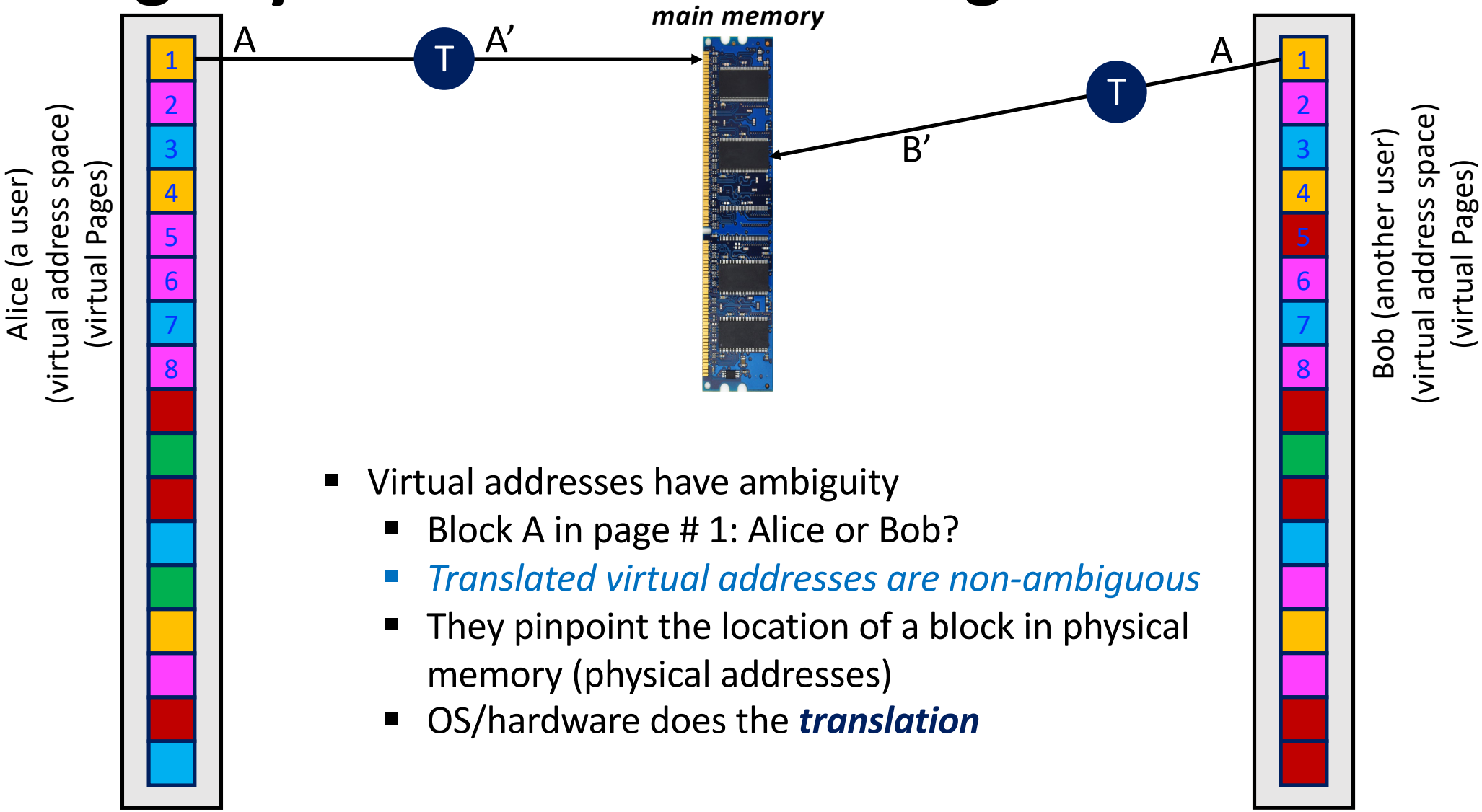
COMP3710 (Class # 5176)
Special Topics in Computer Science
Computer Microarchitecture

Convener: Shoaib Akram
shoaib.akram@anu.edu.au

Australian
National
University

Ambiguity in Virtual Addressing



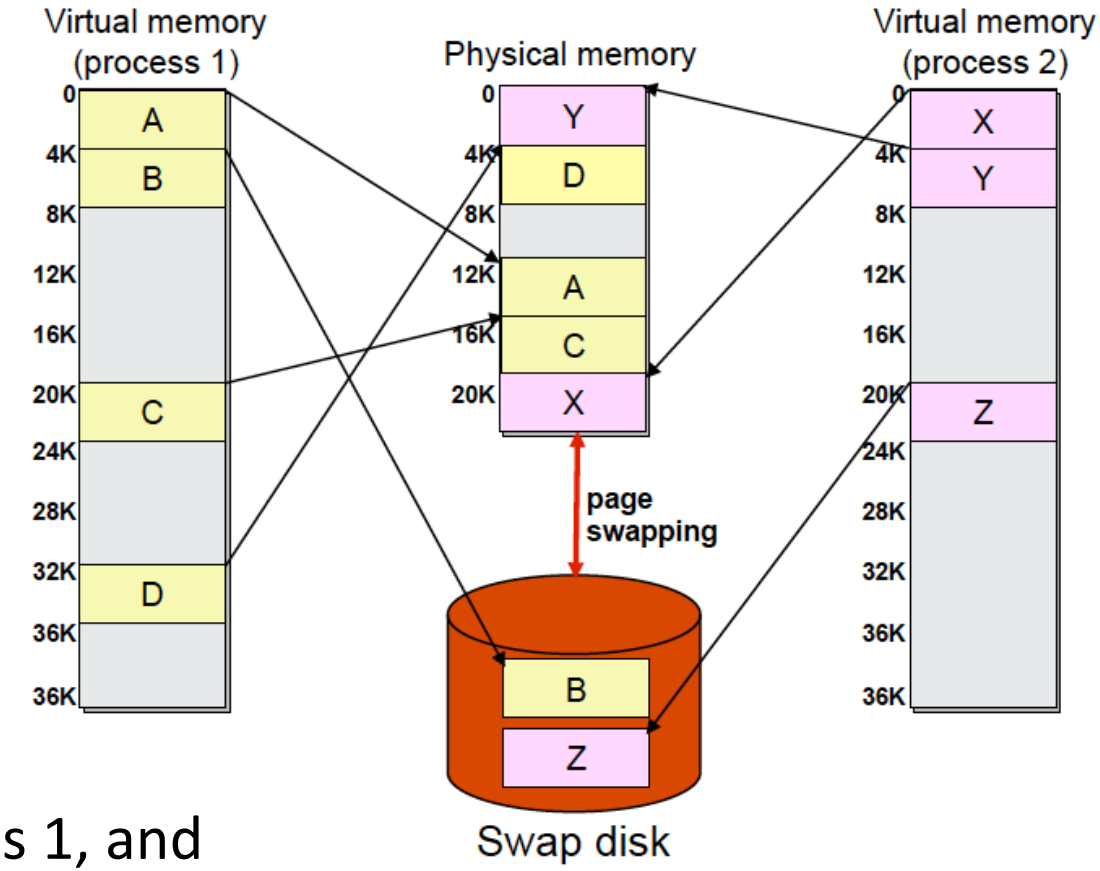
- Virtual addresses have ambiguity
  - Block A in page # 1: Alice or Bob?
  - *Translated virtual addresses are non-ambiguous*
  - They pinpoint the location of a block in physical memory (physical addresses)
  - OS/hardware does the ***translation***

