上海科技大学
ShanghaiTech University

CS211
Advanced Computer Architecture

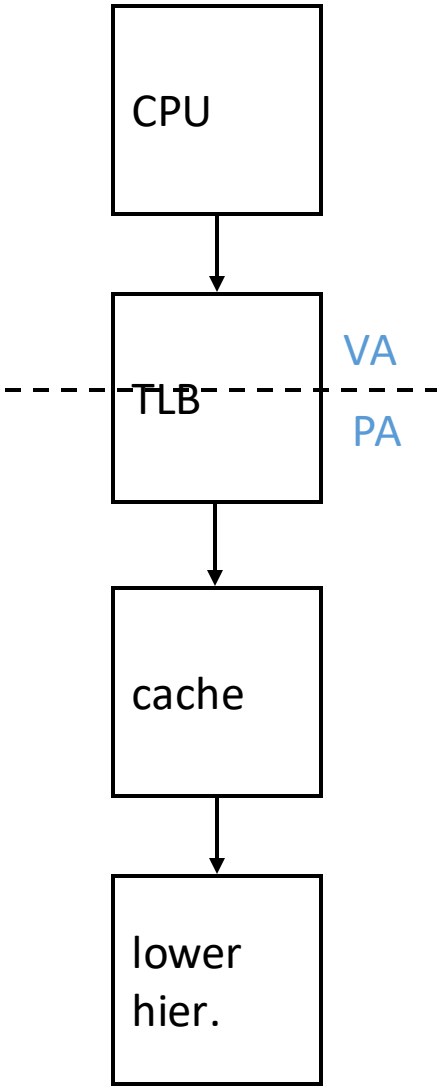
L08 Memory III

Chundong Wang

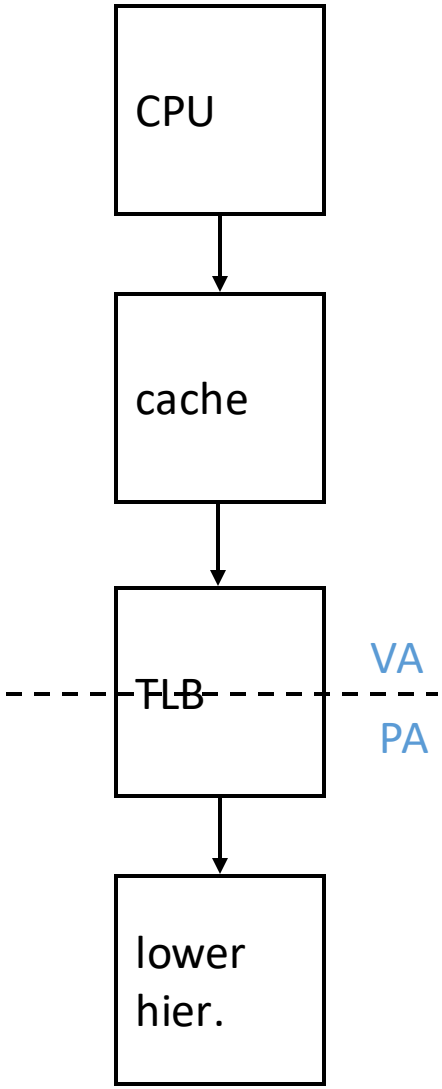
October 17th, 2025

Virtual Memory and Cache Interaction

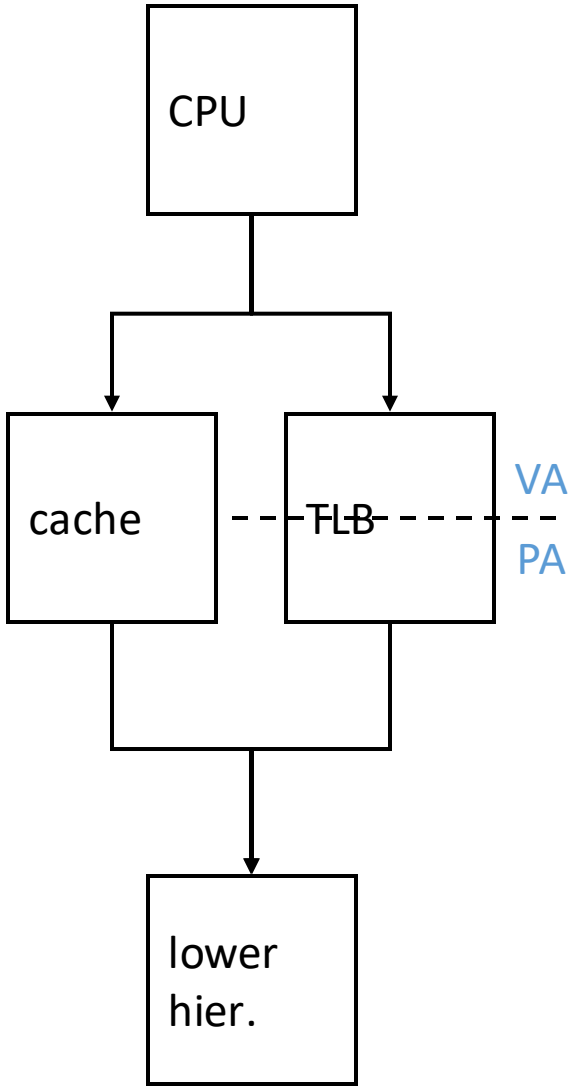
Cache-VM Interaction



physical cache

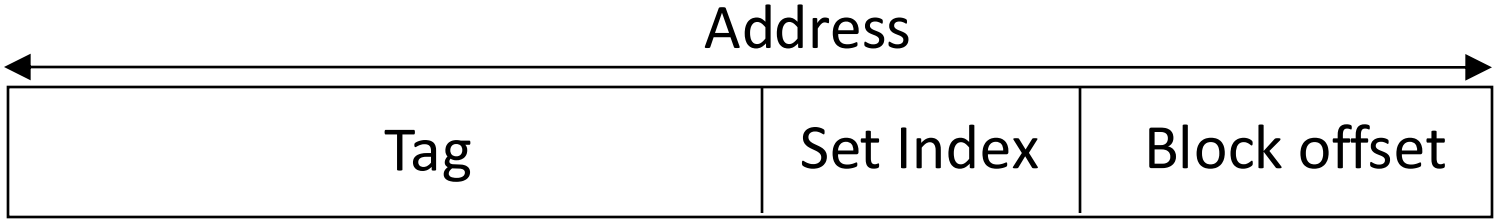


virtual (L1) cache



virtual-physical cache

Address Translation and Caching



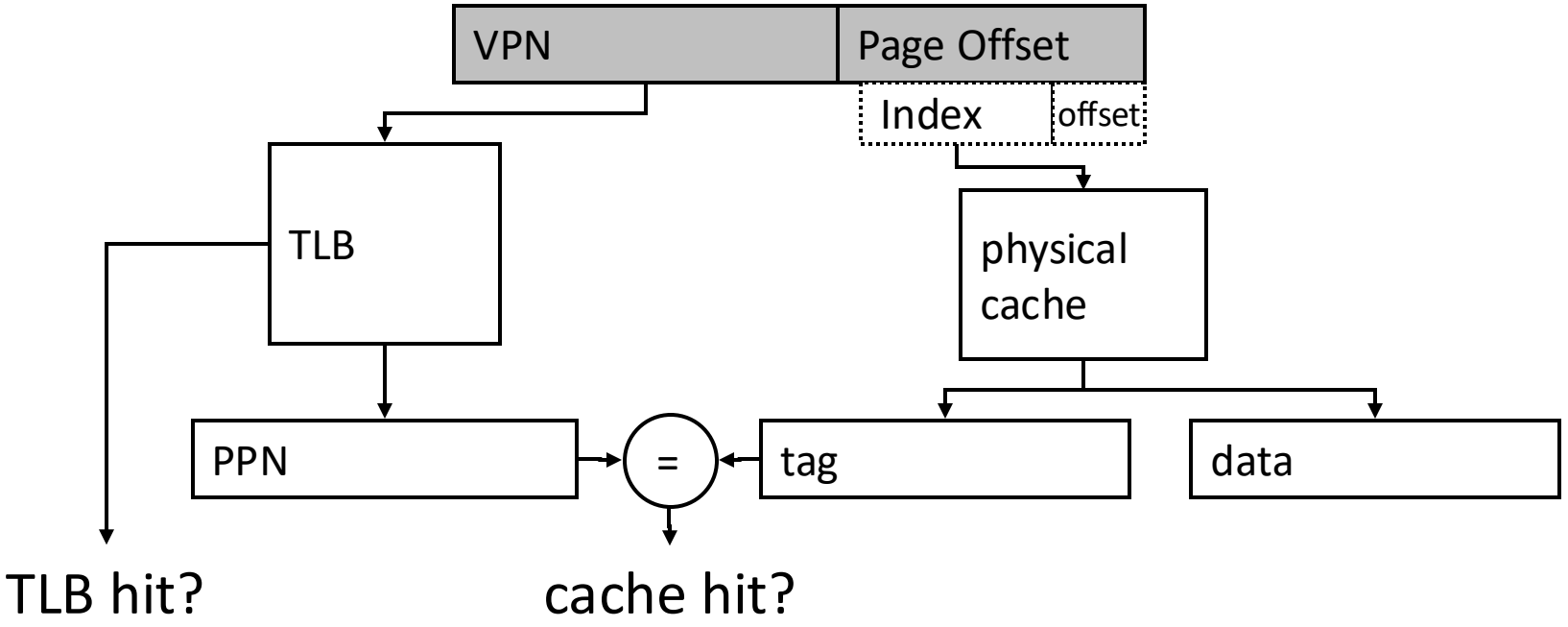
- When do we do the address translation?
  - Before or after accessing the L1 cache?
- In other words, is the cache virtually addressed or physically addressed?
  - Virtual versus physical cache
- What are the issues with a virtually addressed cache?
- Synonym problem:
  - Two different virtual addresses can map to the same physical address → same physical address can be present in multiple locations in the cache → can lead to inconsistency in data

