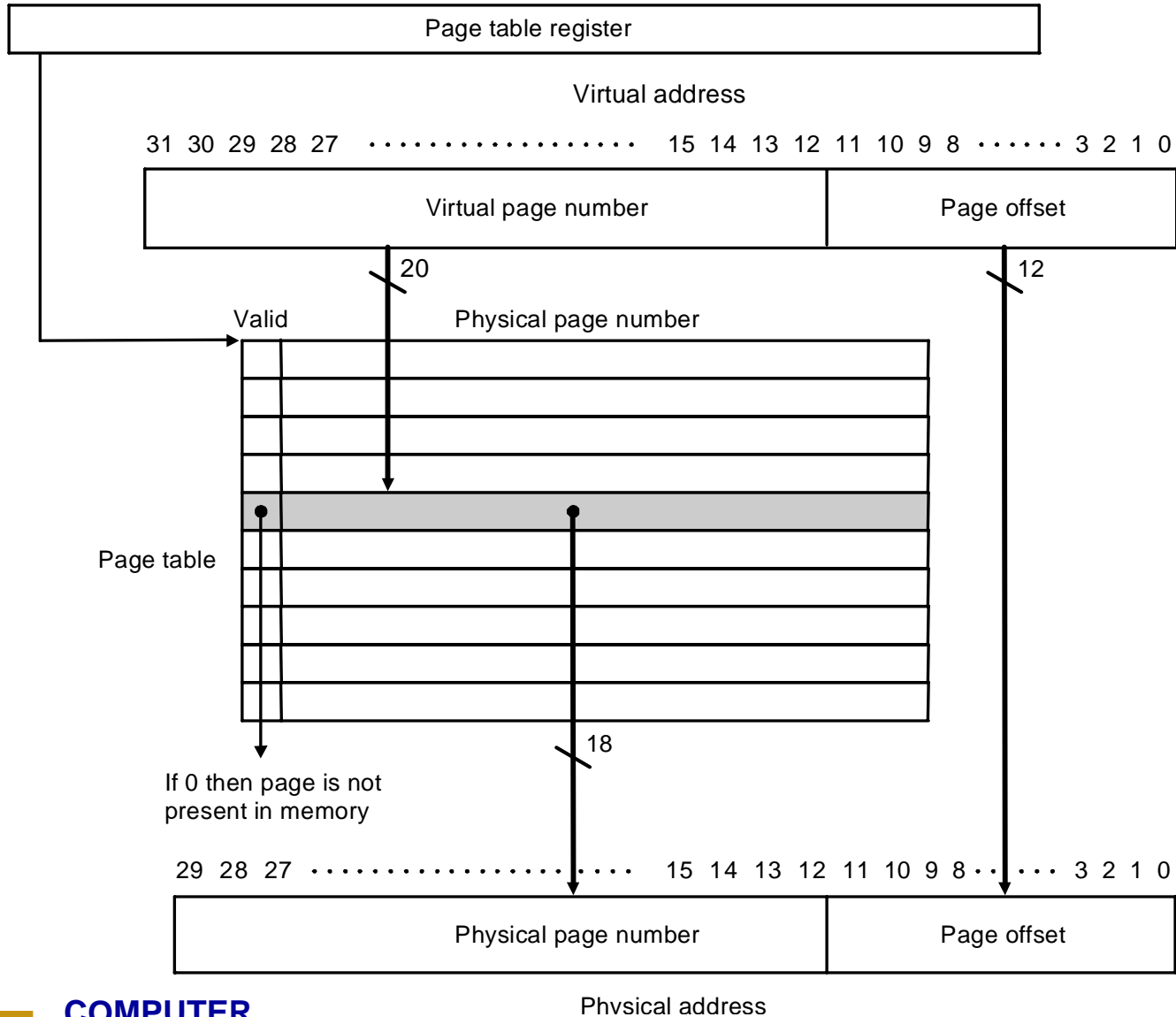
- ❑ If it is fully associative, how do we find the right page **without scanning all of memory?**

Finding the right page

- ❑ **If it is fully associative, how do we find the right page **without scanning all of memory**?**
 - Use an **index**, just like you would for a book.

- ❑ **Our index happens to be called the **page table**:**
 - **Each process has a separate page table**
 - A “page table register” points to the current process’s page table
 - **The page table is indexed with the **virtual page number (VPN)****
 - The VPN is all of the bits that aren’t part of the page offset.
 - **Each entry contains a valid bit, and a **physical page number (PPN)****
 - The PPN is concatenated with the page offset to get the physical address
 - **No tag is needed because the index is the full VPN.**

Page Table picture



How big is the page table?

❑ From the previous slide:

- Virtual page number is 20 bits.
- Physical page number is 18 bits + valid bit -> round up to 32 bits.

❑ How about for a 64b architecture?

Waitaminute!