Homonym and Synonym

- **Homonym**
  - Two similar virtual addresses from different processes with different physical addresses
  - Homonyms in English: same spelling, different meaning
- **Synonym**
  - Two different virtual addresses (same process) with the same physical address
  - Also called virtual aliases

Example of Homonym

- 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
- OS can access any page (full privileges)

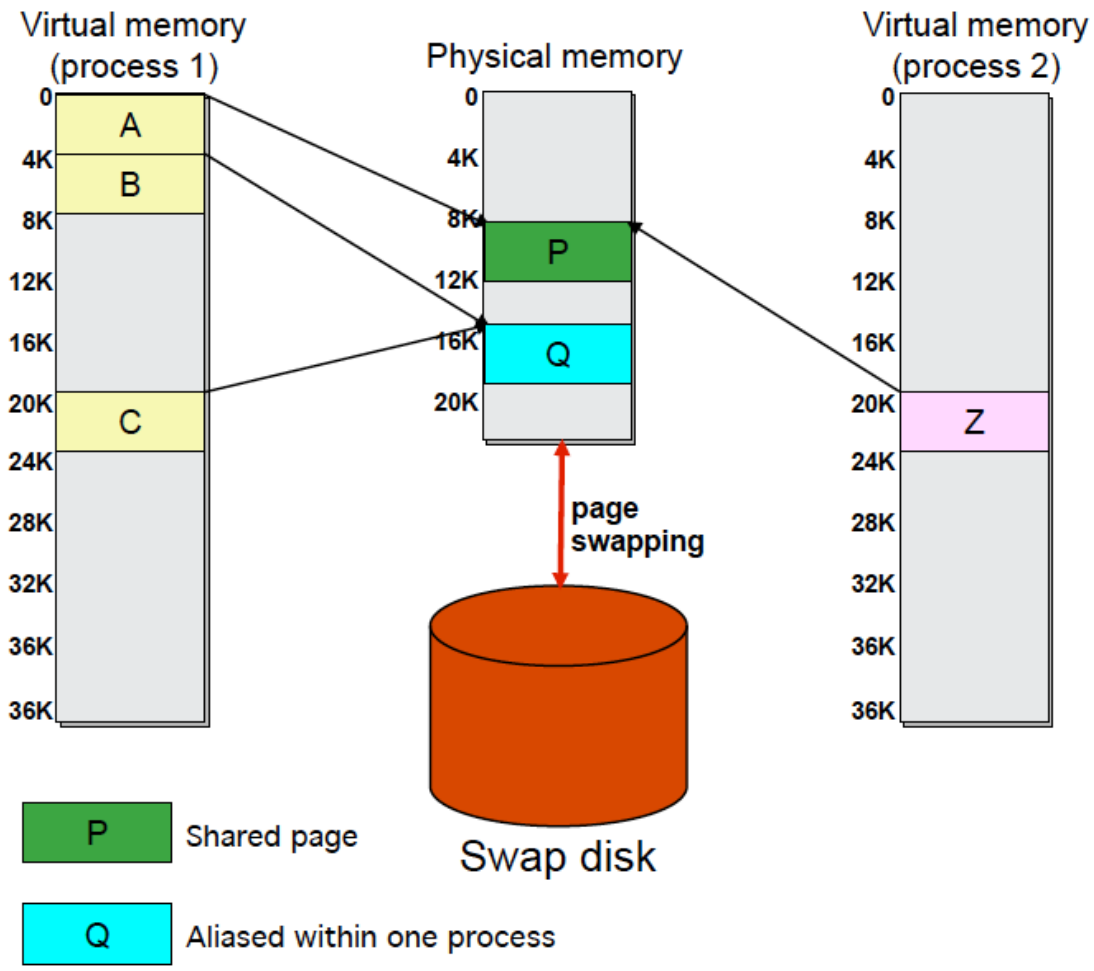


Key idea: Virtual page 0 of process 1, and virtual page 0 of process B are homonyms

Example of Synonym

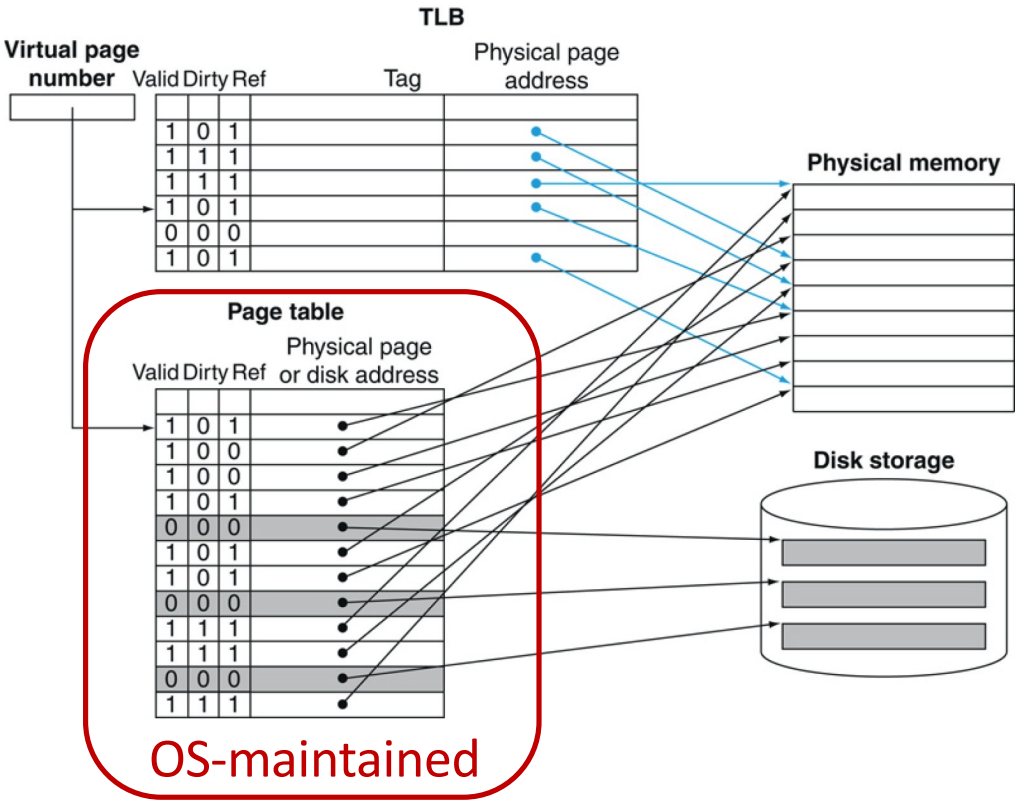
- Process 1 and Process 2 want to share a page of memory
- Process 1 maps virtual page A to physical page P
- Process 2 maps virtual page Z to physical page P
- Permissions can vary between the sharing processors.
- **Note:** Process 1 can also map the same physical page at multiple virtual addresses

