Virtual Memory

Motivation:

- Each process would like to see its own, full, address space
- Clearly impossible to provide full physical memory for all processes
- Processes may define a large address space but use only a small part of it at any one time
- Processes would like their memory to be protected from access and modification by other processes
- The operating system needs to be protected from applications

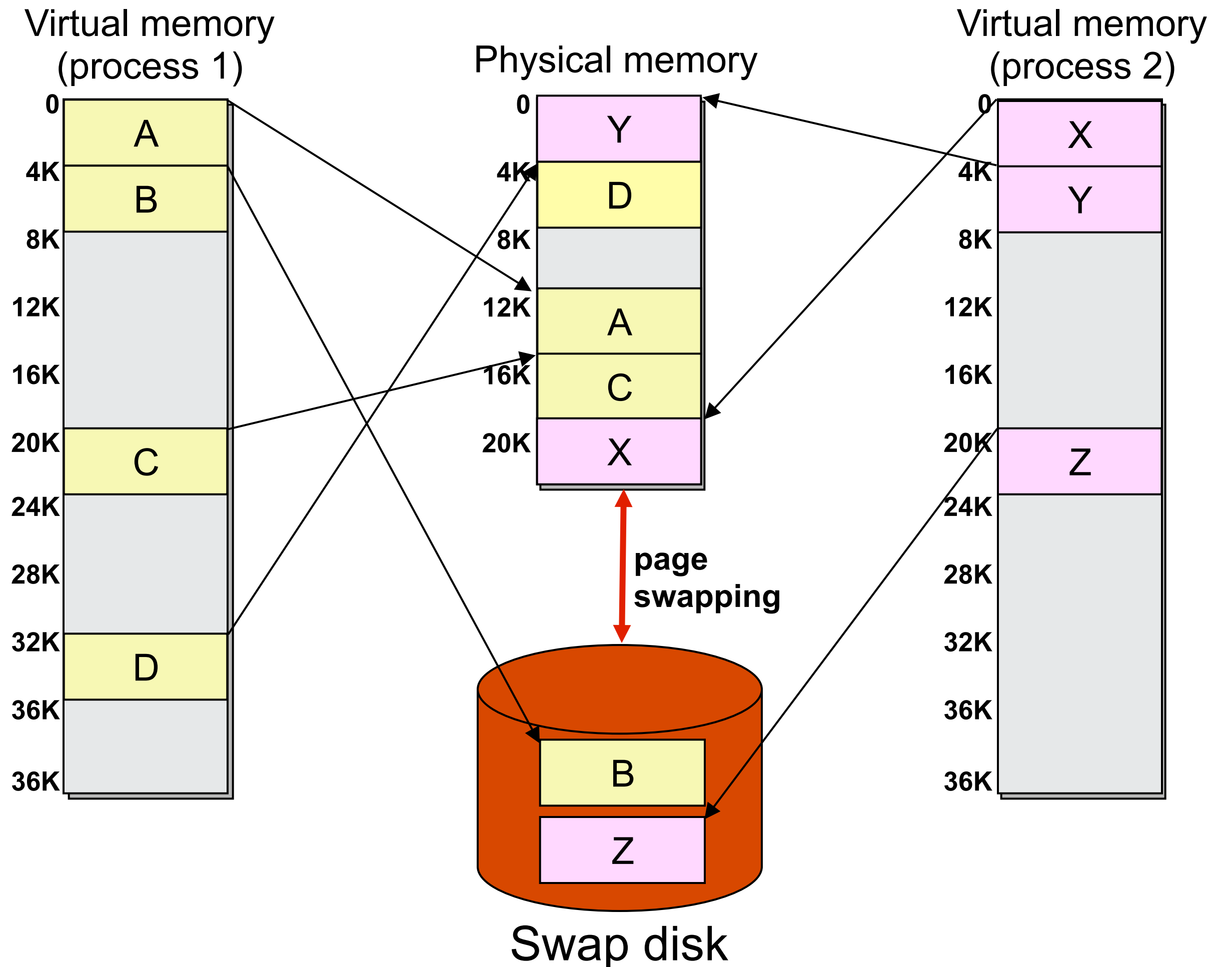
Virtual Memory

Basic idea:

- Each process has its own Virtual Address Space, divided into fixed-sized pages
- Virtual pages that are in use get mapped to pages of physical memory (called page frames).
  - Virtual memory: pages
  - Physical memory: frames
- Virtual pages not recently used may be stored on disk
- Extends the memory hierarchy out to the swap partition of a disk

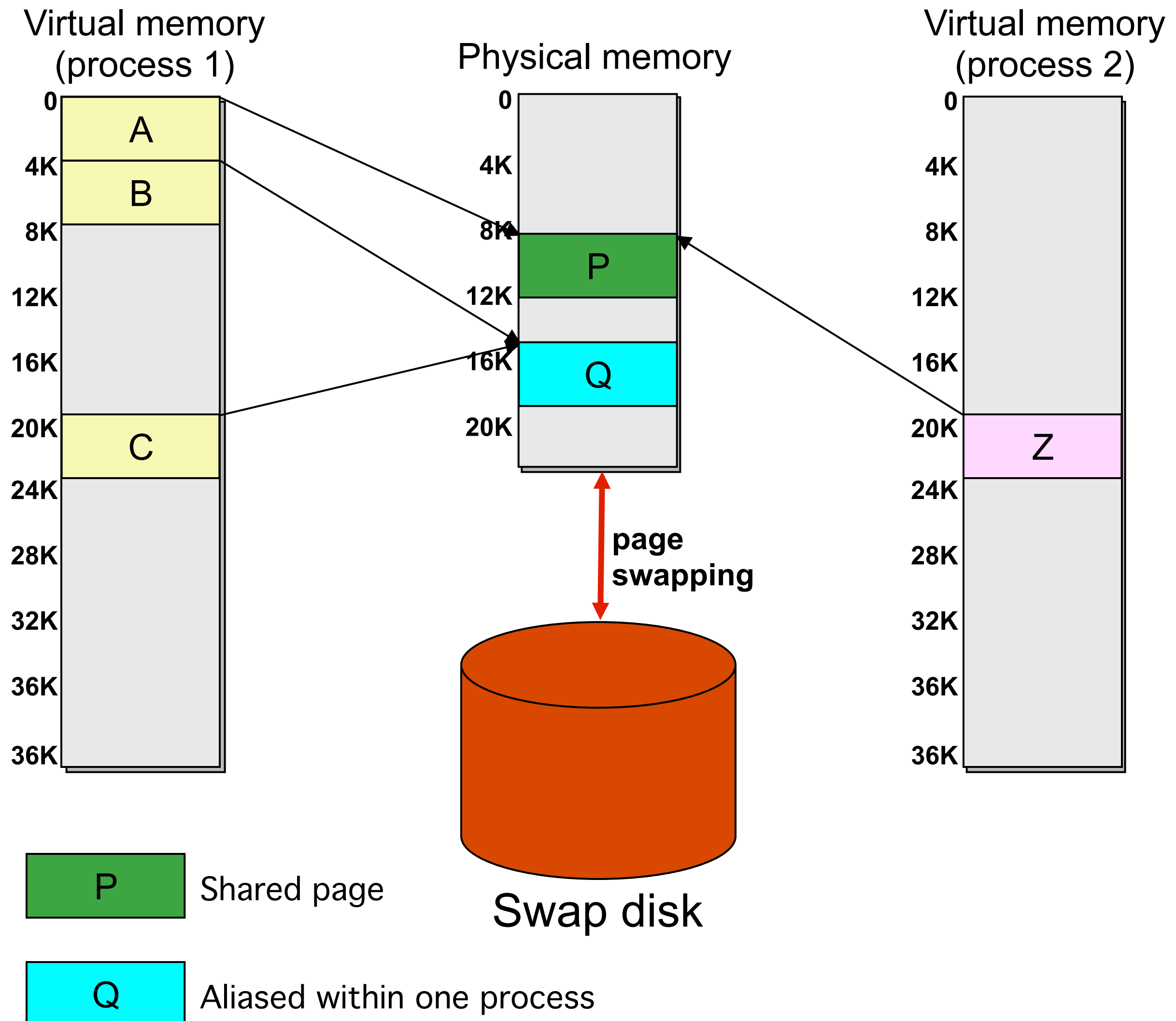
Virtual and Physical Memory

- Example 4K page size
- Process 1 has pages A, B, C and D
- Page B is held on disk
- Process 2 has pages X, Y, Z
- Page Z is held on disk
- Process 1 cannot access pages X, Y, Z
- Process 2 cannot access page A, B, C, D
- O/S can access any page (full privileges)



Sharing memory using Virtual Aliases

- Process 1 and Process 2 want to share a page of memory
- Process 1 maps virtual page A to physical page P
- Process 2 maps virtual page Z to physical page P
- Permissions can vary between the sharing processors.
- O/S can still access any page (full privileges)
- Note: Process 1 can also map the same physical page at multiple virtual addresses !!



Typical Virtual Memory[1] Parameters

parameter	L1 cache	memory
block/page	16-128 bytes	4KB-4MB
hit time	1-3 cycles	100-200 cycles
miss penalty	8-200 cycles	1M-10M cycles
(access time)	6-160 cycles	800K-8M cycles
(transfer time)	2-40 cycles	200K-2M cycles
miss rate	0.1-10%	0.00001-0.001%
size	4KB-64KB	128MB-128GB

H&P 5/e Fig. B.20

- Virtual Memory miss is called a *page fault*
- Page size is usually fixed, but some systems use variable size *segments*

[1] *Note: these parameters are due to a combination of physical memory organization and virtual memory implementation*

Virtual Memory Policies

- Block replacement: choosing a page frame to reuse
  - Minimize misses (page faults) → LRU policy
  - Minimize write backs to disk → give priority to non-modified pages
- Write strategy: policy adopted on a write
  - Write-through would mean writing the cache block back to disk whenever the page is updated in main memory → not practical
  - Write-back policy is always used (with Dirty or modified bit in page table)
    - Some systems use one Dirty bit per block in the page to minimize data writes back to disk
- Inclusivity:
  - Inclusive would mean having a copy of all used pages in disk → too expensive
  - Memory and Disk are non-inclusive in all systems

Virtual Memory Policies

- Block placement: location of page in memory
  - More freedom → lower miss rates, higher hit and miss penalties
  - Memory access time is already high and memory miss penalty (disk access time) is huge ⇒ must minimize miss rates
  - Full associativity → virtual page can be located in any page frame
    - No conflict misses
    - Important to reduce time to find a page in memory (hit time)
  - To place new pages in memory, OS maintains a list of free frames