Homonyms and Synonyms

- Homonym: Same VA can map to two different PAs
  - Why?
    - VA is in different processes
- Synonym: Different VAs can map to the same PA
  - Why?
    - Different pages can share the same physical frame within or across processes
    - Reasons: shared libraries, shared data, copy-on-write pages within the same process, …
- Do homonyms and synonyms create problems when we have a cache?
  - Is the cache virtually or physically addressed?

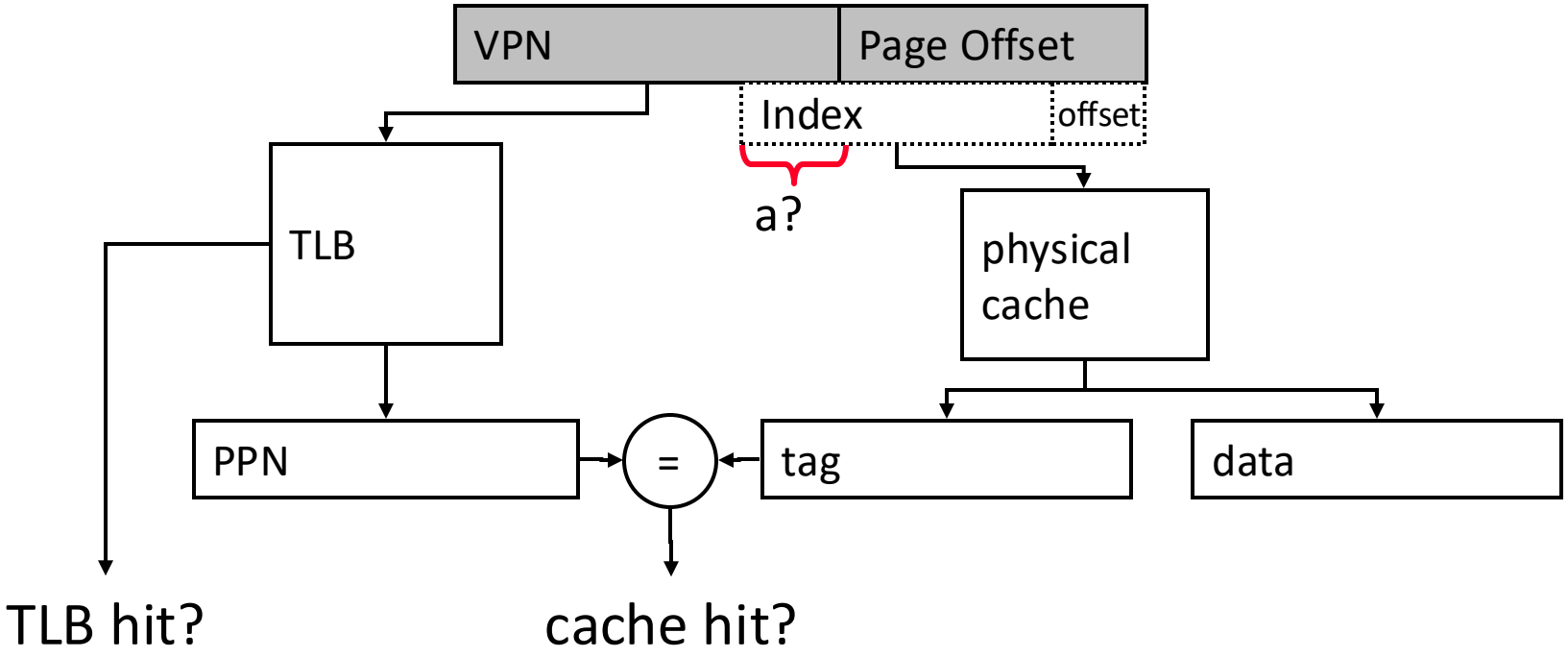
Virtually-Indexed Physically-Tagged

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end



Virtually-Indexed Physically-Tagged

- If C>(page_size × associativity), the cache index bits include VPN ⇒ Synonyms can cause problems
  - Different VAs mapped to the same physical address
  - The same physical address can exist in two locations
- Solutions?