Key idea: Virtual pages B and C are **synonyms**



TLB Issues

- TLB is a dedicated cache for recently accessed page table entries
- Page tables (per process) are stored in physical memory
  - Starting address of the table for the currently executing process is in the page table register (PTR)
  - Memory management unit (MMU) uses the PTR to walk the page table (TLB miss)
- **TLB is exposed to the OS**
  - Many scenarios require the OS to flush an entry (or entries) from the TLB
    - Page replacement
    - Page remapping (in multicores, this results in a *TLB shootdown*)

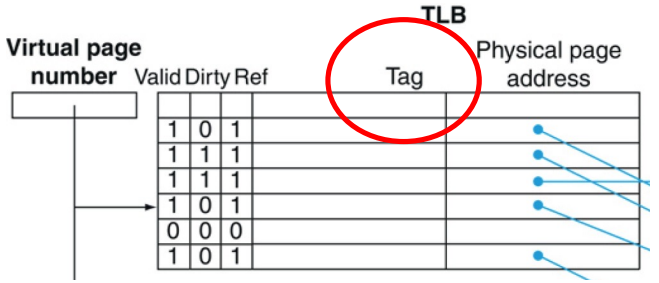


Question

- When we move a page to the swap area (e.g., on replacement) on disk, there is no longer a virtual→physical mapping in the page table. *What happens to the cached contents of that page in the processor caches?*
  - The OS flushes the contents from the cache when it decides to migrate a page to disk
  - The attempt to access any data from that virtual page generates a page fault

Tag Bits in the TLB

- How many tag bits in each TLB entry?
  - 32-bit system, 4 KB page size, 256-entry fully-associative TLB
    - Need to search every entry in the TLB for a virtual page
    - 20 bits for the tag
  - 32-bit system, 4 KB page size, 256-entry direct-mapped TLB
    - 8 bits for indexing the TLB
    - 12 bits for the tag

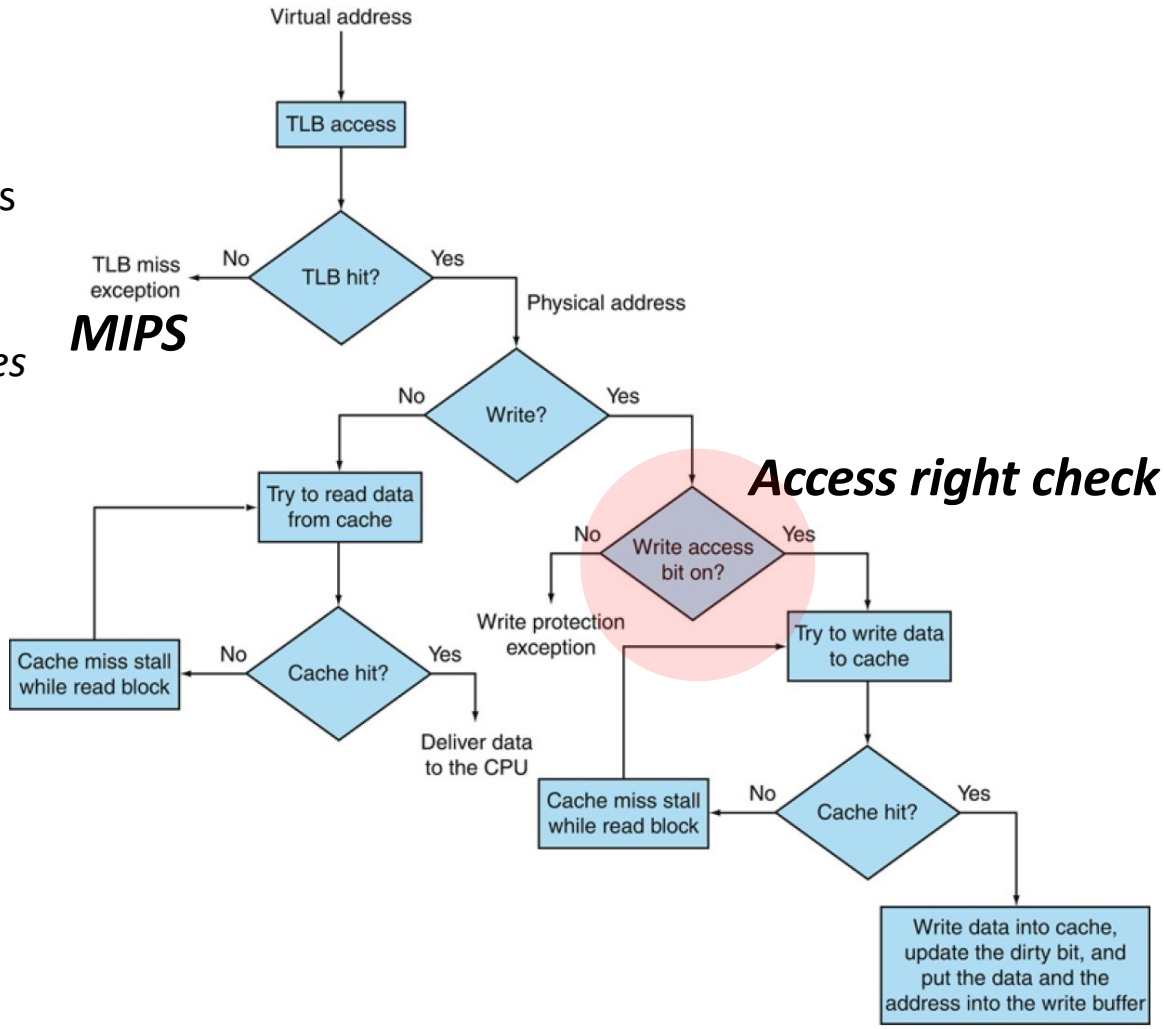


Note: If the virtual page and the physical frame are both the same size (4 KB), then the low-order 12 bits of the physical address remain unchanged after the translation from the virtual address

