ECE/CS 472/572 Computer Architecture: Virtual Memory

Prof. Lizhong Chen

Virtual Memory Motivations

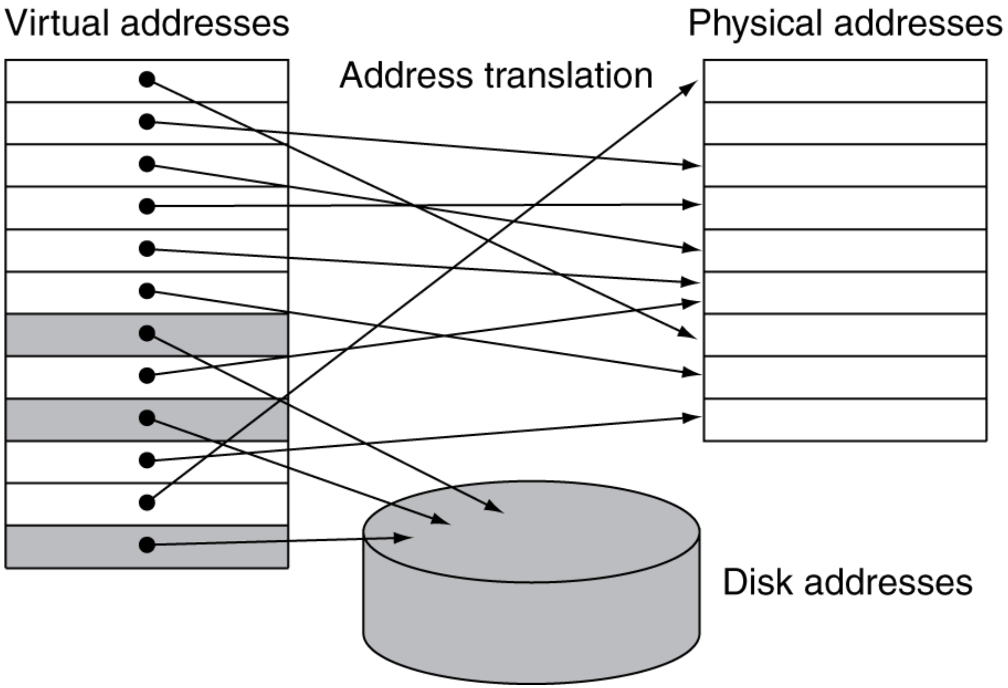
- Motivations
  - Programming burdens of small main memory
    - Programmers explicitly load/unload *overlays*
  - Sharing of memory among multiple programs
    - Efficient and safe
- Virtual Memory (VM) creates an illustration
  - Extremely large address space
  - Much faster than disk (as fast as main memory)