Some Solutions to the Synonym Problem

- Limit cache size to (page size × associativity)
  - get index from page offset
- On a write to a block, search all possible indices that can contain the same physical block, and update/invalidate
  - Used in Alpha 21264, MIPS R10K
- Restrict page placement in OS
  - make sure index(VA) = index(PA)
  - Called page coloring
  - Used in many SPARC processors

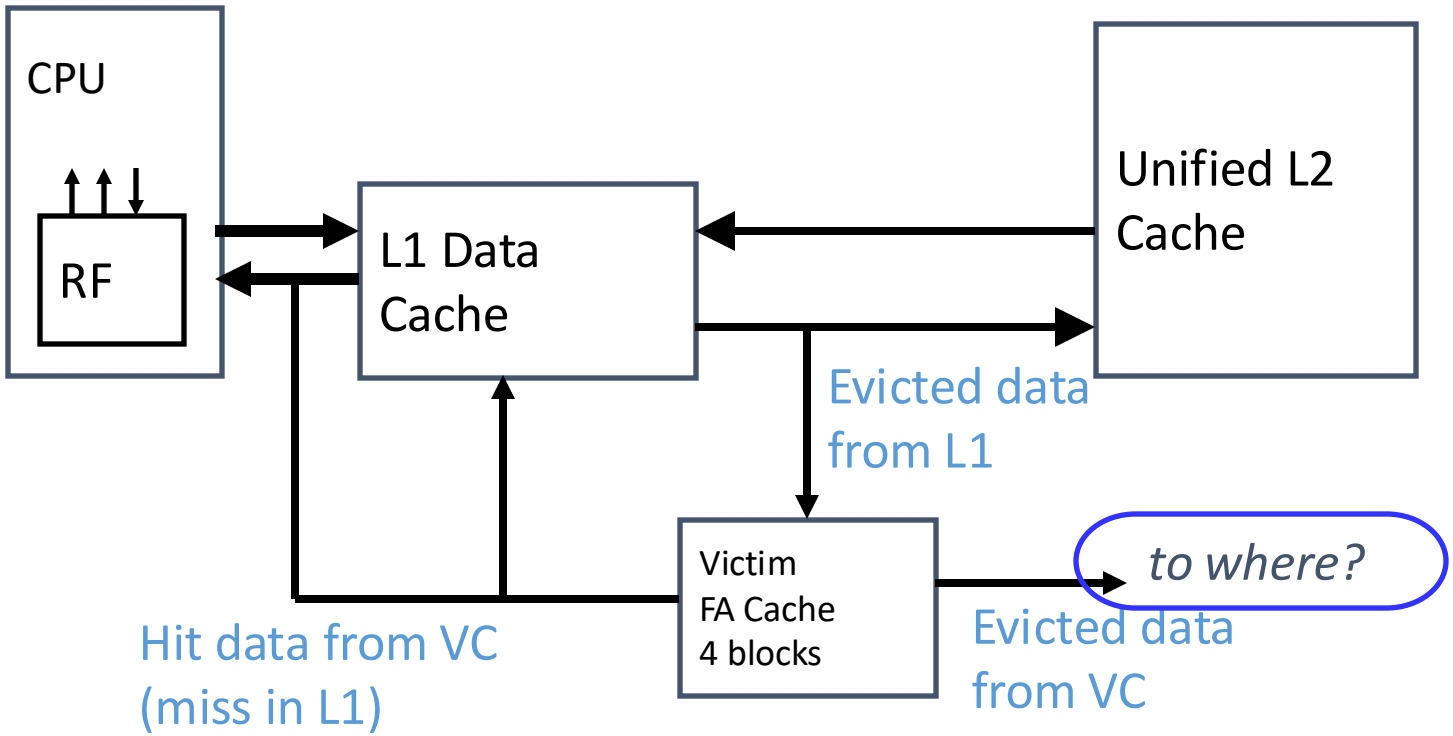
PIPT and VIVT

- Physically indexed, physically tagged (PIPT) caches use the physical address for both the index and the tag.
  - Simple to implement but slow, as the physical address must be looked up (which could involve a TLB miss and access to main memory) before that address can be looked up in the cache.
- Virtually indexed, virtually tagged (VIVT) caches use the virtual address for both the index and the tag.
  - Potentially much faster lookups.
  - Problems when several different virtual addresses may refer to the same physical address
    - Addresses would be cached separately despite referring to the same memory, causing coherency problems.
  - Additionally, there is a problem that virtual-to-physical mappings can change, which would require clearing cache blocks
- Can we have PIVT?

Cache Optimizations

- Small and simple first-level caches
  - Remember direct-mapped L1 cache of MIPS R4000?
- Victim cache
- Way prediction
- Non-blocking cache

Victim Caches (HP 7200)



Victim cache is a small associative *backup* cache, added to a direct-mapped cache, which holds recently evicted lines

- First look up in direct-mapped cache
- If miss, look in victim cache
- If hit in victim cache, swap hit line with line now evicted from L1
- If miss in victim cache, L1 victim -> VC, VC victim->?