Enabling Protection

Can add access rights to the page table entries to protect certain pages from being written

- *Malicious user can manipulate page table entries*
- *OS sets the bit on/off on a user to kernel switch*



Cache Access with Virtual Mem

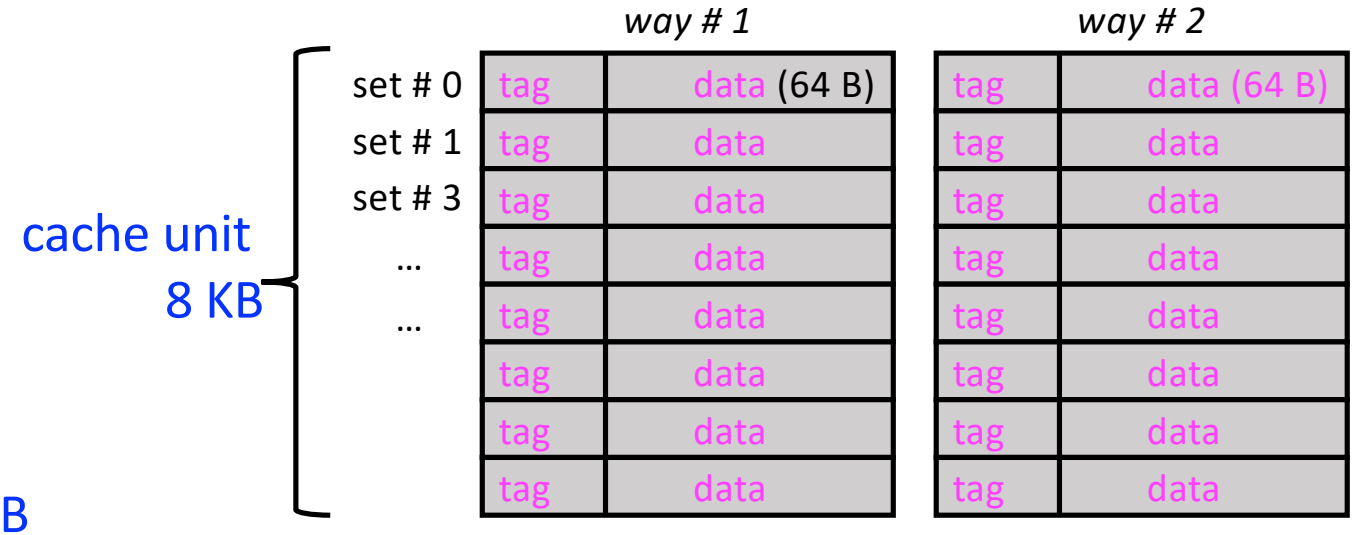
- Recall the two major steps of accessing a *four-way* set-associative cache
  - Finding the set # (n low order bits)
  - Comparing each of the *four* tags to the tag bits of address (high order)

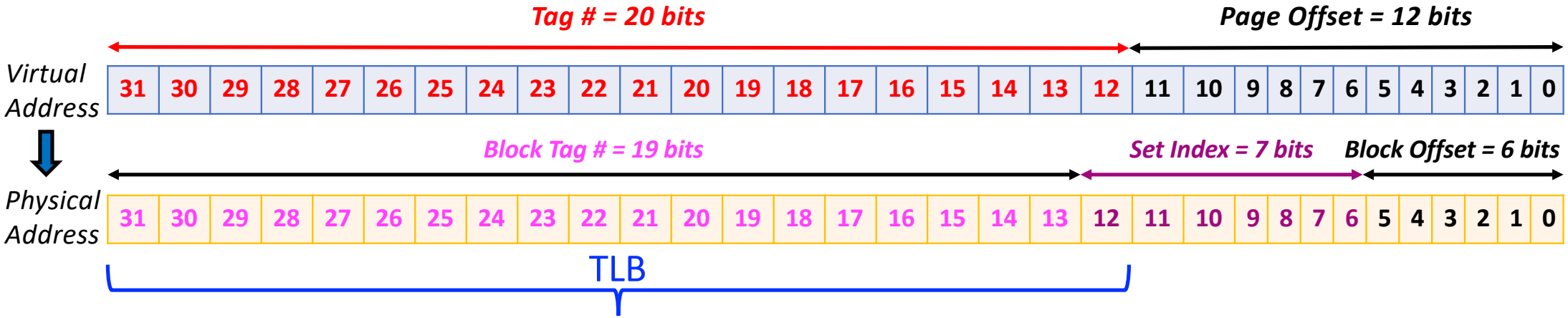
- **Question:** *To use virtual or physical address to index the cache?*
  - **P**hysically **I**ndexed and **P**hysically **T**agged cache (**PIPT**)
    - TLB access is on the critical path
  - **V**irtually **I**ndexed and **V**irtually **T**agged cache (**VIVT**)
    - Two virtual pages (different processes, Alice & Bob) mapped to the same physical address
    - Two virtual pages (one process, Alice) mapped to the same physical page

PIPT (2-Way Set Assoc)

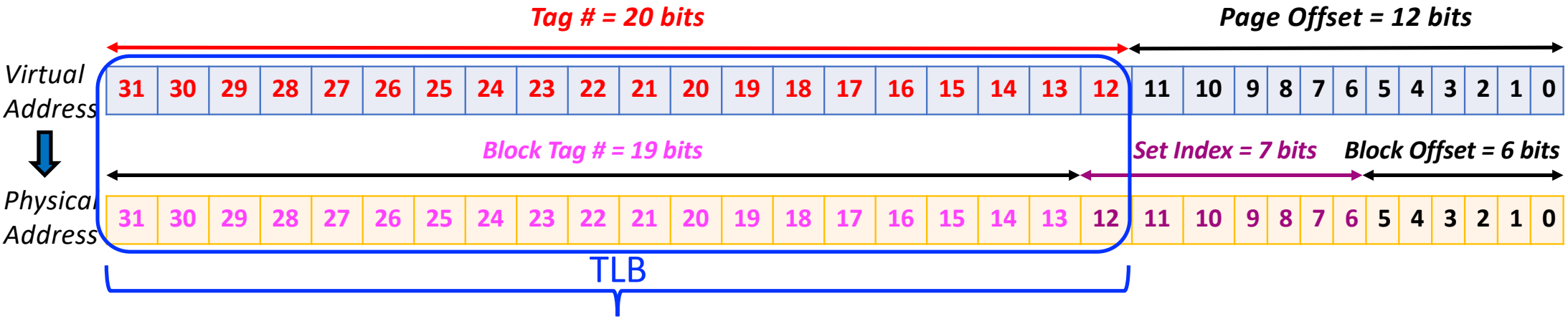
System specs
- 32 bit addresses
- Page size = 4 KB
- 4 GB physcial memory