- We've just replaced every memory access $\text{MEM}[\text{addr}]$ with:

$\text{MEM}[\text{MEM}[\text{MEM}[\text{MEM}[\text{PTBR} + \text{VPN1}] + \text{VPN2}] + \text{VPN3}] + \text{offset}]$

– *i.e.*, 4 memory accesses

- And **we haven't talked about the bad case yet** (*i.e.*, page faults)...

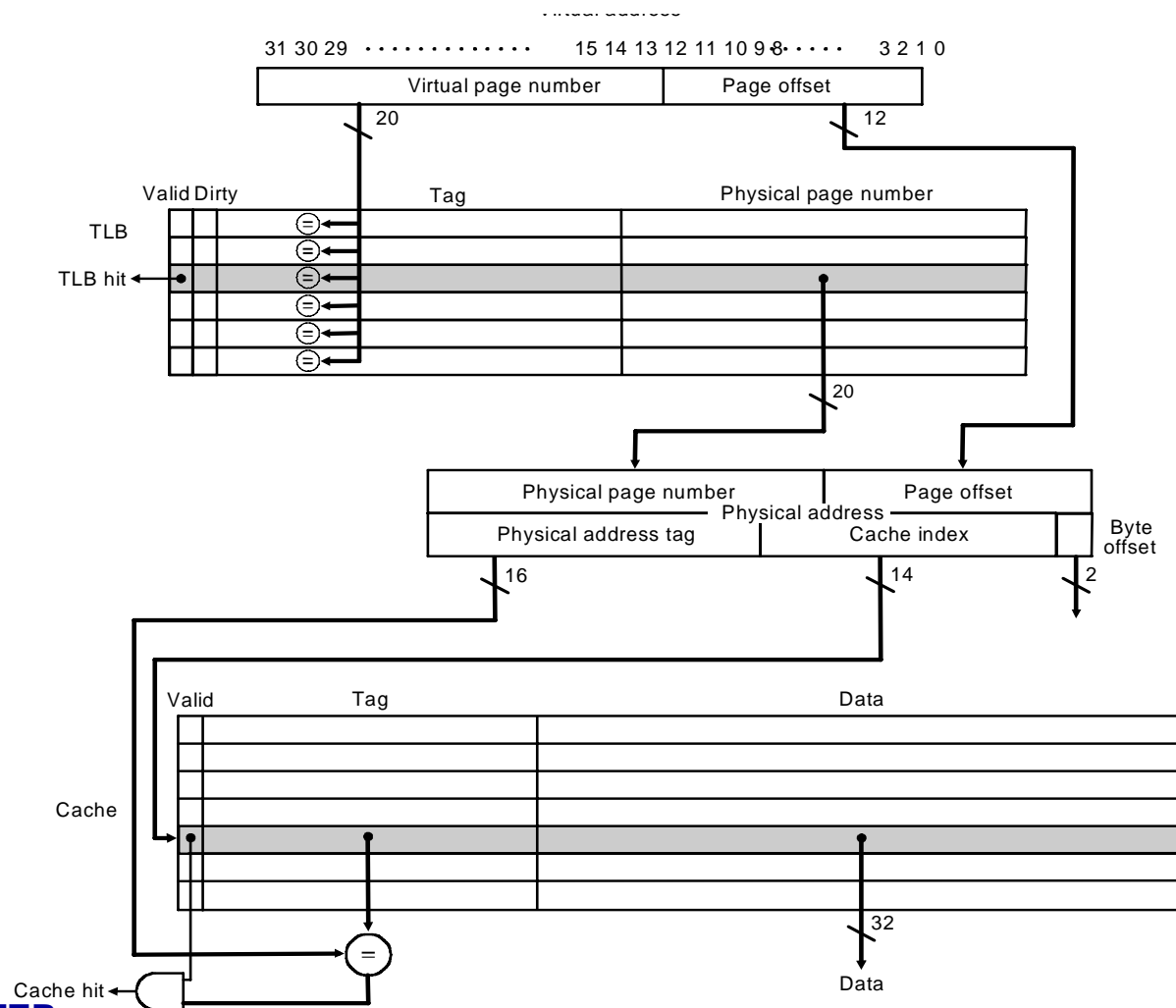
“Any problem in CS can be solved by adding a level of indirection”

– except too many levels of indirection...