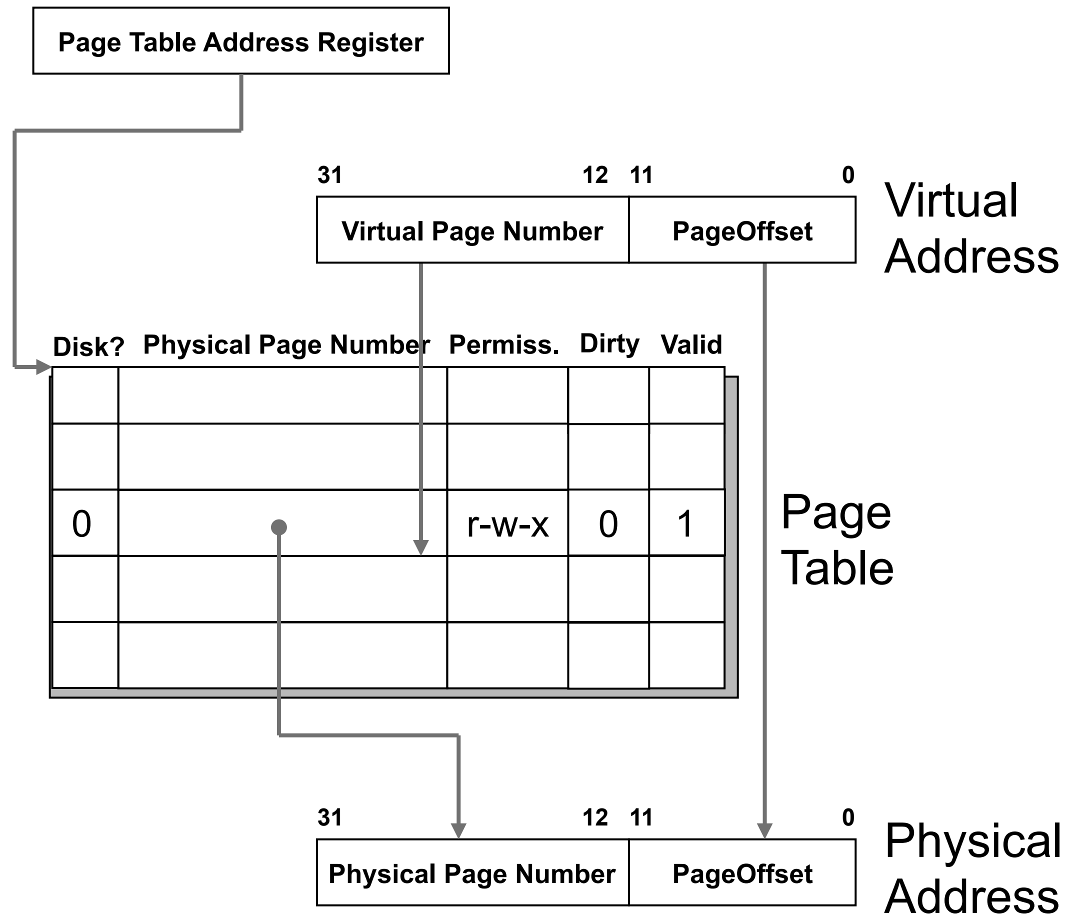
- Block placement may be constrained by use of translated virtual address bits when indexing the cache (see later)

Virtual Memory Policies

- Block identification: finding the correct page frame
  - Assigning tags to memory page frames and comparing tags is impractical
  - OS maintains a table that maps all virtual pages to physical page frames: Page Table
  - Table is updated with a new mapping every time a virtual page is allocated a page frame
  - Table is accessed on a memory request to translate virtual to physical address → inefficient!
    - Solution: cache translations (more later)
  - The number of entries in the table is the number of virtual pages → many!
    - e.g., 4KB pages → 2^{20}=1M entries for a 32b address space; 2^{52} entries for a 64b address space
    - Solution: hash virtual addresses to avoid maintaining a map from each virtual page (many) to physical frame (few). Resulting structure is inverted page table.
  - One Page Table per process