Hypothetical cache
- Block size = 64 bytes
- # sets = 128
- # sets X block size = 8 KB
- Two pages fit in one unit/way





PIPT – Removing Ambiguity

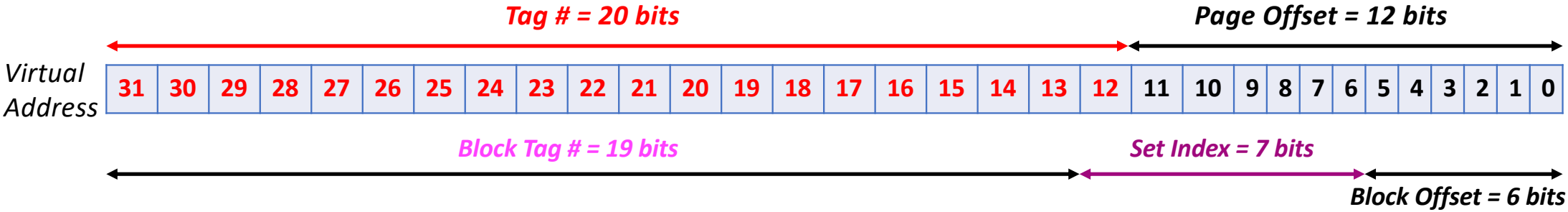


- Size of virtual and physical page is 4 KB
  - The low-order 12 bits do not change after translation
  - The page offset is used to index into the page to access the word/byte
- We must remove the ambiguity in the 13th bit and wait for the translation
  - Two processes may have the same 13th bit in the virtual addresses but not in their physical addresses

PIPT

- **Advantage (Will become clear once we see other indexing schemes)**
  - No constraint on the size of the cache unit
    - (# sets X block size) *can exceed the 4 KB page size*
  - Multiple pages can fit inside a single unit
    - An easy way to build large caches without increasing associativity
- **Disadvantage**
  - TLB access and cache access are serialized
  - Must get translation from TLB to find the correct set in the cache

VIVT



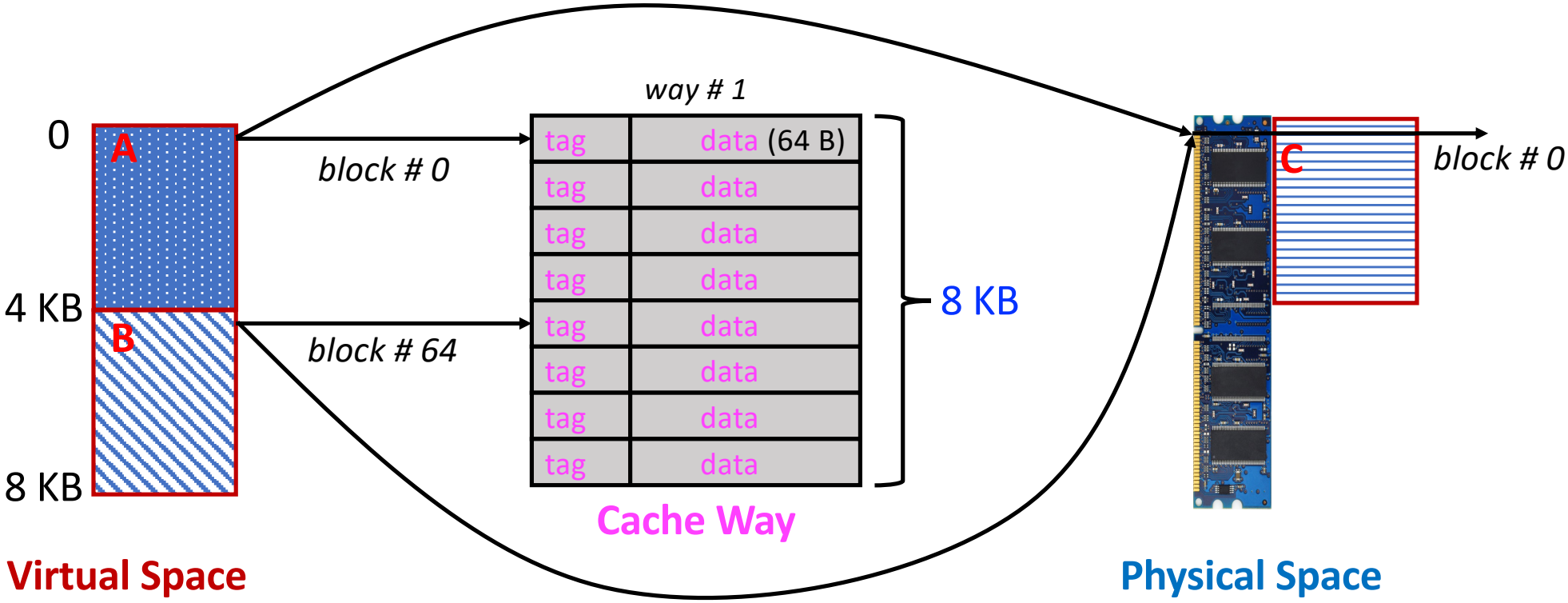
Two problems

- Virtual aliasing due to multiple users
  - *Cache flushing on a context switch solves this problem*
- Virtual aliasing due to multiple virtual pages mapped to a single physical page
  - Two copies of the same physical block in the cache (correctness problem)
  - How can we solve this problem?

Problem

Synonyms (aliases)

- When the OS/program uses two virtual addresses for the same physical address
  - Allowed behavior with some practical uses
- Duplicate addresses are aliases of each other
- If one is modified, then the other has wrong data (impossible with a physical cache)