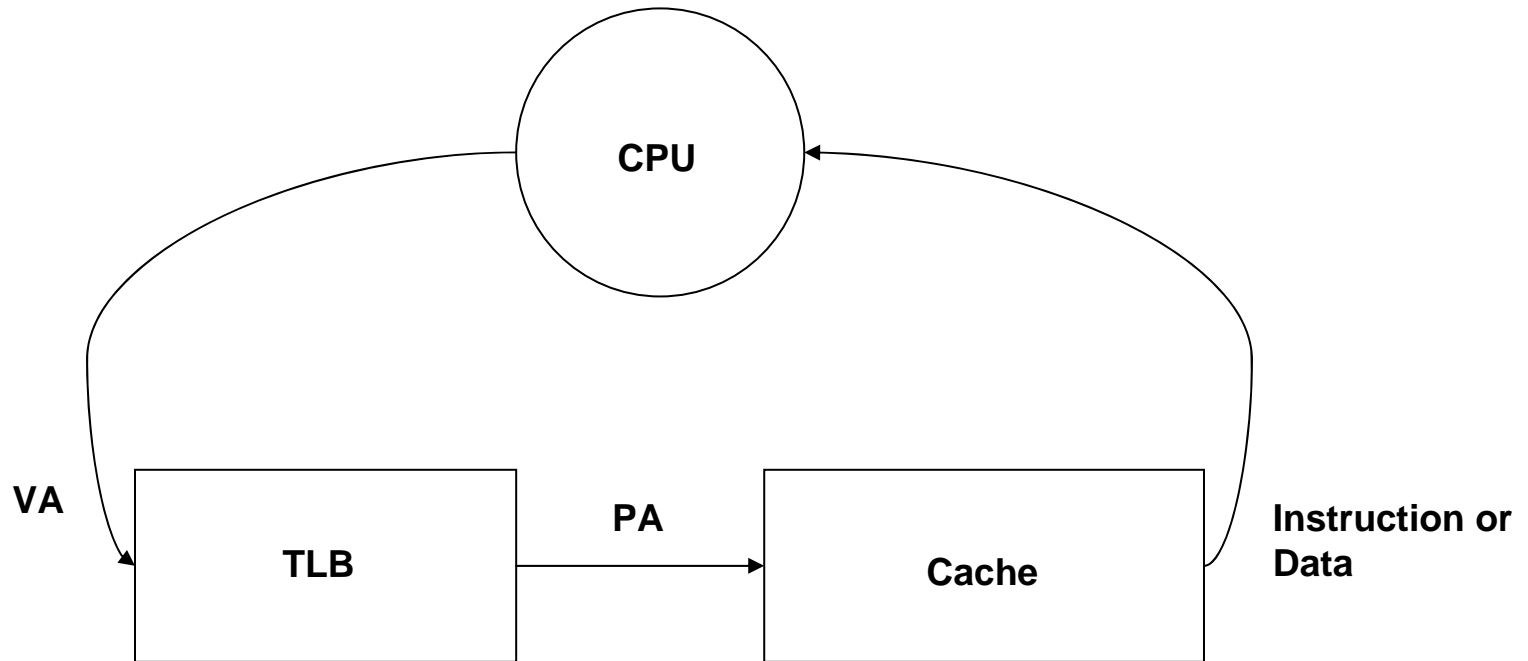
- How do we deal with too many levels of indirection?

Caching translations

- Virtual to Physical translations are cached in a Translation Lookaside Buffer (TLB).