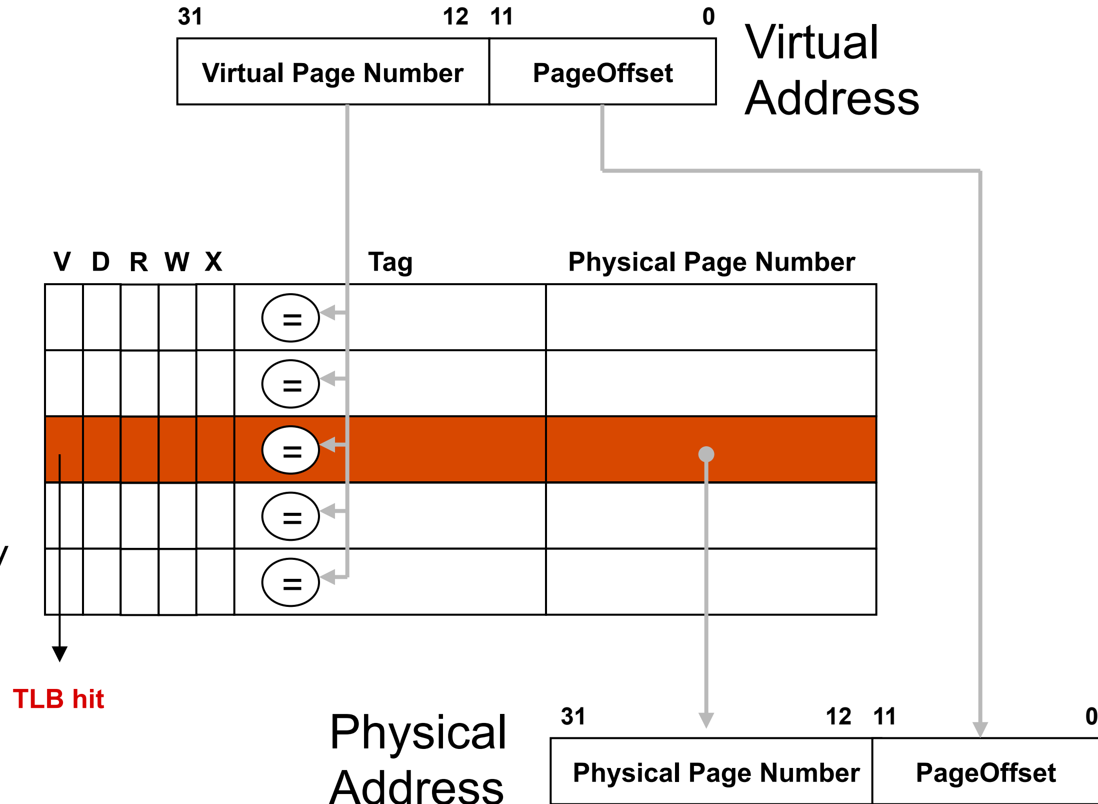
Page Tables and Address Translation

- Page table contains a translation for all virtual pages
- One page table for each process, and one for the system
- Each page has specific access permissions
  - Read, Write, Execute
- Bit indicates if page is on disk, in which case Physical Page Number indicates location within swap file
- “Dirty” bit indicates if there were any writes to the page
- Page table can be very large, so is often itself stored in virtual memory of the OS, and large parts may be swapped out
- CPU needs a cache of recently used Page Table Entries.