Virtual Memory Idea

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs

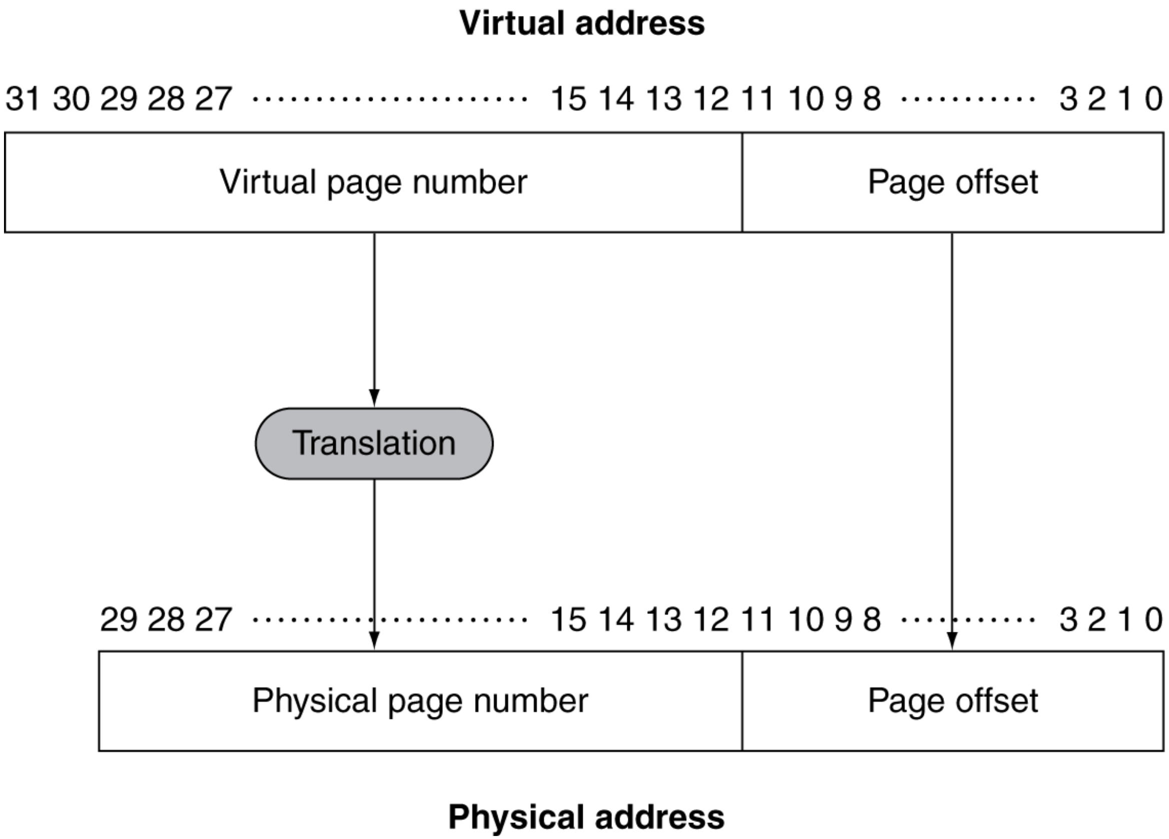
Virtual vs. Physical Address

- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page (e.g., 4KB size)
  - VM translation "miss" is called a page fault



Address Translation

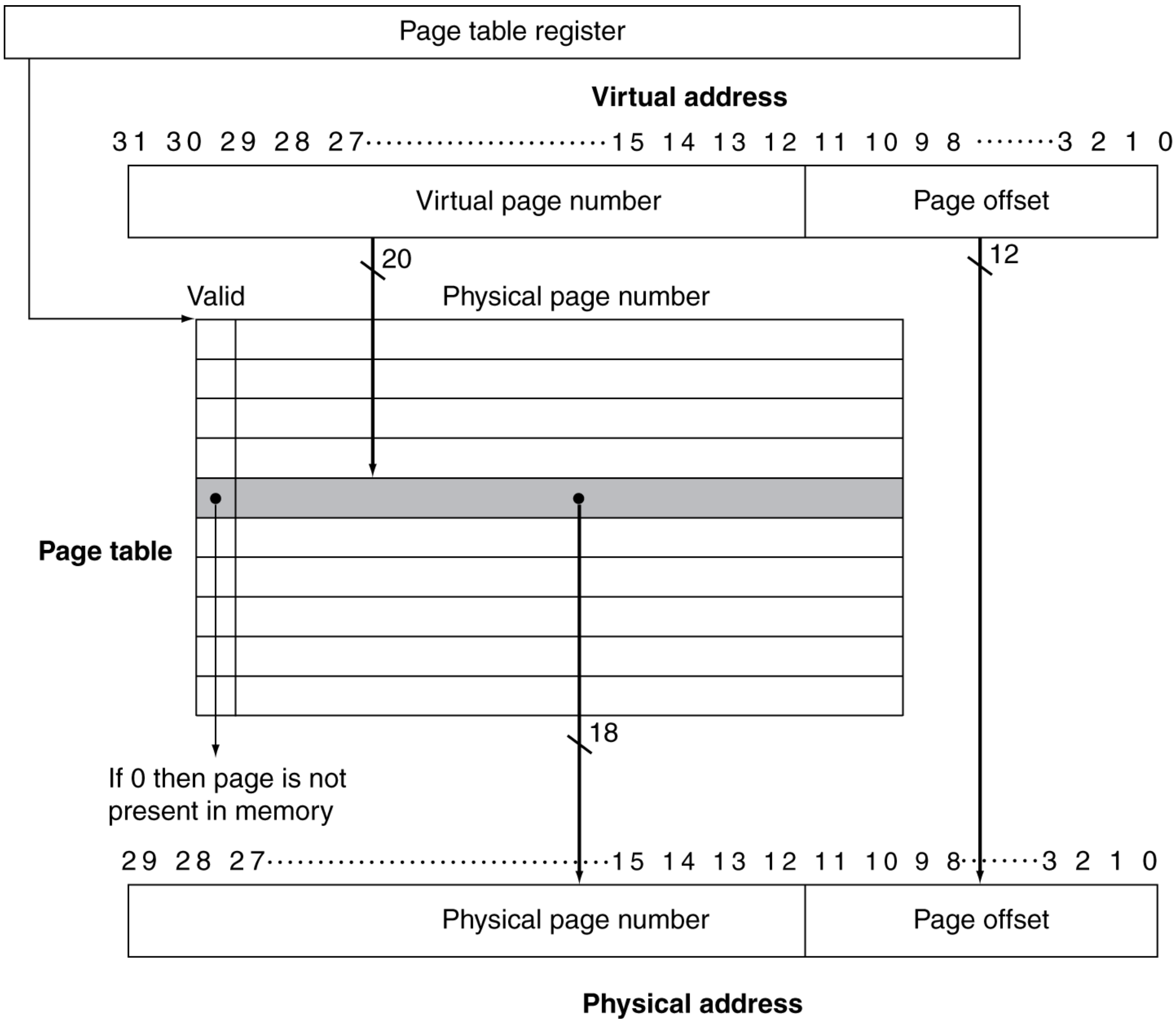
- Fixed-size pages (e.g., 4KB)