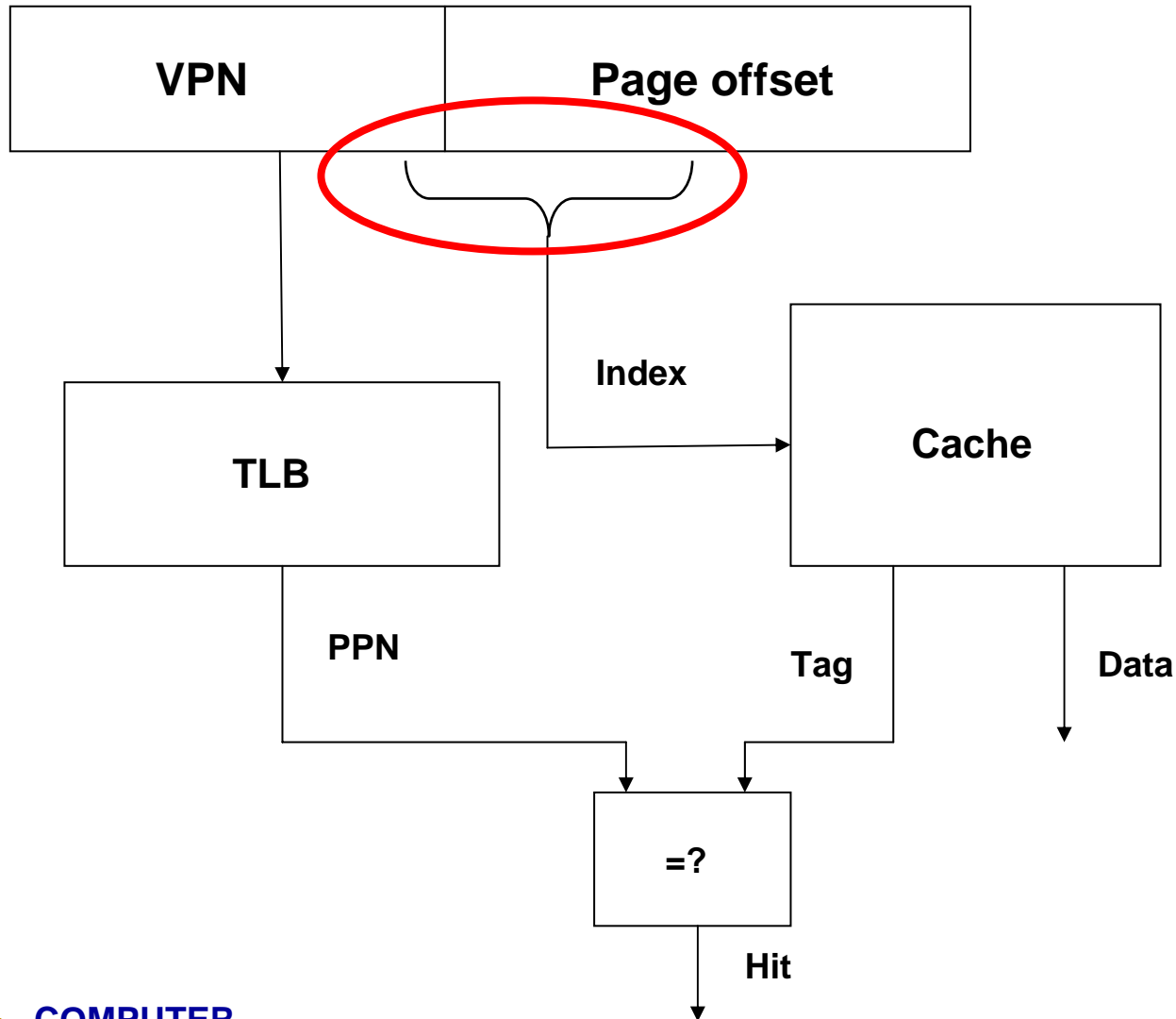


Path of memory access

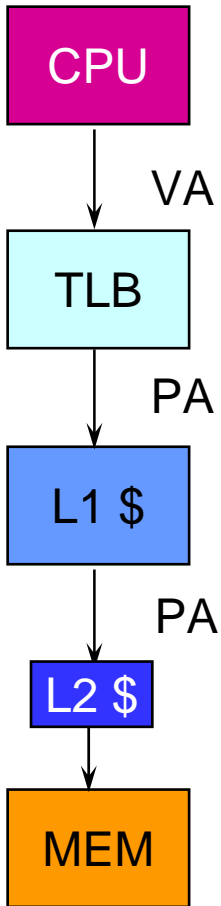


How to get translation out of the critical path?

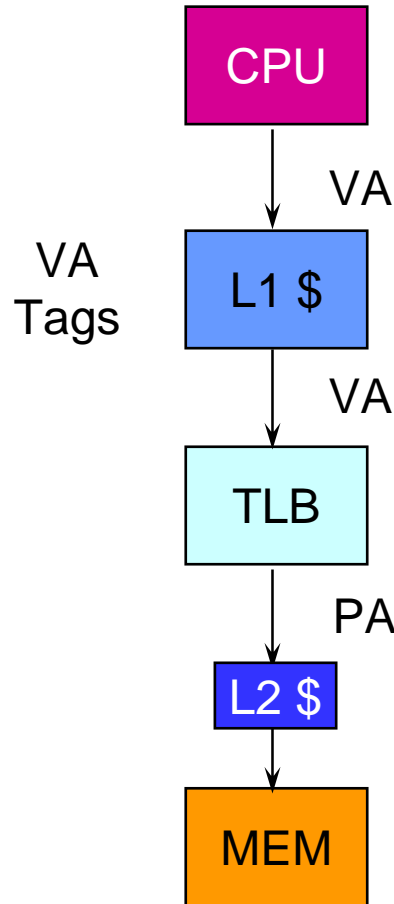
Virtually indexed physically tagged cache (VIPT)