Translation Look-aside Buffers

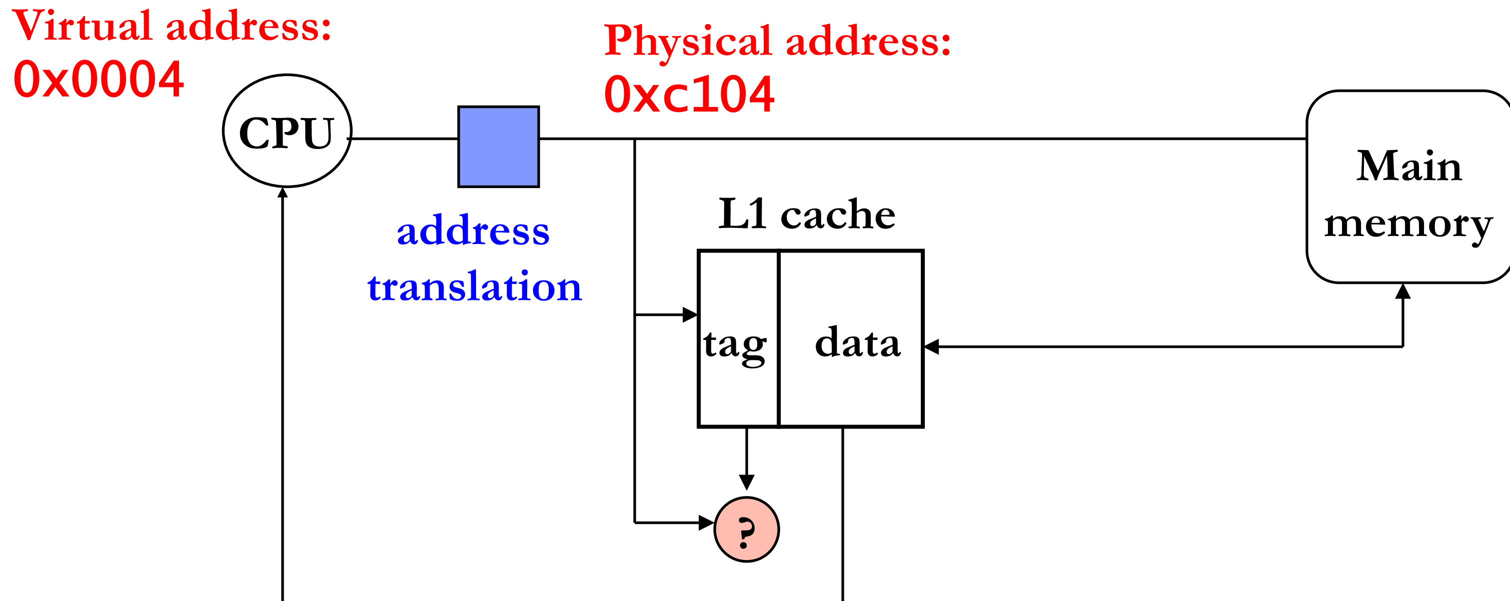
- Typically a small, fully-associative cache of Page Table Entries (PTE)
- Tag given by VPN for that PTE
- PPN taken from PTE
- Valid bit required
- D bit (dirty) indicates whether page has been modified
- R, W, X bits indicate Read, Write and Execute permission
- Permissions are checked on every memory access
- Physical address formed from PPN and Page Offset
- TLB Exceptions:
  - TLB miss (no matching entry)
  - Privilege violation
- Often separate TLBs for Instruction and Data references



How to address a cache in a virtual-memory system

Option 1: physically-addressed caches → perform address translation before cache access

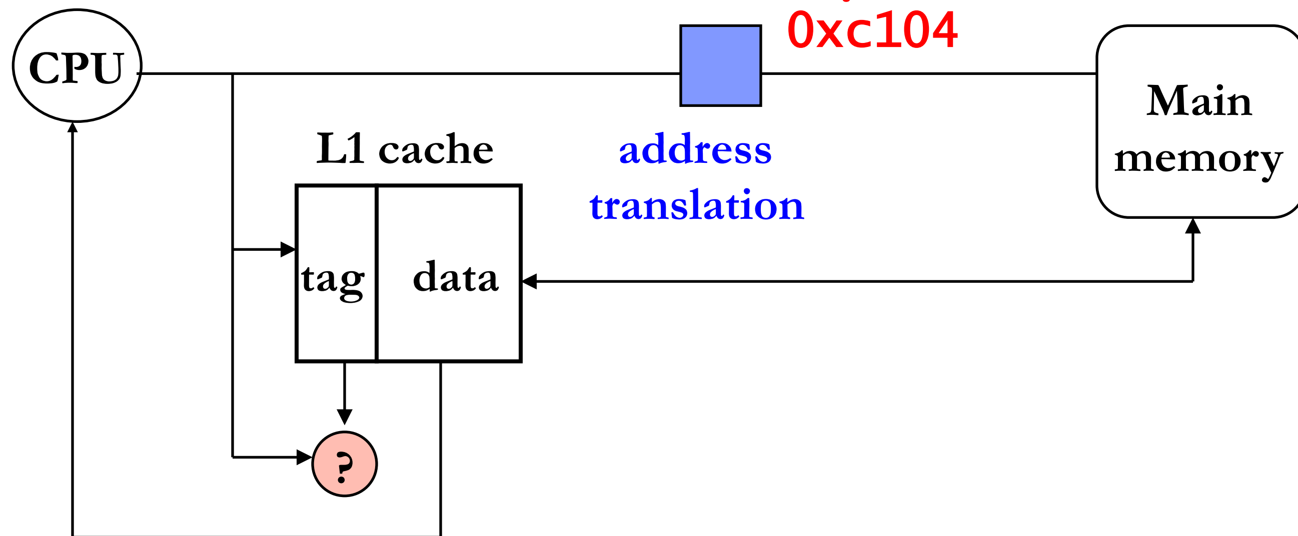
– Hit time is increased to accommodate translation ☹