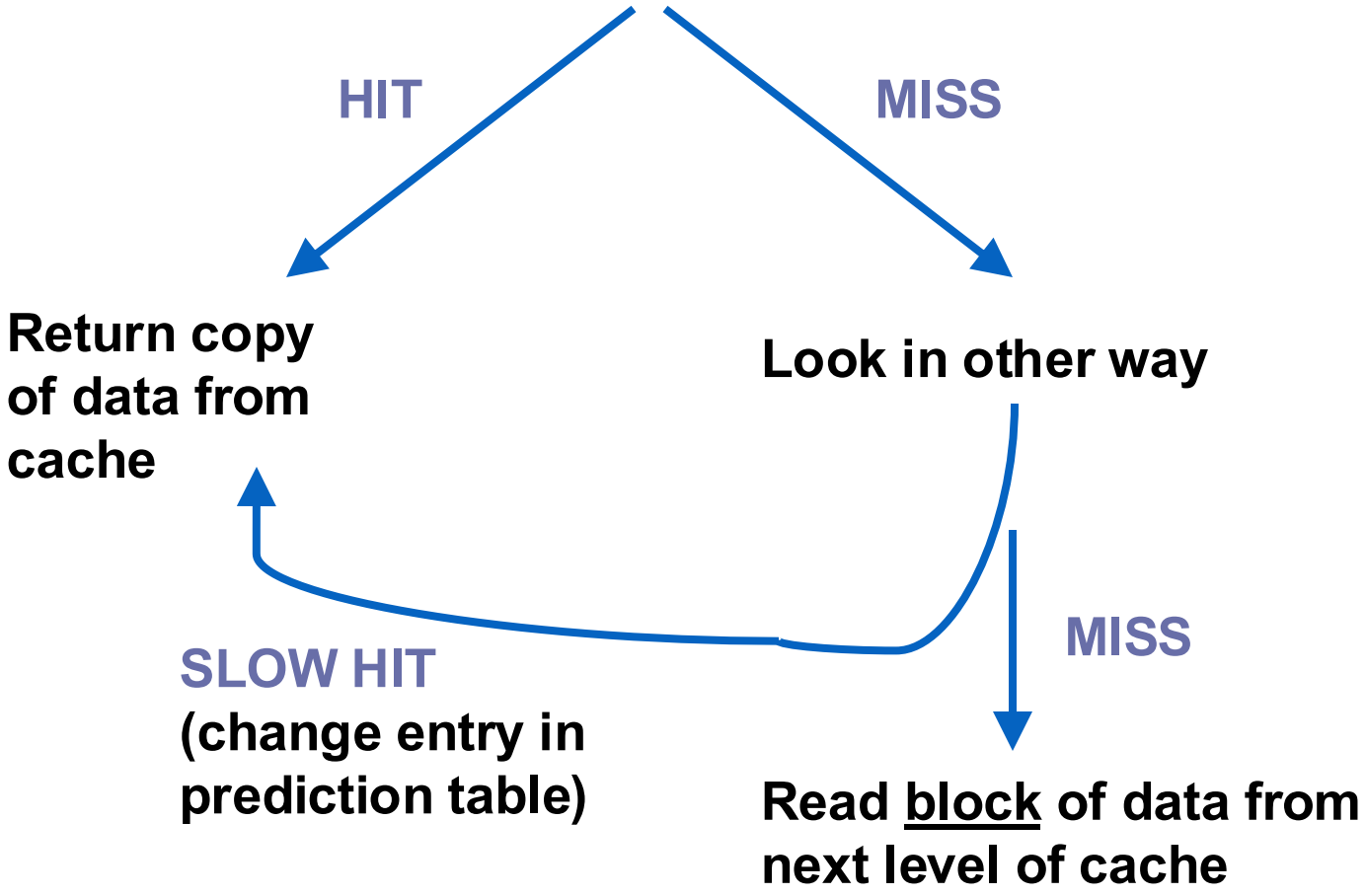
Fast hit time of direct mapped but with reduced conflict misses

Way prediction to reduce hit time

- Combine fast hit time of direct-mapped and the lower conflict misses of 2-way set-associative caches
  - Check one way first (speed of direct-mapped cache)
  - On a miss, check the other way, if hits, call it a pseudo-hit (or slow hit)
  - Way prediction is a bit to indicate which way to check first (changes dynamically)
- May extend prediction to more than 2-way set-associative caches
- Potentially saving power
  - Why?
- Drawback: CPU pipeline is hard if hit takes sometimes 1 and sometimes 2 cycles

Way-Predicting Caches (MIPS R10000 L2 cache)

- Use processor address to index into way-prediction table
- Look in predicted way at given index, then:



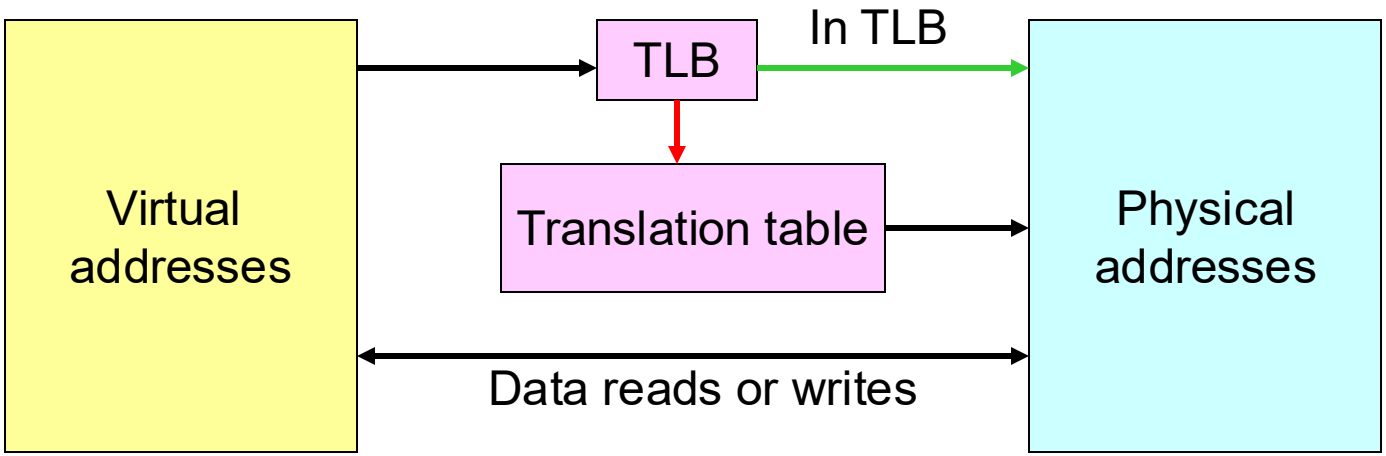
Increasing Cache Bandwidth with Non-Blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires Full/Empty bits on registers or out-of-order execution
- "*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, and can get miss to line with outstanding miss (secondary miss)
  - Requires pipelined or banked memory system (otherwise cannot support multiple misses)
  - Pentium Pro allows 4 outstanding memory misses
  - Cray X1E vector supercomputer allows 2,048 outstanding memory misses

TLB

- Translation lookaside buffer
- Caching on address translations
  - Tracks frequently used translations
  - Avoids translations in the common cases
- Process references the same page repeatedly
  - Translating each virtual address to physical address is wasteful

Caching Applied to Address Translation



Example of the TLB Content

Virtual page number (VPN)	Physical page number (PPN)	Control bits
2	1	Valid, rw
-	-	Invalid
0	4	Valid, rw

TLB Lookups

- One simple way
  - Sequential search of the TLB table
- ***Direct mapping***
  - assigns each virtual page to a specific slot in the TLB
  - e.g., use upper bits of virtual page number (VPN) to index TLB
- ***Set associativity:***
  - uses N TLB banks to perform lookups in parallel
- ***Fully associative cache:*** allows looking up all TLB entries in parallel

TLB Management

- Typically
  - Small TLBs are fully associative
  - Large TLBs and muti-level TLBs are common today
- Replacement of TLB Entries
  - Direct mapping
    - Entry replaced whenever a VPN mismatches
  - Associative caches
    - Random replacement
    - LRU (least recently used)
    - NMRU (not most recently used)
    - Depending on reference patterns
- Who does replacement?
  - Hardware-level
    - TLB replacement is mostly random, simple and fast
  - Software-level
    - Memory page replacements are more sophisticated
    - CPU cycles vs. cache hit rate

Consistency between TLB and Page Tables

- Different processes have different page tables
  - TLB entries need to be invalidated on context switches
  - Alternatives:
    - Tag TLB entries with process IDs
    - Additional cost of hardware and comparisons per lookup
- TLB Shootdown
  - Cause 1: the sharing of PTEs between cores
    - e.g., given a shared page, one core changes the page's permissions
    - e.g., when R/W pages become Read-only due to fork()
  - Cause 2: TLB coherence not maintained by hardware for x86
  - OS allows a core to flush a PTE in another core's TLB for coherence
    - Through Inter-processor interrupt (IPI)

Case studies

Recall we saw their caches in last lecture?