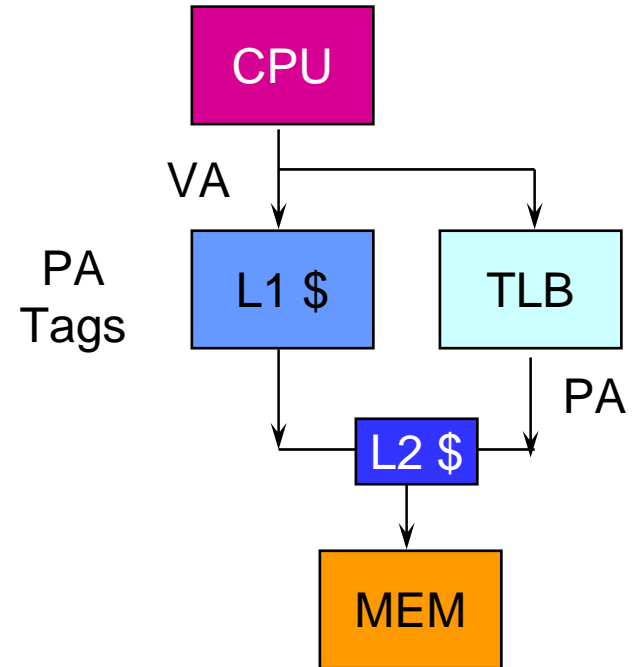
In general



Conventional Organization
PIPT



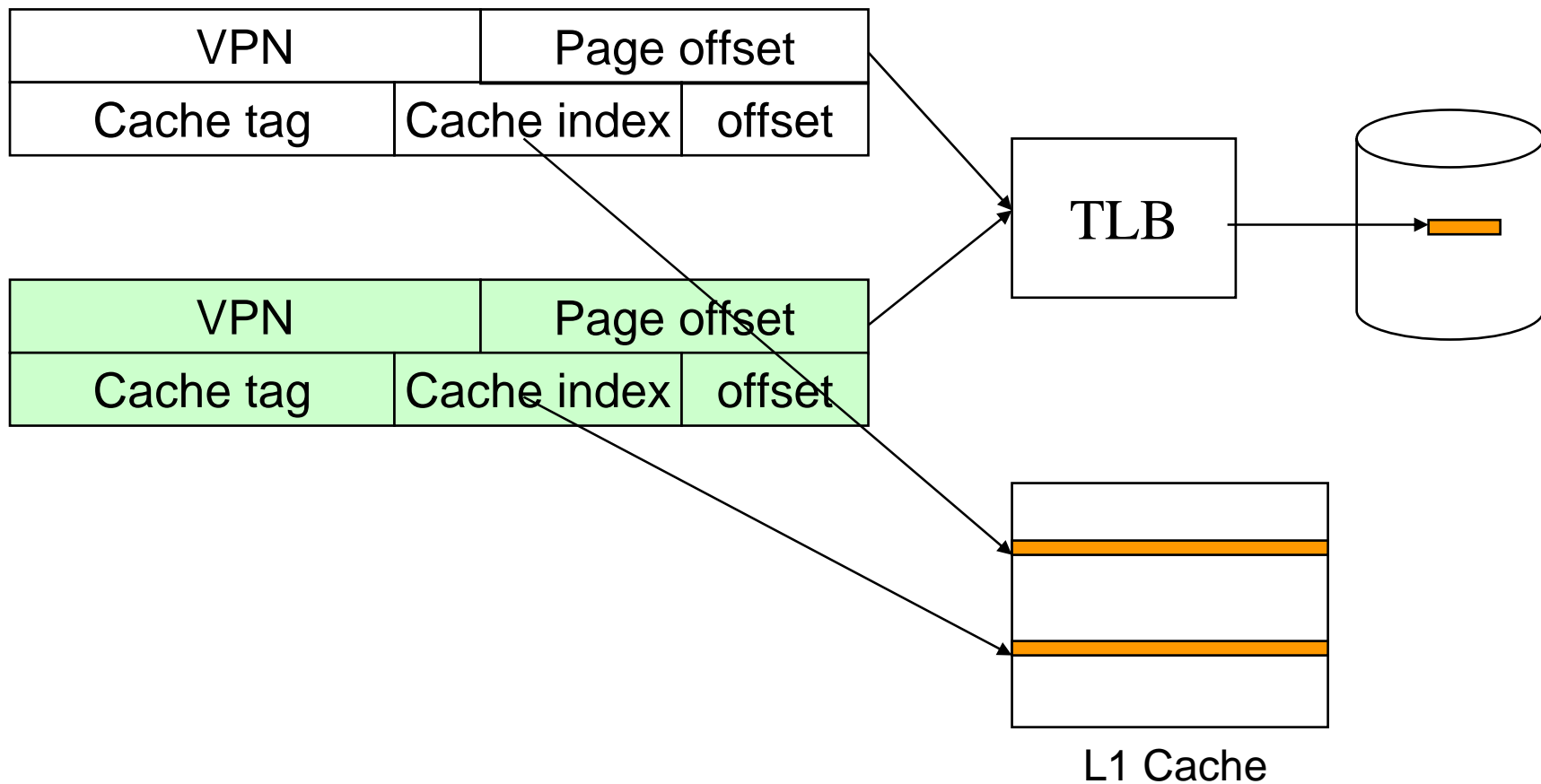
Virtually Addressed Cache
Translate only on miss
VIVT in L1