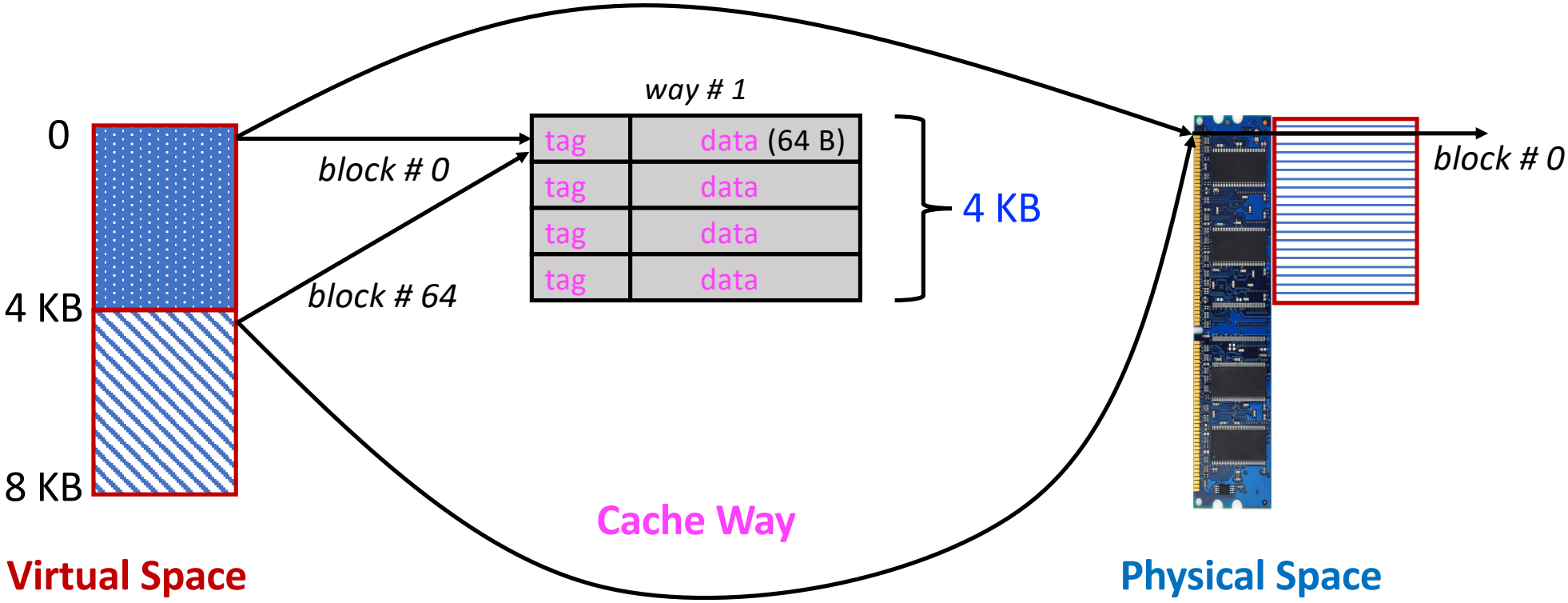


Solution

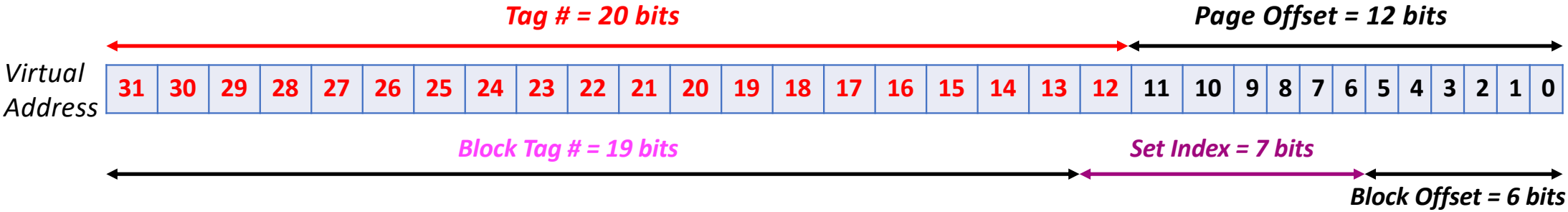
Limit the cache unit size (# sets X block size) to be 4 KB

Dealing with Synonyms

Dealing with synonyms comes with a limitation on cache design



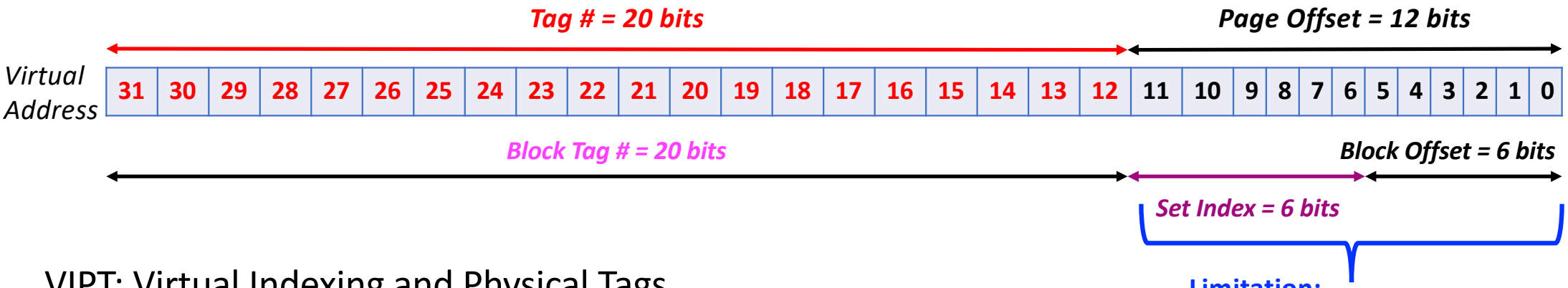
VIVT



Two problems

- Virtual aliasing due to multiple users
  - *Cache flushing on a context switch solves this problem*
- Virtual aliasing due to multiple virtual pages mapped to a single physical page
  - Two copies of the same physical block in the cache (correctness problem)
  - How can we solve this problem?

VIPT

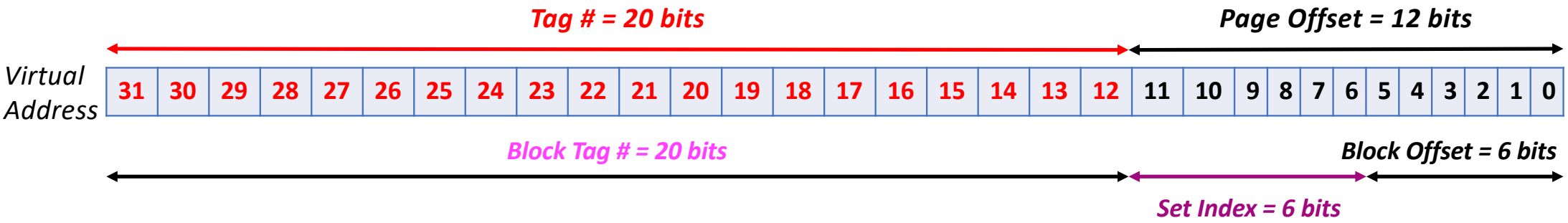


VIPT: Virtual Indexing and Physical Tags

- Use virtual bits of the page offset to index the cache
  - Limitation: Cache unit size cannot exceed 4 KB
- Use physical tags to eliminate ambiguity due to virtual aliases (multiple users)
  - *Better alternative to flushing on context switches*