- Two-level TLBs of ARM Cortex A53

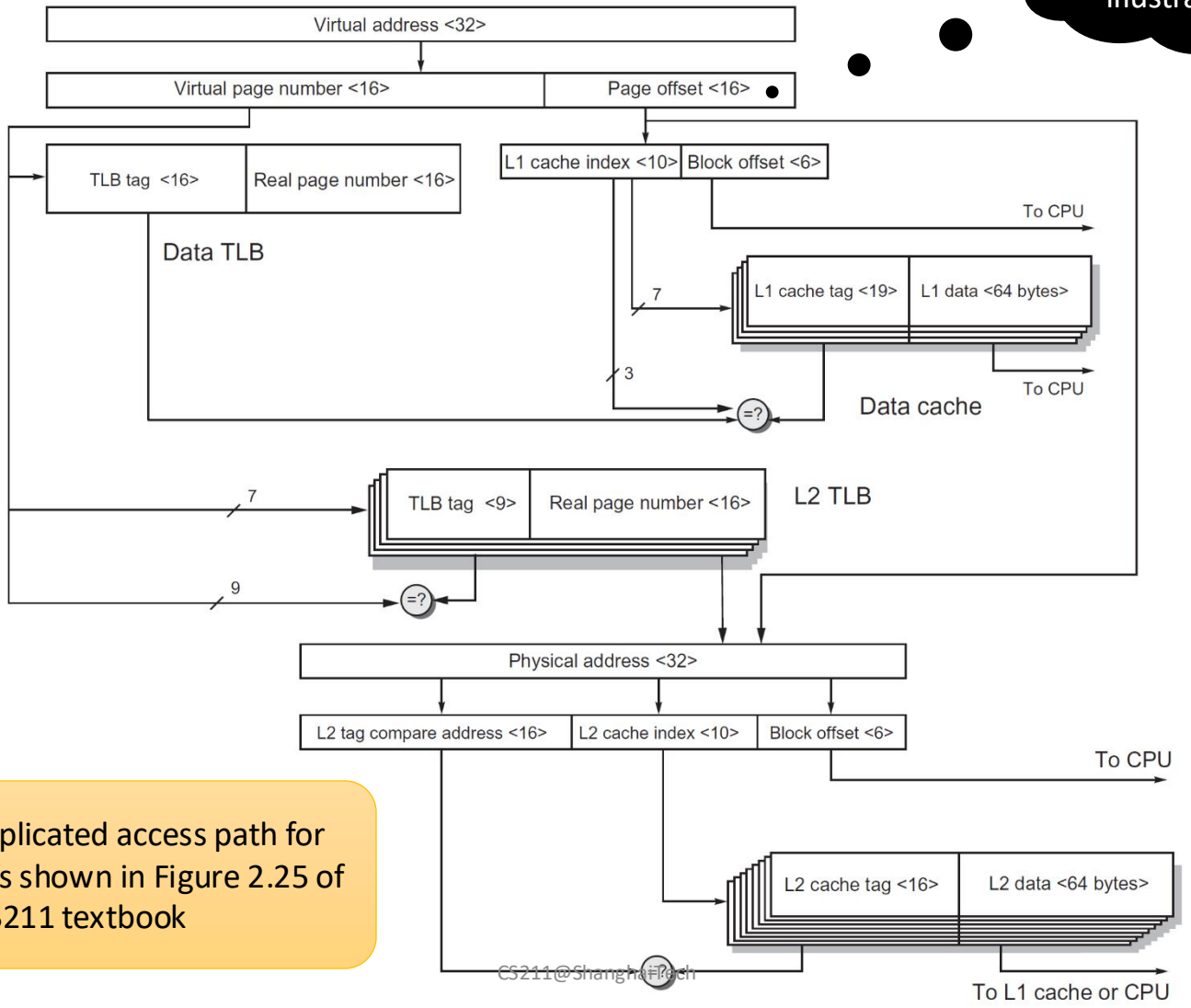
Structure	Size	Associativity
Instruction MicroTLB	10 entries	Fully associative
Data MicroTLB	10 entries	Fully associative
L2 Unified TLB	512 entries	4-way set associative

- Two-level TLBs of Intel i7 6700

Structure	Size	Associativity
Instruction TLB	128 entries	8-way set associative
Data TLB	64 entries	4-way set associative
L2 Unified TLB	1536 entries	12-way set associative