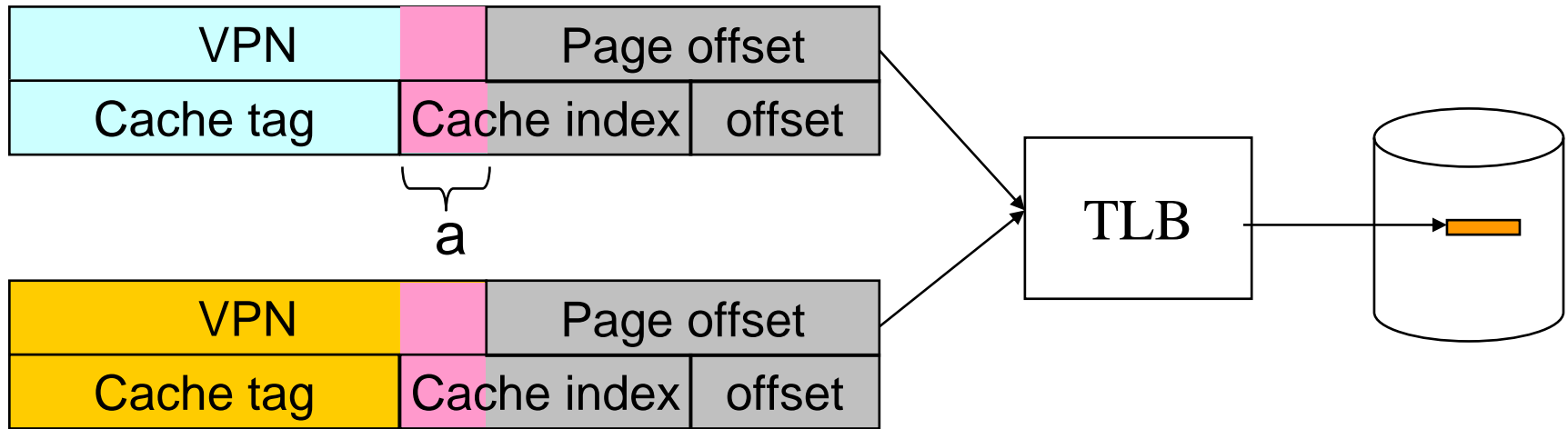
Overlap \$ access
with VA translation
VIPT

Synonym problem with virtually indexed cache

- ❑ **Synonym** — two different VA's are mapped to the same PA, but have two copies of the data in different locations.



Understanding synonyms



- ❑ If two VPNs do not differ in **a** then there is no synonym problem, since they will be indexed to the same set of a VIPT cache
- ❑ How to remove it?
 - Max number of sets = page size / cache line size
 - Ex: 4kB page, 32B line, max set = 128
 - Make physical index match virtual index – page coloring
 - ...

What about a TLB miss?

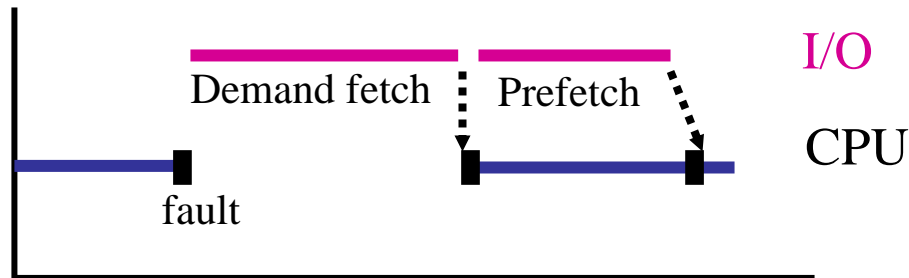
- ❑ **If we miss in the TLB, we need to “walk the page table”**
 - In MIPS, an exception is raised and software fills the TLB
 - In x86, a “hardware page table walker” fills the TLB

- ❑ **What if the page is not in memory?**
 - This situation is called a page fault.
 - The operating system will have to request the page from disk.
 - It will need to select a page to replace.
 - The O/S tries to approximate LRU
 - The replaced page will need to be written back if dirty.

Demand Paging

- ❑ Missing pages are loaded from disk into memory at *time of reference (on demand)*.

The alternative would be to prefetch into memory in anticipation of future accesses (need good predictions).



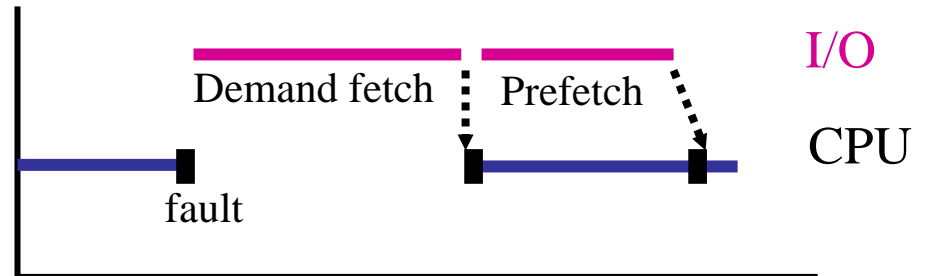
- ❑ Page fault occurs because *valid bit* in page table entry (PTE) is *off*. The OS:
 - allocates an empty frame*
 - initiates the read of the page from disk
 - updates the PTE when I/O is complete
 - restarts faulting process