Page Tables

- Stores the VA=>PA mapping information
  - Page table is stored in physical memory
  - Array of page table entries (PTEs), indexed by virtual page number
  - Page table register points to page table in memory
- If a page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If a page is not present in memory
  - PTE can refer to location in swap space on disk

Translation Using a Page Table

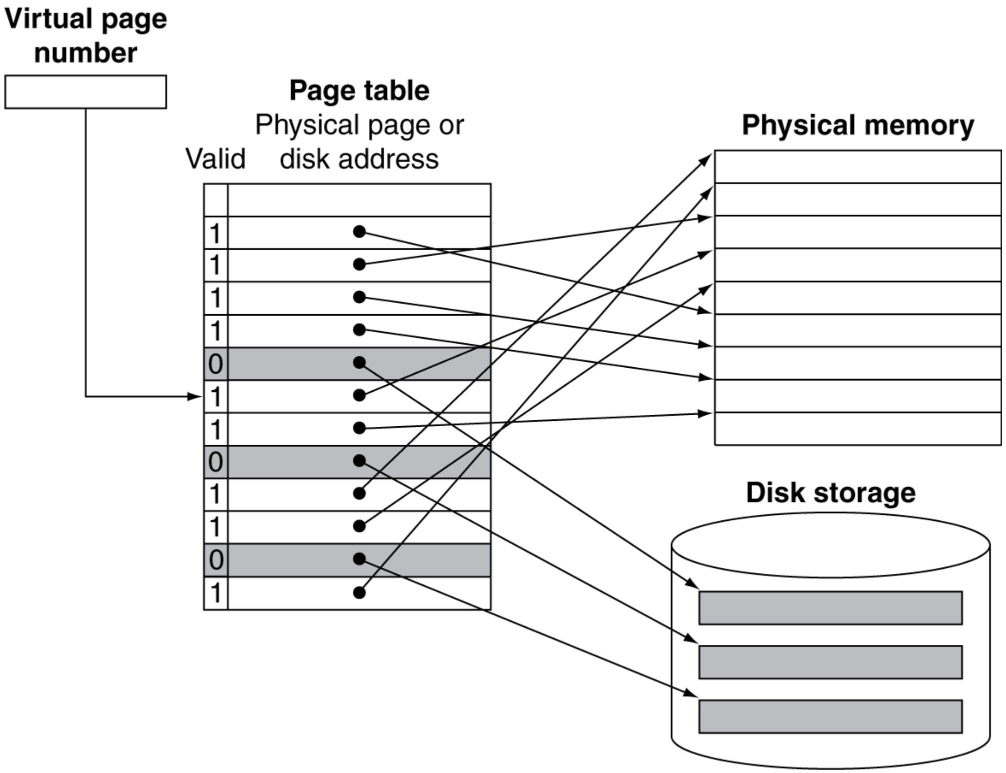


Example: Page Table Size

- 32-bit virtual address
- 4KB page size
- 4 bytes per page table entry (PTE)
- # of PTEs: $2^{32}/2^{12} = 2^{20}$ = 1 million
- Page table size: 1M x 4B = 4MB
- One page table per process