How to address a cache in a virtual-memory system

Option 2: virtually-addressed caches → perform address translation after cache access if miss

- Hit time does not include translation ☺
- Aliases ☹

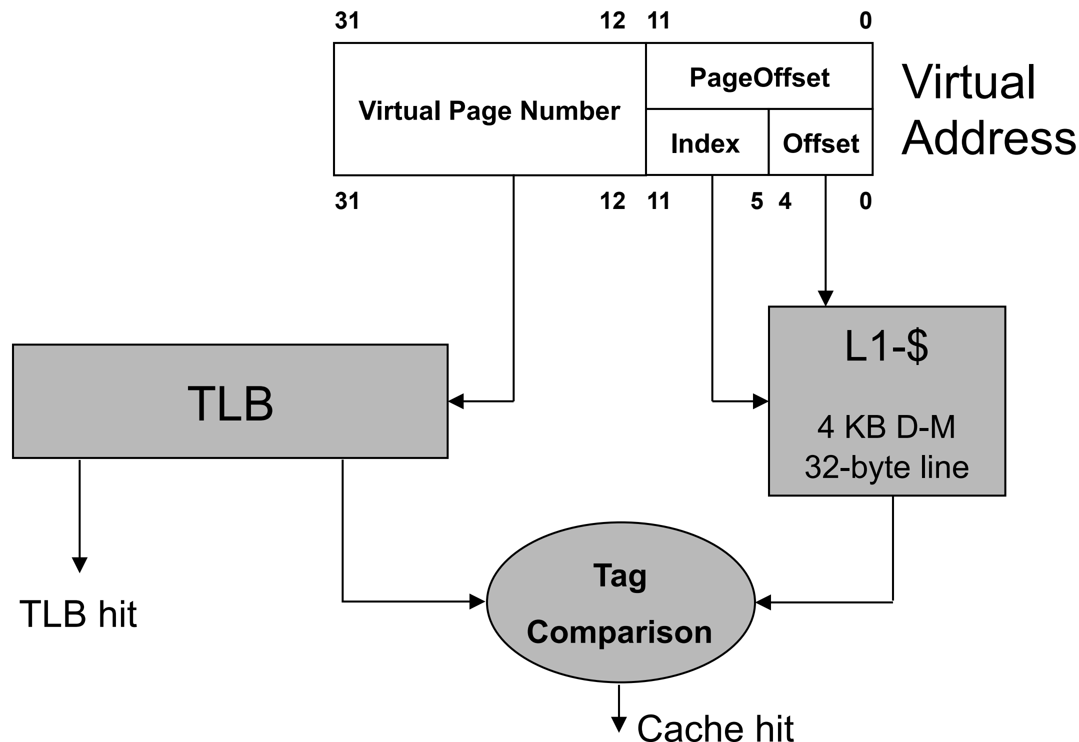


Problems with virtual aliases and caches

- Virtually tagged data cache problems:
  - A program may use different virtual addresses pointing to the same physical address
    - Two copies could exist in the same data cache
    - Writing to copy 1 would not be reflected in copy 2
    - Reading copy 2 would get stale data
  - Does not provide a coherent view of memory
  - Must be able to distinguish across different processes
    - Flush cache on context switch or add process ID to each tag
- Solution:
  - Use Physical address tags
  - Aliases have same physical address, therefore same tag
  - Only one copy exists in each cache
- Implications for CPU-cache interactions:
  - Must translate addresses before cache tag check
  - May still be able to index cache using non-translated low-order address bits under certain circumstances.

VI-PT: translating in parallel with L1-$ access

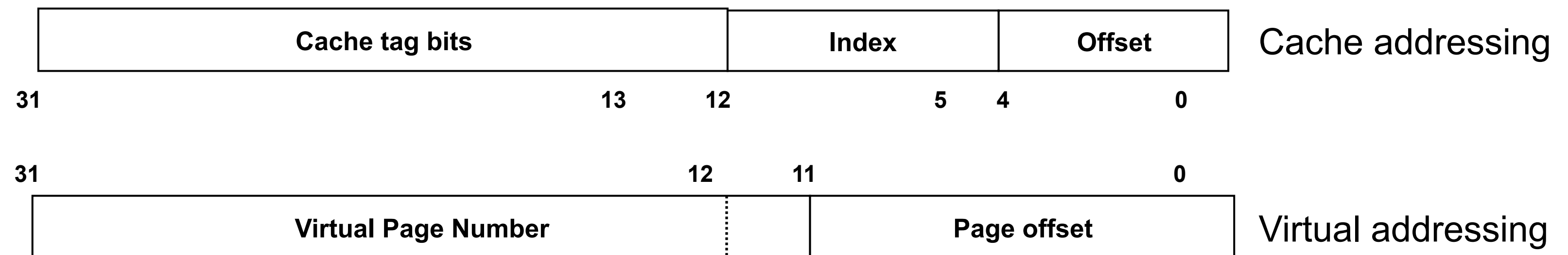
- Access TLB and L1-$ in parallel
- Requires that L1-$ index be obtained from the non-translated bits of the virtual address.
- This constraint in the number of bits available for the index limits the size of the cache!



IMPORTANT:
If the cache Index extends beyond bit 11, into the translated part of the address, then translation must take place before the cache can be indexed