The data access path of Cortex A53

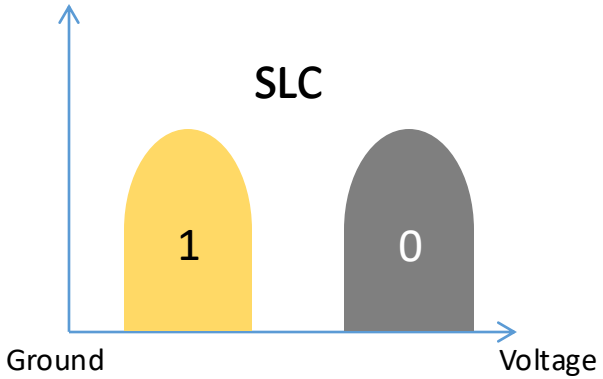
A page with 64KB for illustration

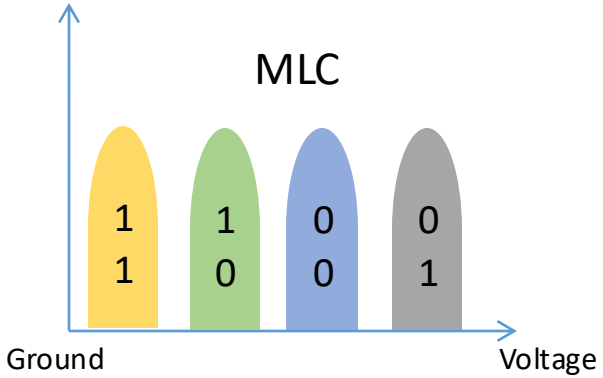


A more complicated access path for Intel i7 6700 is shown in Figure 2.25 of CS211 textbook

Flash Memory

- NAND flash and NOR flash
  - NAND flash is more prevalent today
- Widely used today
  - SSD
  - SD and Micro-SD cards
  - USB thumb drives
- SLC, MLC
  - Single-level Cell vs. Multi-level Cell





Trade-off emerges here again

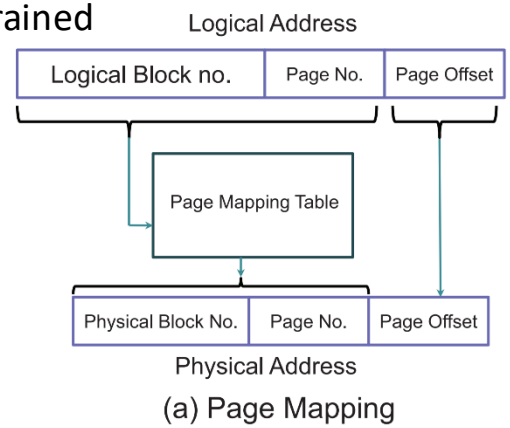
Source: https://dl.acm.org/doi/10.1145/3129257

Program/Erase

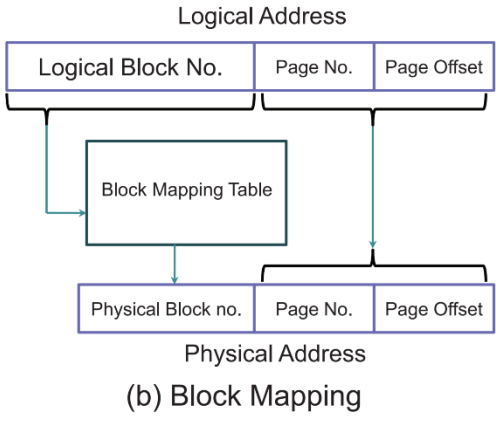
- Three operations
  - Program (write), Read, Erase
  - The unit of program and read is a flash page, from 2KB to 8KB.
  - The unit of erase is a flash block, composed of multiple pages, **much larger**
- Out-of-place update
  - A flash page cannot be reprogrammed unless the block where it is erased.
- Address translation
  - Flash translation layer between logical address to flash address
- Write endurance and bad block management
  - A cell undergoes a limited number of P/E flips
- Garbage collection
  - Invalid pages across flash blocks

Flash Translation Layer

- Mapping logical address from file system to flash address
- Two basic ones
  - Block-level mapping
    - One logical block mapped to one flash block
    - A small mapping table, coarse-grained
  - Page-level mapping
    - One logical page mapped to one flash page
    - A large mapping table, fine-grained
- Hybrid mapping
  - Inspired by CPU caching
- Demand-based mapping
  - Inspired by TLB



(a) Page Mapping



(b) Block Mapping

Hybrid mapping

- Block mapping + Page mapping
  - A majority of blocks as *data* blocks ← block mapping
  - A small part of blocks a *log* blocks ← page mapping
- Mapping steps on writing data to a page
  - First, do a block mapping, if page free, write down
  - Else, find a free log page in a log block, write down
  - No free log pages, pick a log block to merge it with corresponding data block(s)
- Associativity between log block and data block(s)
  - BAST: block associative sector translation
    - One log block exclusively dedicated to one data block
  - FAST: fully associative section translation
    - One log block shared by all data blocks
  - Set-associative translation?

Direct-mapped

Fully associative

https://ieeexplore.ieee.org/document/1010143
https://dl.acm.org/doi/10.1145/1275986.1275990

Acknowledgements

- These slides contain materials developed and copyright by:
  - Prof. An-I A. Wang (FSU)
  - Prof. Onur Mutlu (ETH Zurich)
  - Prof. Krste Asanovic (UC Berkeley)
  - Prof. Rami Melhem (Pitt)