Mapping Pages to Storage

- Swap space: the space on disk created by OS for all the pages of a process
- Memory data is a subset of data in swap space



Page Fault Penalty

- On page fault, the page is fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Large page size
  - Fully associative placement
  - Smart replacement algorithms
  - Write-back instead of write-through

Replacement

- To reduce page fault rate
  - Prefer least-recently used (LRU) replacement
  - Reference bit in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

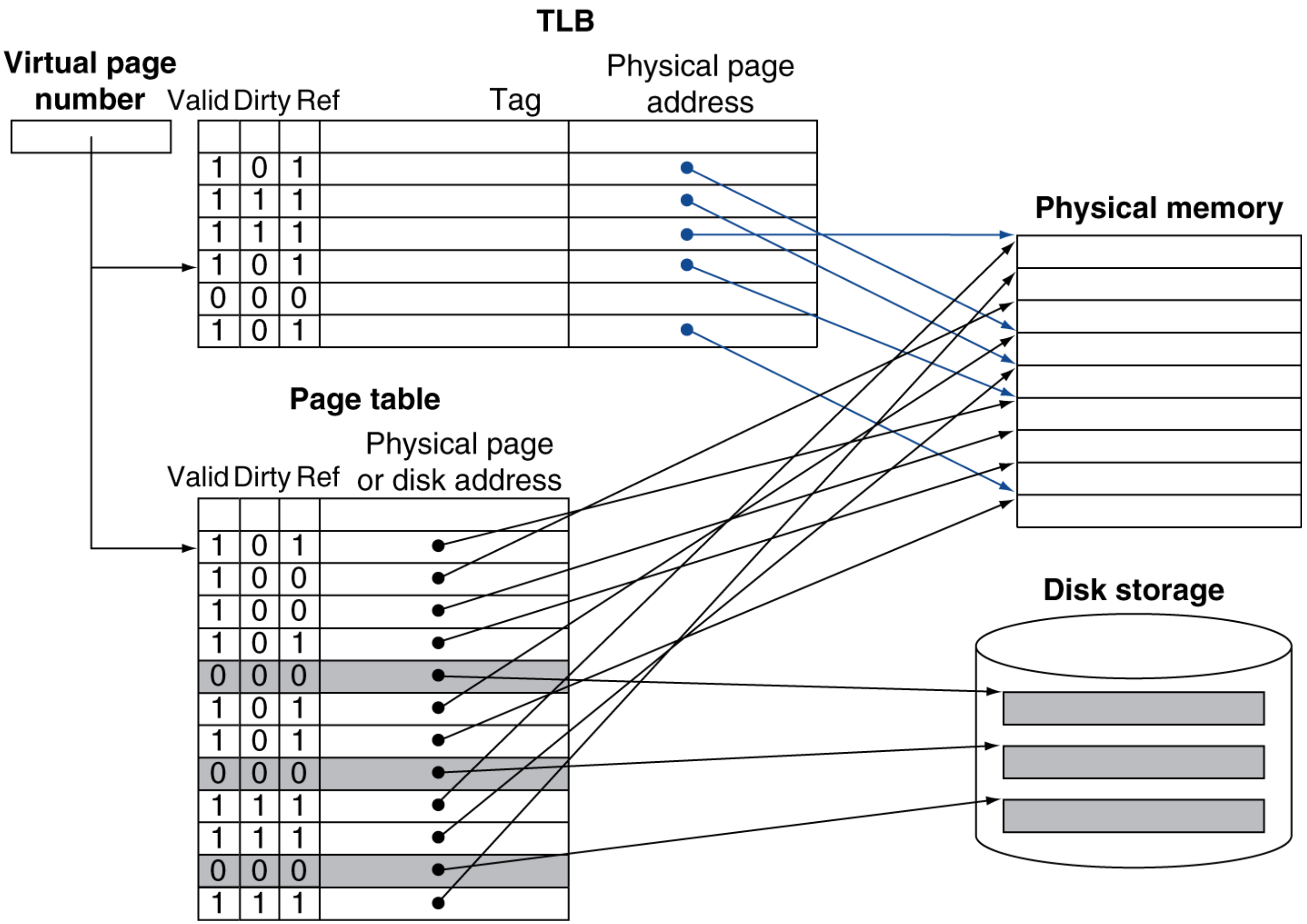
Writes

- Disk writes take millions of cycles
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE is set when page is written
  - Flush cache!