* Placement and possible Replacement policies

Prefetching Issues

- **Pro: overlap of disk I/O and computation on resident pages. Hides latency of transfer.**

- **Need information to guide predictions**



- **Con: bad predictions**

- **Bad choice: a page that will never be referenced.**
- **Bad timing: a page that is brought in too soon**

Impacts:

- **taking up a frame that would otherwise be free.**
- **(worse) replacing a useful page.**
- **extra I/O traffic**

Memory Protection

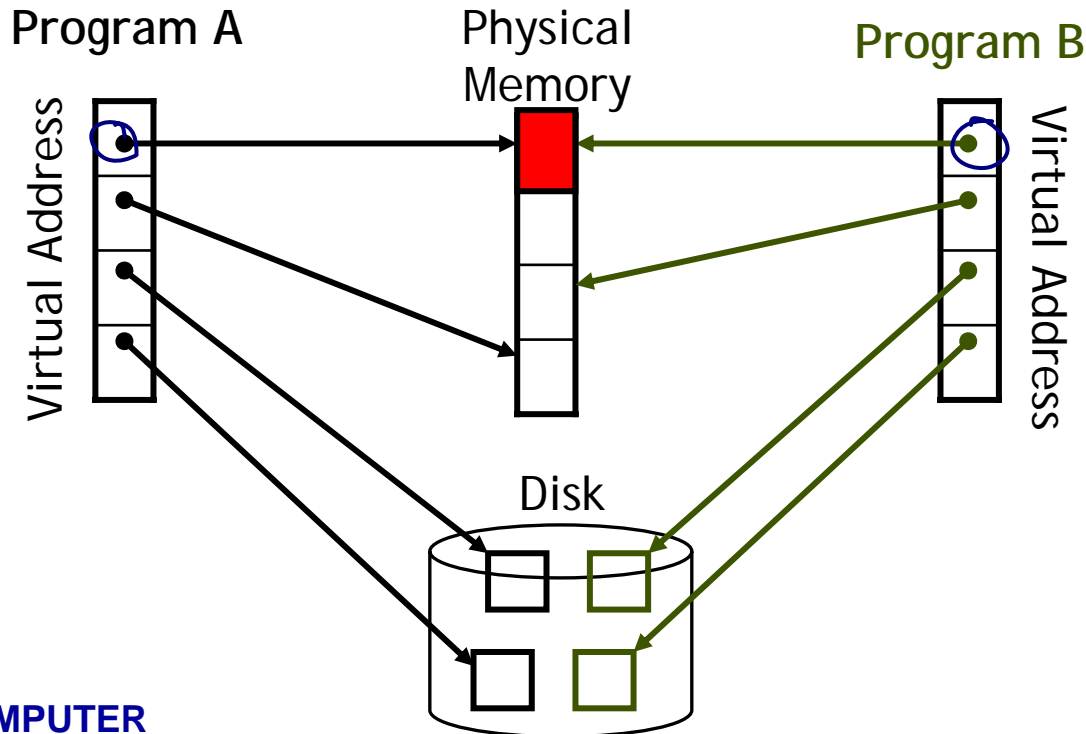
- ❑ In order to prevent one process from reading/writing another process's memory, we must ensure that a process cannot read/write each other's pages
 - Add PID tag into the page table

- ❑ Only the OS can change the virtual-to-physical translations.

- ❑ Typically, this is done by:
 - Having two processor modes: user & kernel.
 - Only the O/S runs in kernel mode
 - Only allowing kernel mode to write to the virtual memory state, e.g.,
 - The page table
 - The page table base pointer
 - The TLB

Sharing Memory

- ❑ Paged virtual memory enables sharing at the granularity of a page, by allowing two page tables to point to the same physical addresses.
- ❑ For example, if you run two copies of a program, the O/S will share the code pages between the programs.



Summary

❑ **Virtual memory is pure manna from heaven:**

- It means that we don't have to manage our own memory.
- It allows different programs to use the same memory.
- It provides protect between different processes.
- It allows controlled sharing between processes (albeit somewhat inflexibly).

❑ **The key technique is indirection:**

- Yet another classic CS trick you've seen in this class.
- Many problems can be solved with indirection.

❑ **Caching made a few cameo appearances, too:**

- Virtual memory enables using physical memory as a cache for disk.
- We used caching (in the form of the Translation Lookaside Buffer) to make Virtual Memory's indirection fast.