Limitation:
sets * line-size cannot be greater than page size

VIPT



Operation of a cache with VIPT

- Use the page offset to index the cache (**step # 1**)
- In parallel, get the translation from the TLB (**step # 1**)

Overlapped in time

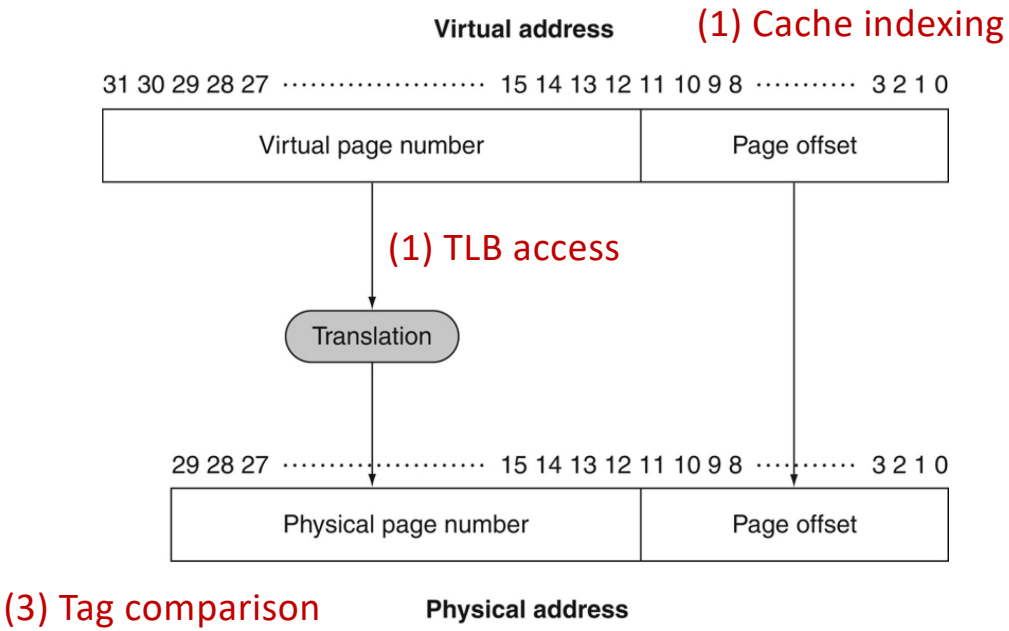
- Then, compare the physical tag to the translated portion of the address (**step # 2**)

VIPT

Operation of a cache with VIPT

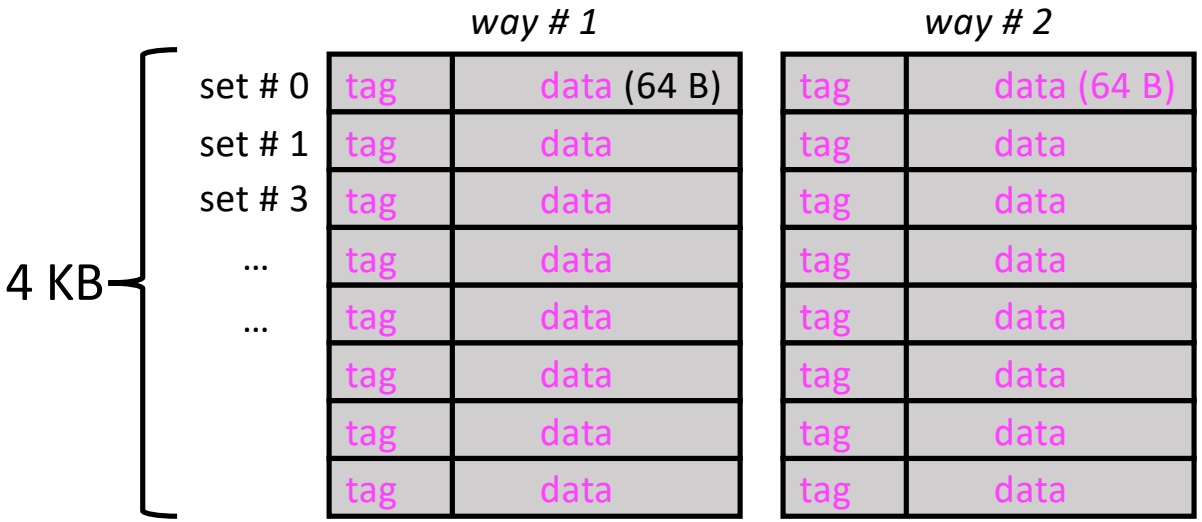
- Use the page offset to index the cache (**step # 1**)
- In parallel, get the translation from the TLB (**step # 1**)
- Then, compare the physical tag to the translated portion of the address (**step # 2**)

Overlapped in time



Question

- The upper-most cache uses VIPT for performance reasons. *With the size of the cache unit limited to 4 KB, what option does a microarchitect has to increase the cache capacity?*
  - With OS vendors not taking measures to eliminate aliasing in software, the L1-D cache size is stuck around 32 KB to 64 KB
  - Increasing associativity to increase cache capacity → High L1-D latency



Popular Addressing Strategy

Level 1 is VIPT
Level 2-3 are PIPT

VIPT Example

Specs

- Address spaces: 64-bit virtual, 41-bit physical
- 8 KB page size
- 8 KB direct-mapped L1 cache
- 64 Byte blocks (lines)
- 256 entry direct-mapped TLB

Explaining A-K (# bits)

8 KB page

- 13 bit page offset (**A**)
- 51 bits virtual page # (**B**)