Fast Translation Using a TLB

- Problem: address translation may require an extra memory references
  - One to access the PTE in memory
  - Then the actual memory access
- However, access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1		●
1	1	1		●
1	1	1		●
1	0	1		●
0	0	0		
1	0	1		●

Page table

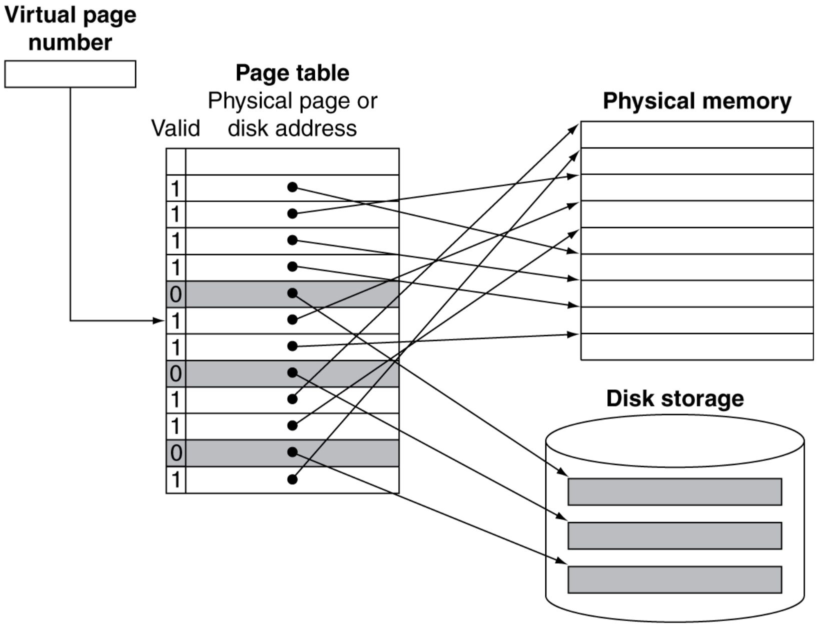
Valid	Dirty	Ref	Physical page or disk address
1	0	1	●
1	0	0	●
1	0	0	●
1	0	1	●
0	0	0	●
1	0	1	●
1	0	1	●
0	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