Reducing Cache Miss Penalty

1. Reduce Miss Penalty: Early Restart and Critical Word First

- ❑ Don't wait for full block to be loaded before restarting CPU
 - **Early restart**—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
 - **Critical Word First**—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- ❑ Generally useful only in large blocks,

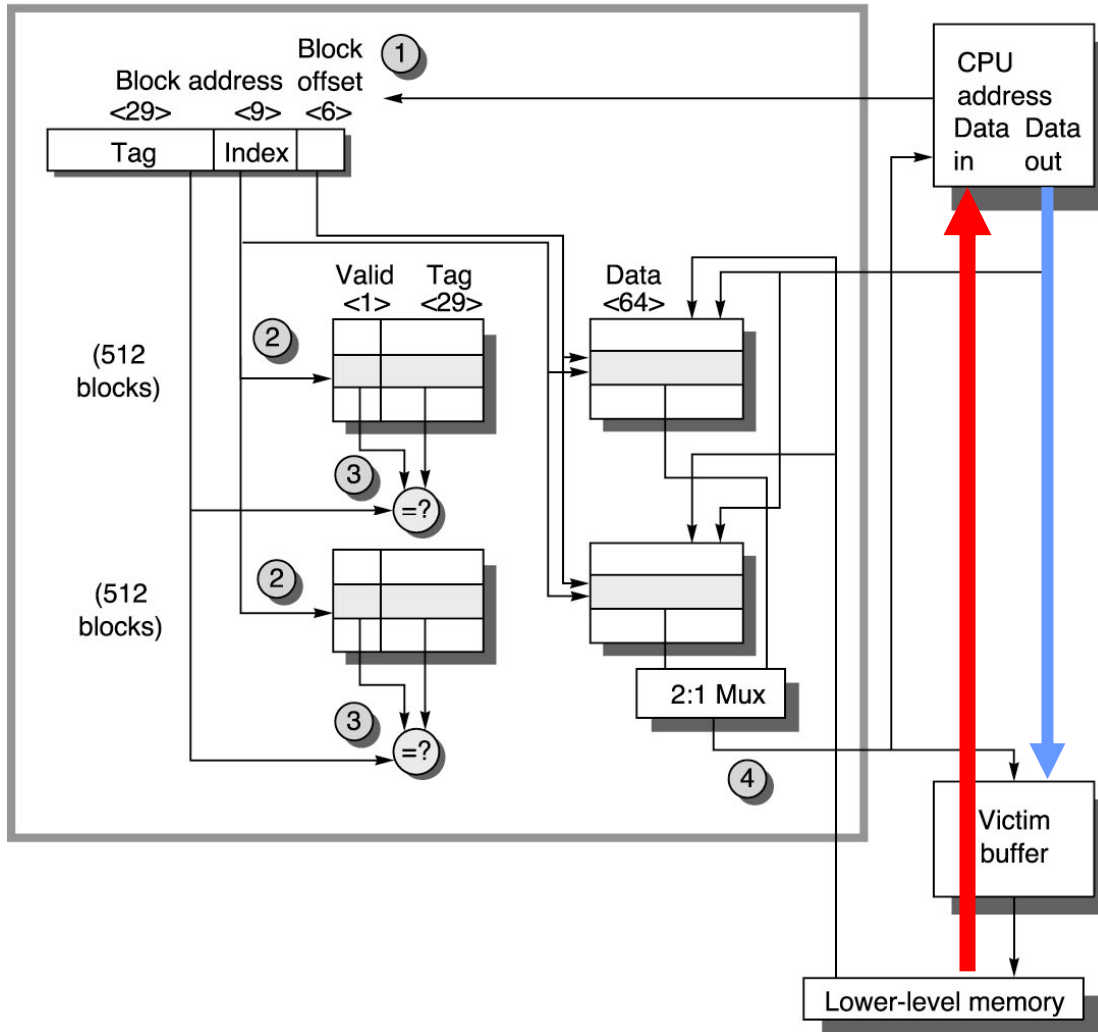


block

2. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- ❑ Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
 - requires F/E bits on registers or out-of-order execution
 - requires multi-bank memories
- ❑ “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- ❑ “hit under multiple miss” or “miss under miss” may further lower the effective miss penalty by overlapping multiple misses
 - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
 - Requires *multiple memory banks* (otherwise cannot support)
 - Penium Pro allows 4 outstanding memory misses

3. Reducing Miss Penalty: Read Priority over Write on Miss



- **Difference between loads and stores**
 - **Loads** – dependent instructions will wait for the results
 - **Stores** – end in the memory stage, no dependent instructions

3. Reducing Miss Penalty: Read Priority over Write on Miss

- ❑ **Write-through with write buffers offer RAW conflicts with main memory reads on cache misses**
 - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
 - Check write buffer contents before read; if no conflicts, let the memory access continue
- ❑ **Write-back also want buffer to hold misplaced blocks**
 - Read miss replacing dirty block
 - Normal: Write dirty block to memory, and then do the read
 - Instead copy the dirty block to a write buffer, then do the read, and then do the write
 - CPU stall less since restarts as soon as do read