256-entry direct-mapped TLB

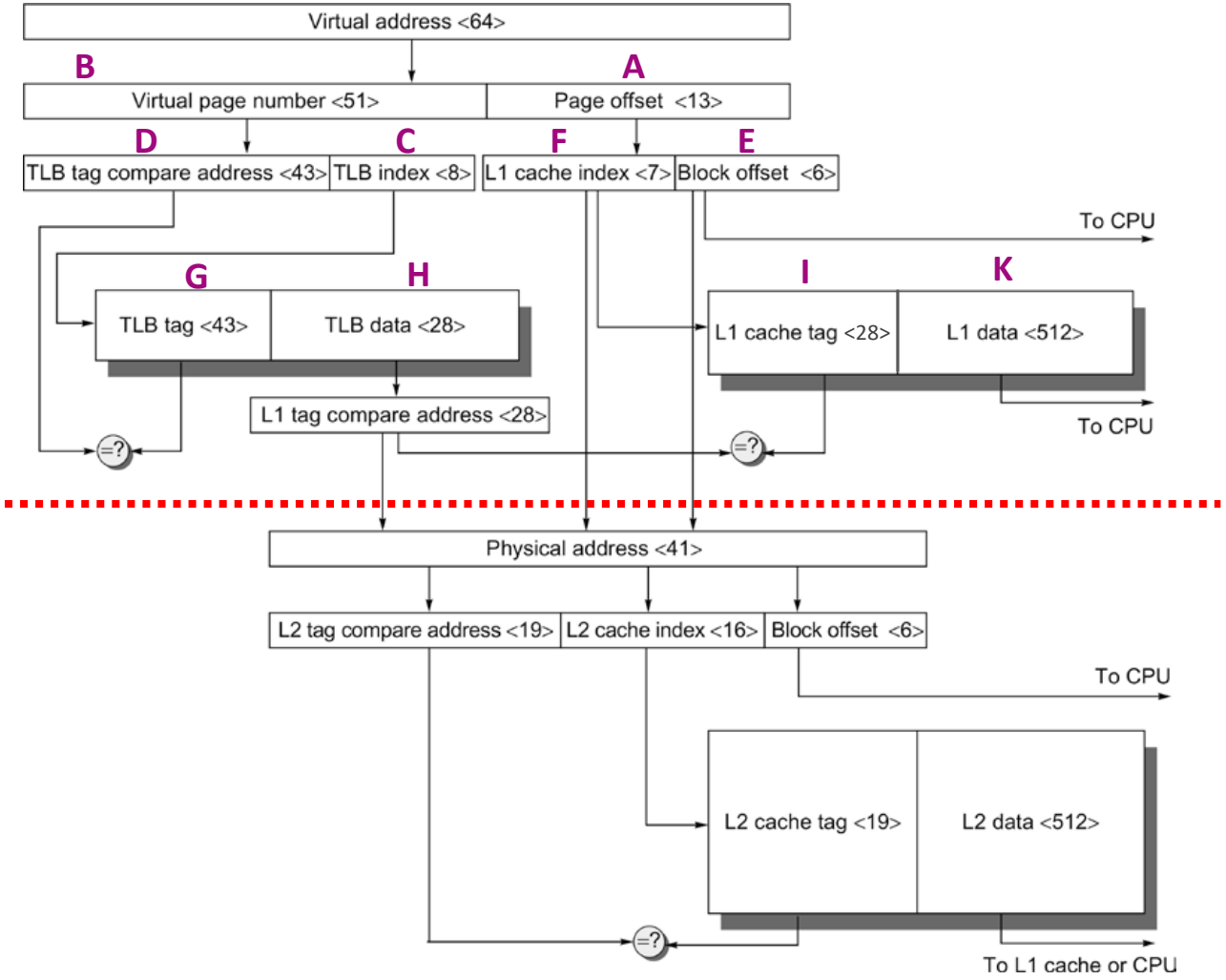
- 8 bits to index the TLB (**C**)
- TLB tag is 51 *minus* 8 = 43 (**D**)

Tagging the physcial pages

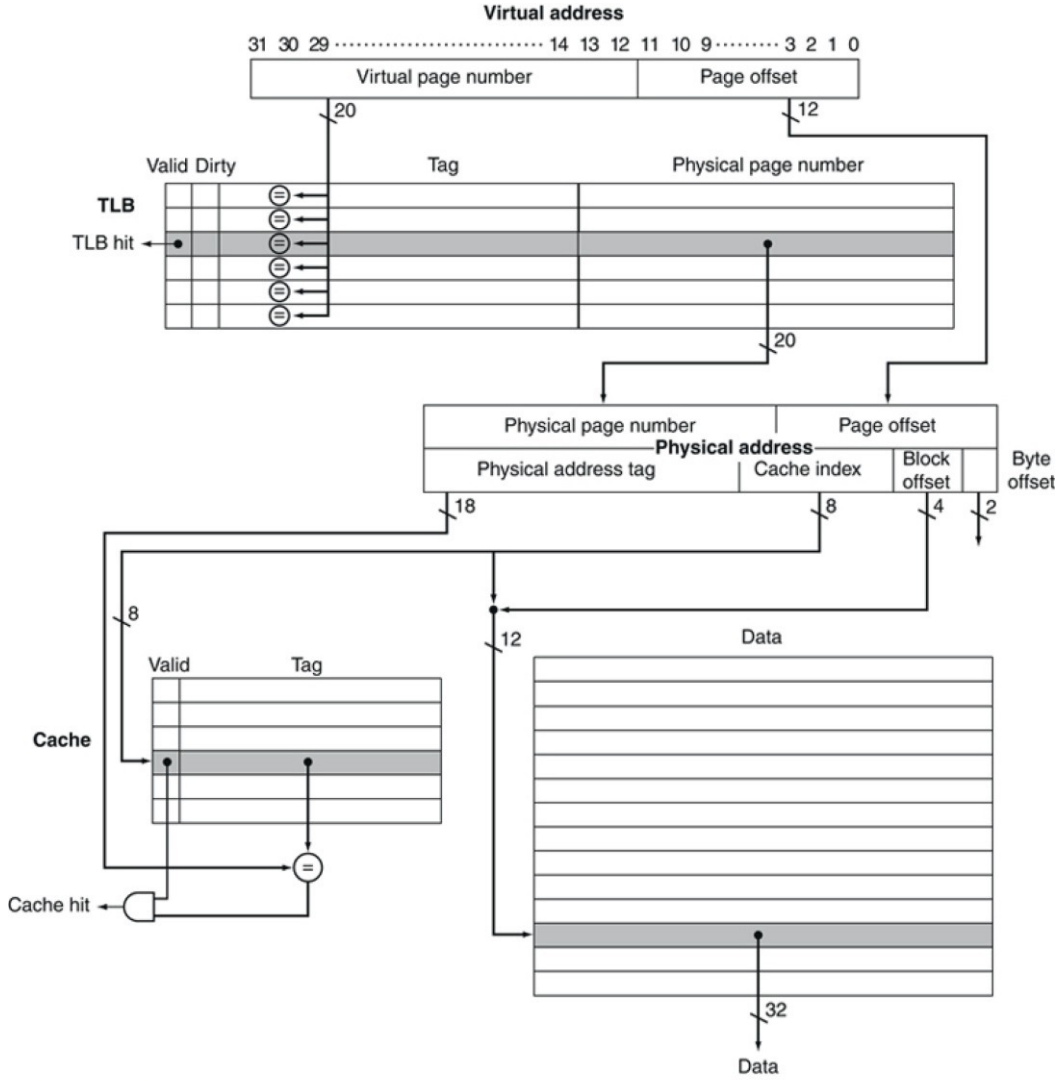
- 41 – 13 = 28 (**H,I**)

L1 Cache

- 64-byte line size (6 bits) **E**
- 128 sets (blocks) → 7 bits **F**



Example: PIPT Addressed Cache



Complete the translation before indexing the cache