TLB Miss Handler

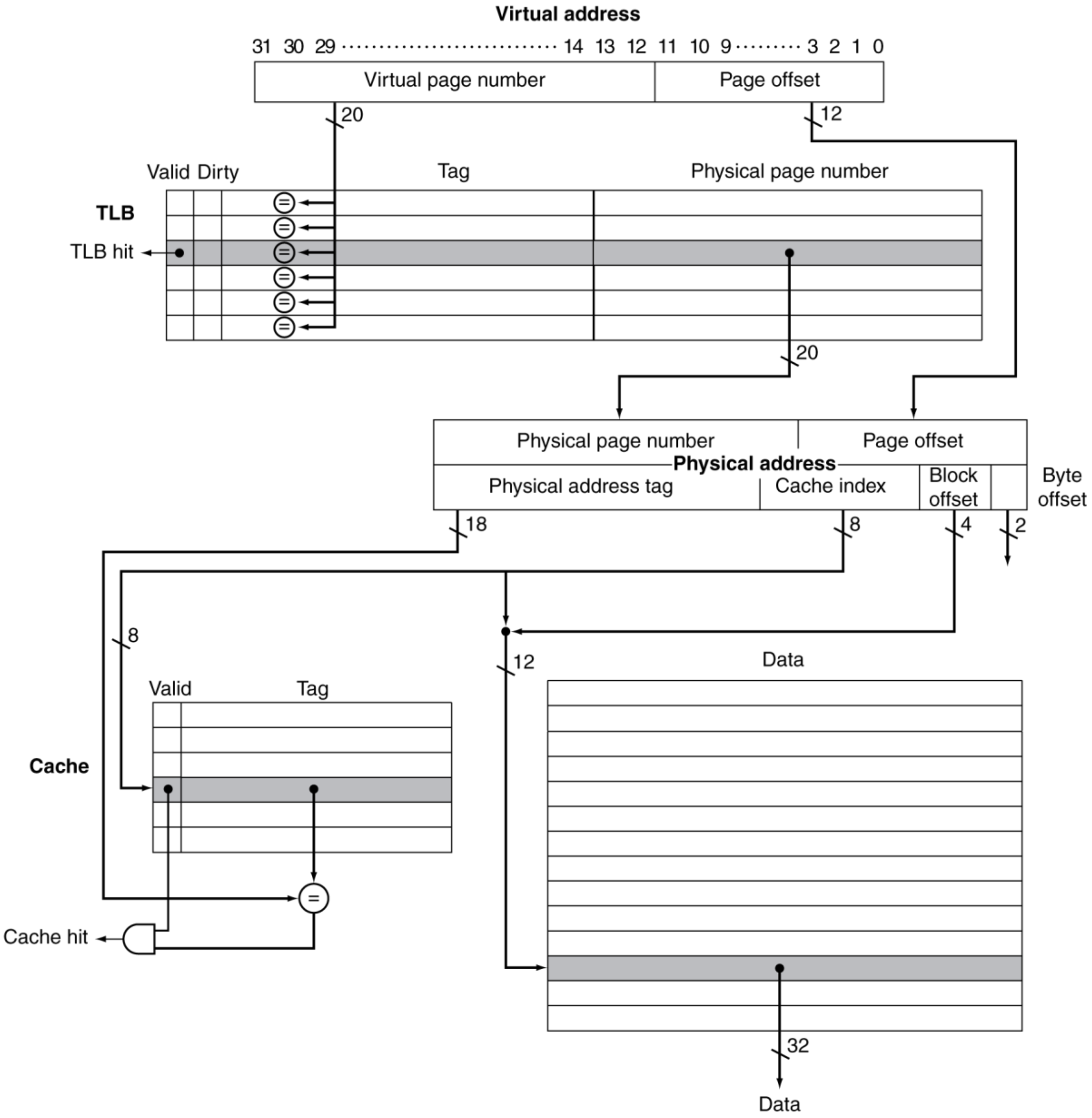
- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
  - Flush cache if needed
- Read page into memory
- Update page table
- Restart from faulting instruction

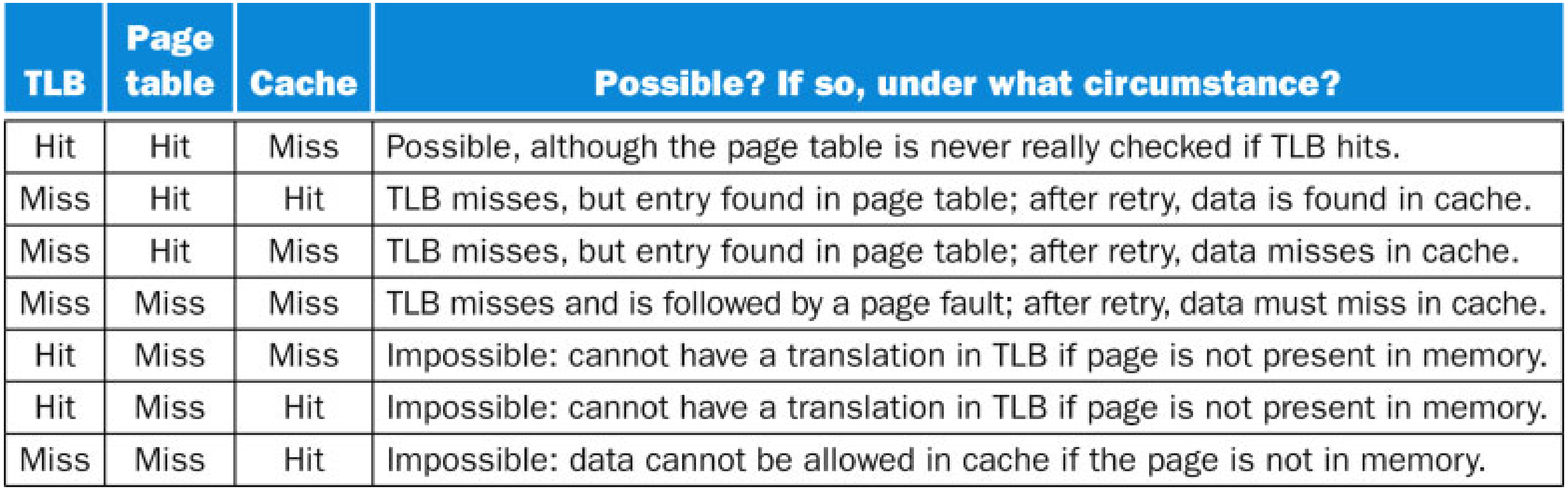


TLB in Intrinsity FastMATH



TLB Miss, Page Fault, Cache Miss

- A memory reference can encounter 3 types of misses: TLB miss, page fault, and cache miss
- Best case: hit in TLB & hit in cache
- Otherwise: 7 combinations



TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - Need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

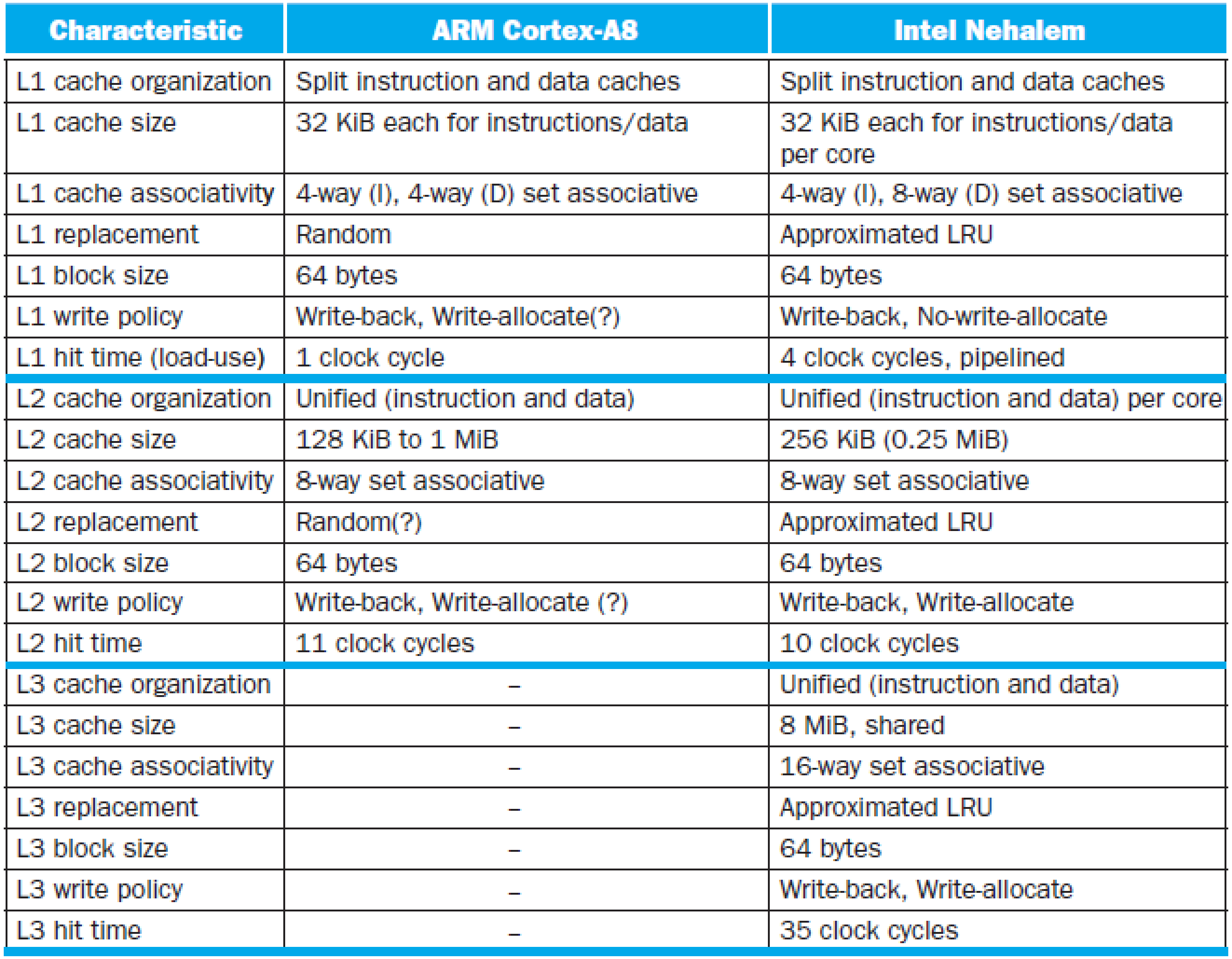
Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