Coping with large VI-PT caches

- Multi-way caches: multiple blocks in the same set
- Larger page size: more bits available for the index
- Rely on page allocator in the O/S to allocate pages such that the translation of index bits would always be an identity relation
  - Hence, if virtual address A translates to physical address P, then Page Allocator must guarantee that: V[11] == P[11]
  - This approach is referred to as "page coloring".
- Check other potential sets for aliases on a miss
  - E.g., AMD Athlon: 64KB 2-way cache w/ 4KB pages → on a miss, 7 add'l cycles to check for aliases in other sets



Summary: how to address a cache

- VI-VT : Virtually indexed, virtually tagged
  - L1-$ indexed with virtual address, before translation, tag contains virtual address
  - Con: Cannot distinguish virtual aliases or synonyms in cache
  - Pro: Only perform TLB lookup on L1-$ miss

- PI-PT : Physically indexed, physically tagged
  - Translation first; then cache access
  - Con: Translation occurs in sequence with L1-$ access – high latency

- VI-PT : Virtually indexed, physically tagged
  - L1-$ indexed with virtual address, or often just the un-translated bits
  - Translation must take place before tag can be checked
  - Con: Translation must take place on every L1-$ access
  - Pro: No aliases in the cache; works with cache-coherent shared memory

- PI-VT : Physically indexed, virtually tagged
  - Not interesting