Write Policy

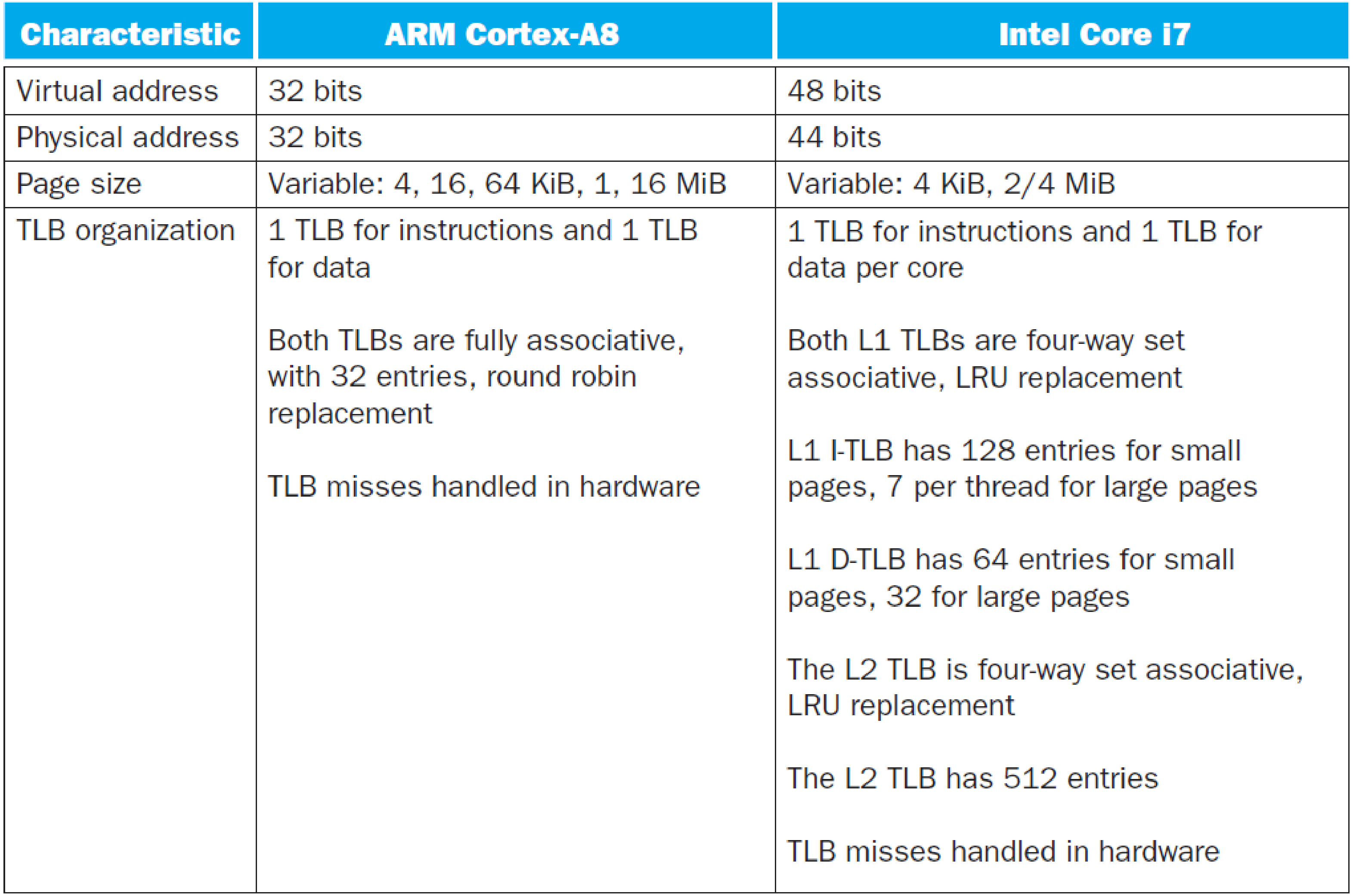
- Write-through
    - Update both upper and lower levels
    - Simplifies replacement, but may require write buffer
- Write-back
    - Update upper level only
    - Update lower level when block is replaced
    - Need to keep more state
- Virtual memory
    - Only write-back is feasible, given disk write latency

Multilevel On-Chip Caches



Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

2-Level TLB Organization



Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Memory Hierarchy Summary

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors