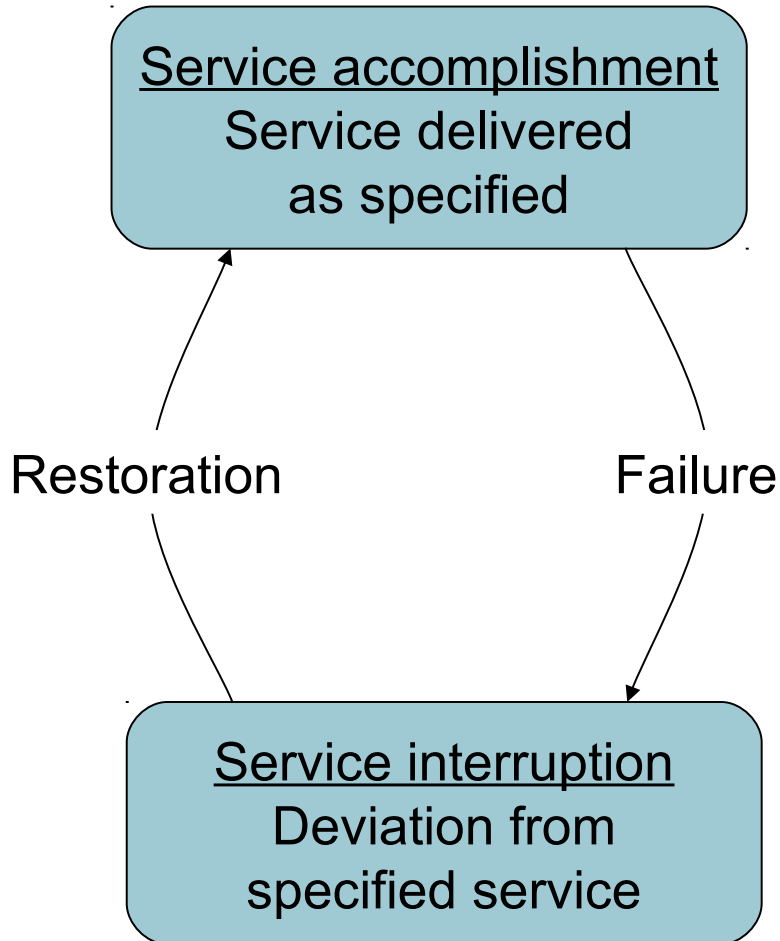


Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Part 4

Dependability



- Fault: failure of a component
 - May or may not lead to system failure

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
 - $MTBF = MTTF + MTTR$
- $Availability = MTTF / (MTTF + MTTR)$
- Improving Availability
 - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
 - Reduce MTTR: improved tools and processes for diagnosis and repair

The Hamming SEC Code

- Hamming distance
 - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection
 - E.g. parity code
- Minimum distance = 3 provides single error correction, 2 bit error detection

Encoding SEC

- To calculate Hamming code:
 - Number bits from 1 on the left
 - All bit positions that are a power 2 are parity bits
 - Each parity bit checks certain data bits:

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	
Parity bit coverage	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Decoding SEC

- Value of parity bits indicates which bits are in error
 - Use numbering from encoding procedure
 - E.g.
 - Parity bits = 0000 indicates no error
 - Parity bits = 1010 indicates bit 10 was flipped

SEC/DEC Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4
- Decoding:
 - Let H = SEC parity bits
 - H even, p_n even, no error
 - H odd, p_n odd, correctable single bit error
 - H even, p_n odd, error in p_n bit
 - H odd, p_n even, double error occurred
- Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits

Virtual Machines

- Host computer emulates guest operating system and machine resources
 - Improved isolation of multiple guests
 - Avoids security and reliability problems
 - Aids sharing of resources
- Virtualization has some performance impact
 - Feasible with modern high-performance computers
- Examples
 - IBM VM/370 (1970s technology!)
 - VMWare
 - Microsoft Virtual PC

Virtual Machine Monitor

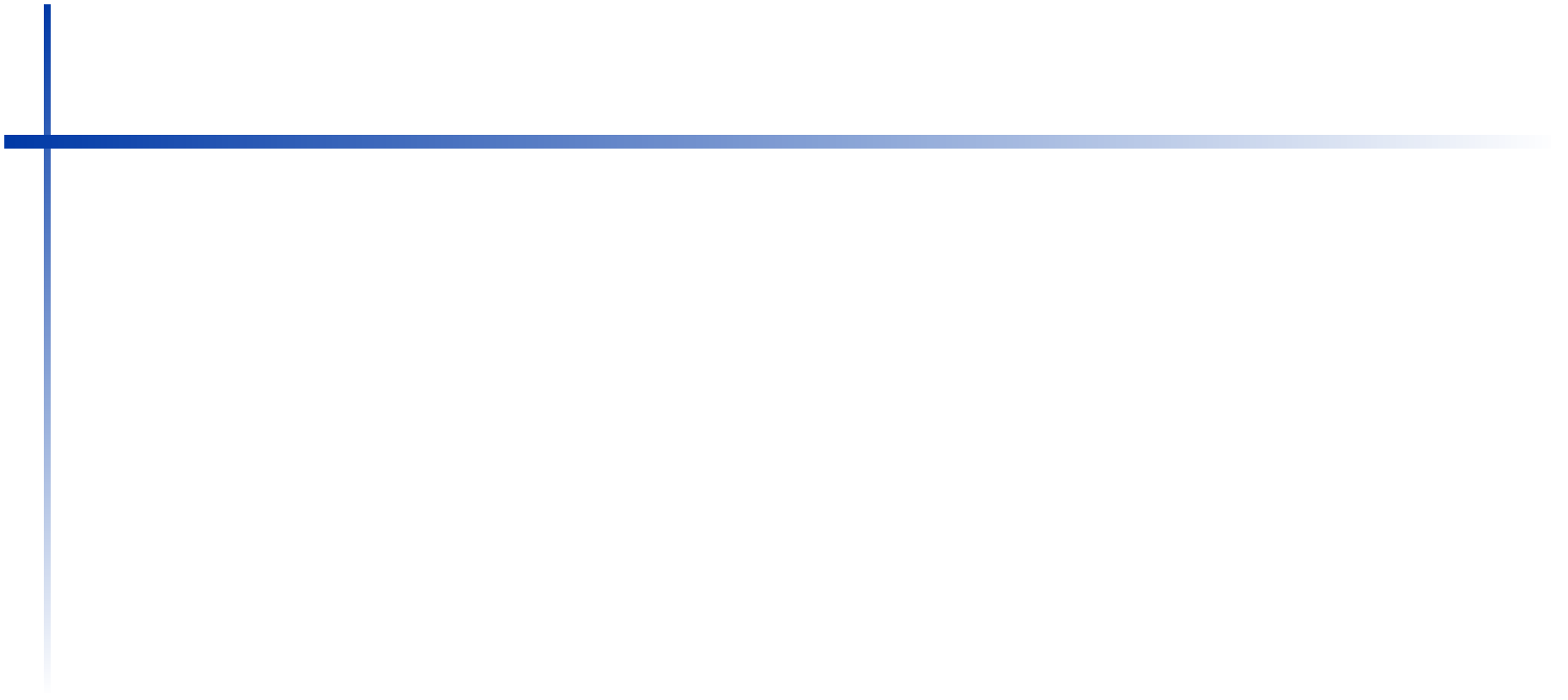
- Maps virtual resources to physical resources
 - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
 - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
 - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
 - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
 - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
 - VMM emulates a virtual timer
 - Emulates interrupt for VM when physical timer interrupt occurs

Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
 - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
 - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
 - Current ISAs (e.g., x86) adapting

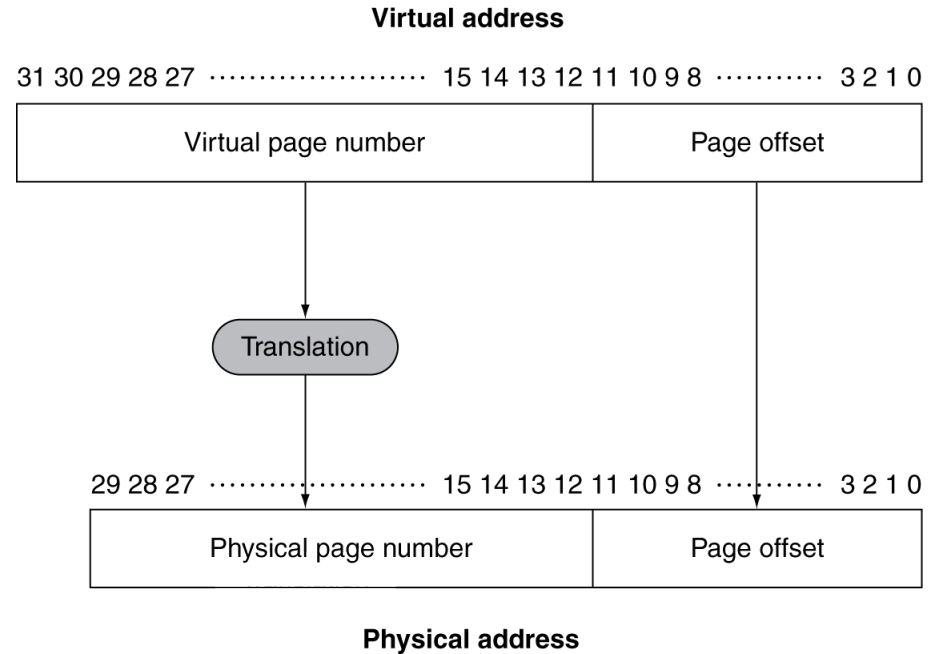
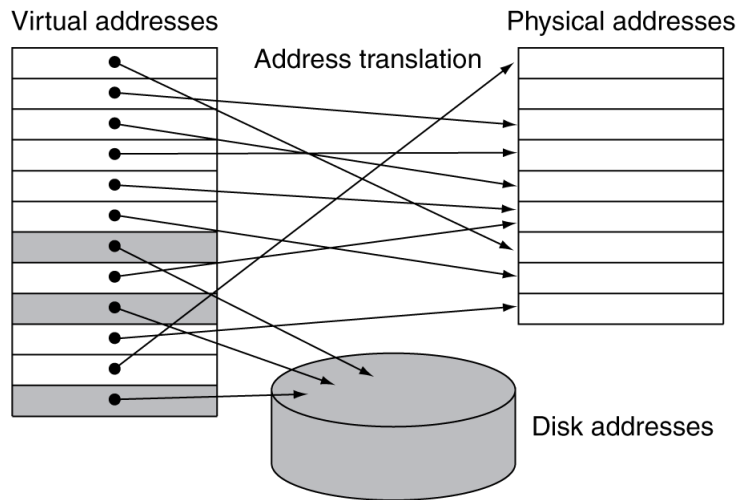


Virtual Memory

- 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
- CPU and OS translate virtual addresses to physical addresses
 - VM “block” is called a page
 - VM translation “miss” is called a page fault

Address Translation

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



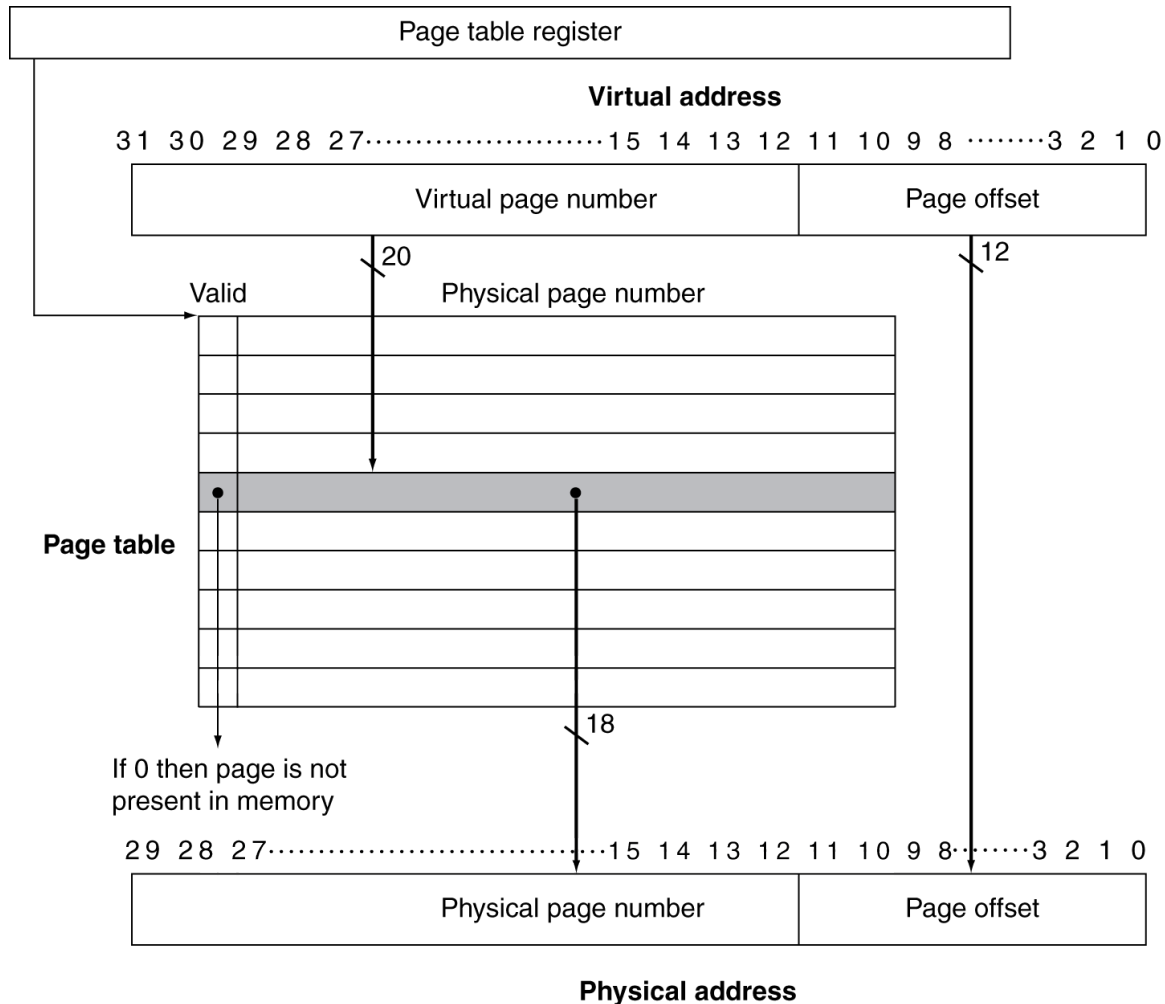
Page Fault Penalty

- On page fault, the page must be fetched from disk
 - Takes millions of clock cycles
 - Handled by OS code
- Try to minimize page fault rate
 - Fully associative placement
 - Smart replacement algorithms

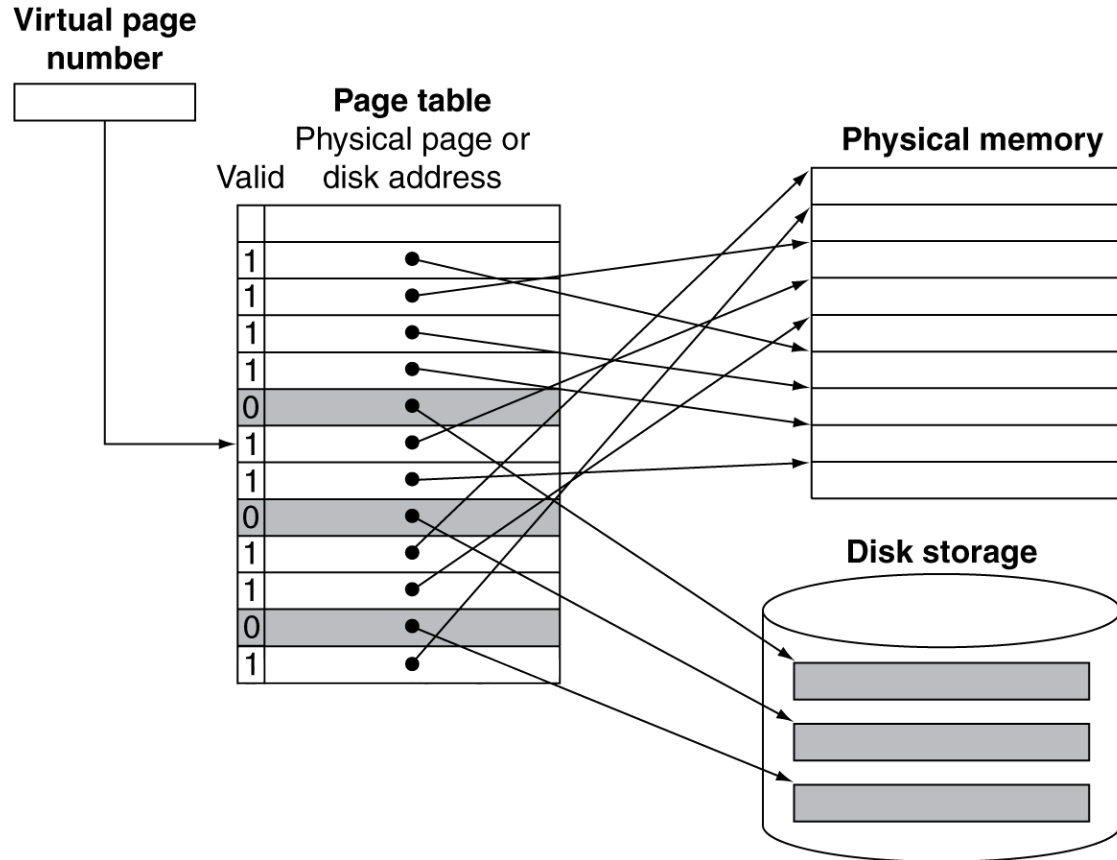
Page Tables

- Stores placement information
 - Array of page table entries(PTE), indexed by virtual page number
 - Page table register in CPU points to page table in physical memory
- If page is present in memory
 - PTE stores the physical page number
 - Plus other status bits (referenced, dirty, ...)
- If page is not present
 - PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



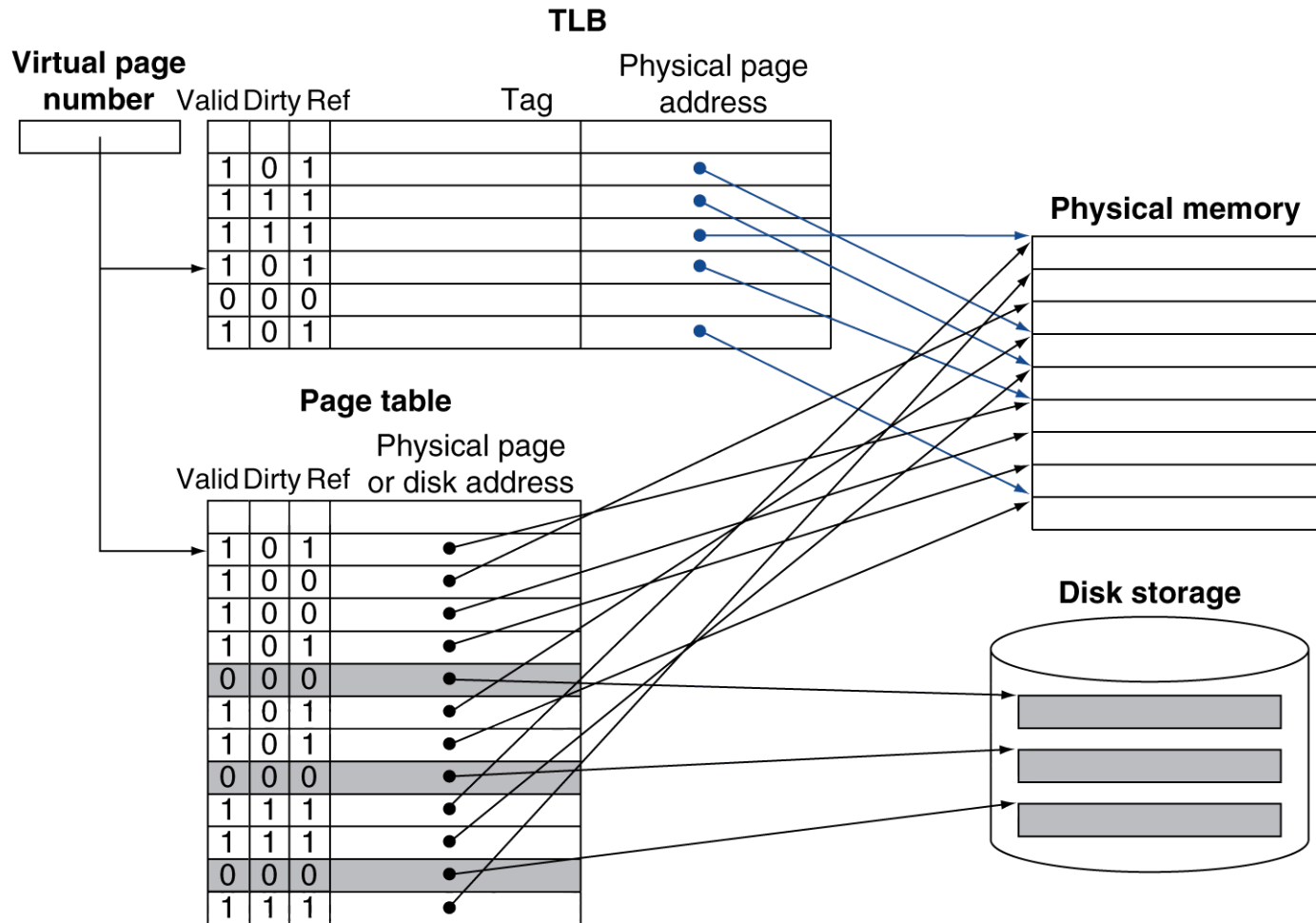
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
 - Reference bit (aka use 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
- Disk writes take millions of cycles
 - Block at once, not individual locations
 - Write through is impractical
 - Use write-back
 - Dirty bit in PTE set when page is written

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
 - One to access the PTE
 - Then the actual memory access
- But 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 Misses

- If page is in memory
 - Load the PTE from memory and retry
 - Could be handled in hardware
 - Can get complex for more 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

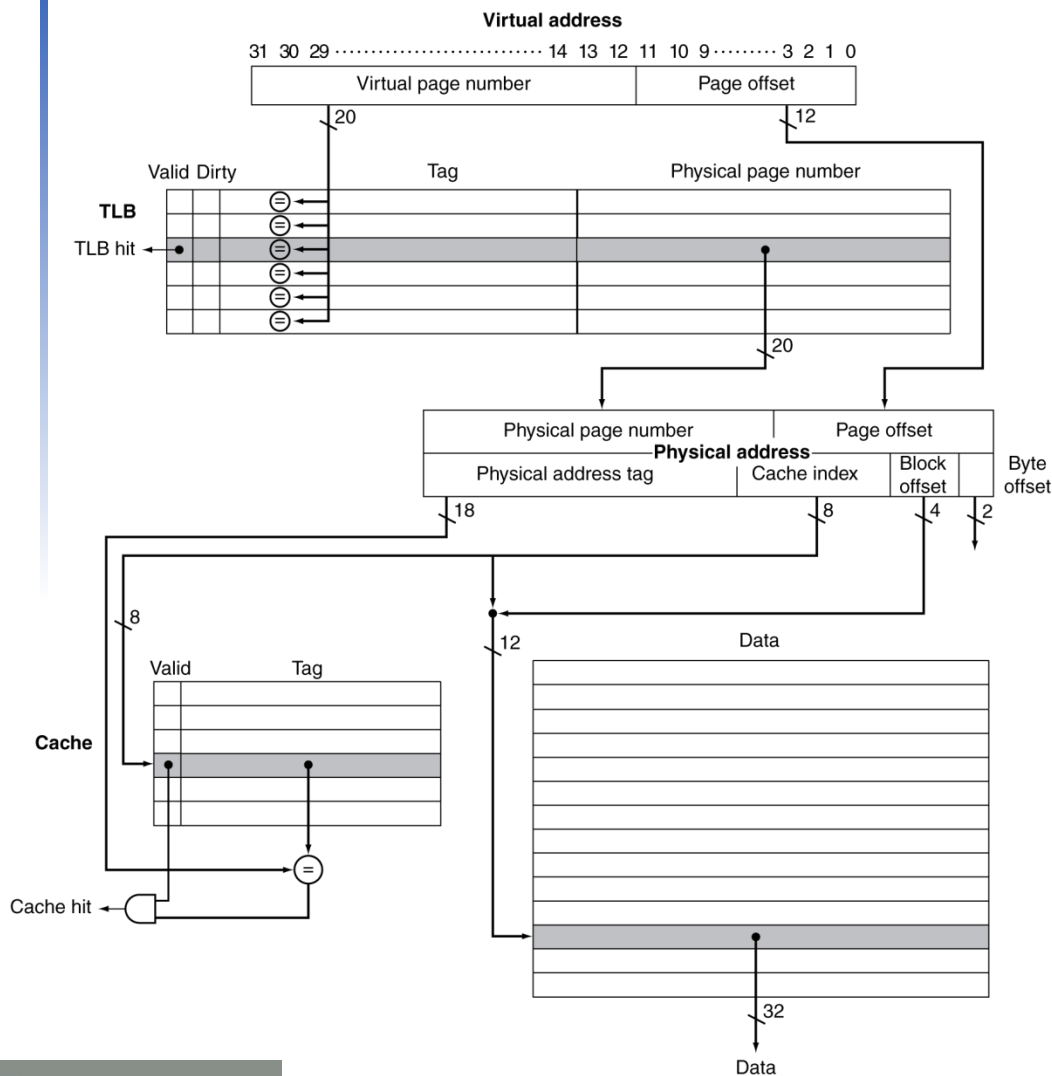
TLB Miss Handler

- TLB miss indicates
 - Page present, but PTE not in TLB
 - Page not present
- Must recognize TLB miss before destination register overwritten
 - Raise exception
- Handler copies PTE from memory to TLB
 - Then restarts instruction
 - If page not present, page fault will occur

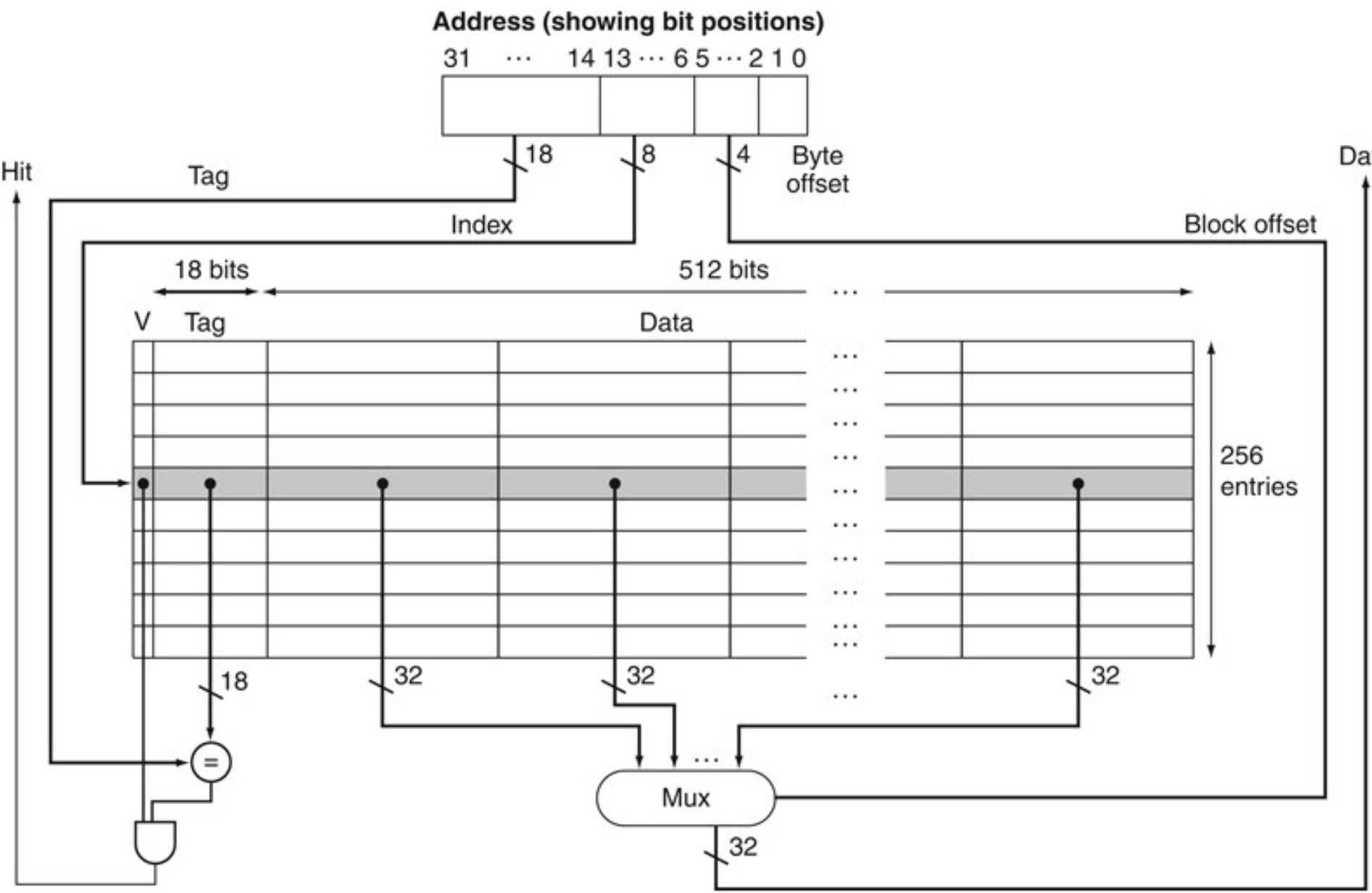
Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
 - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
 - Restart from faulting instruction

TLB and Cache Interaction



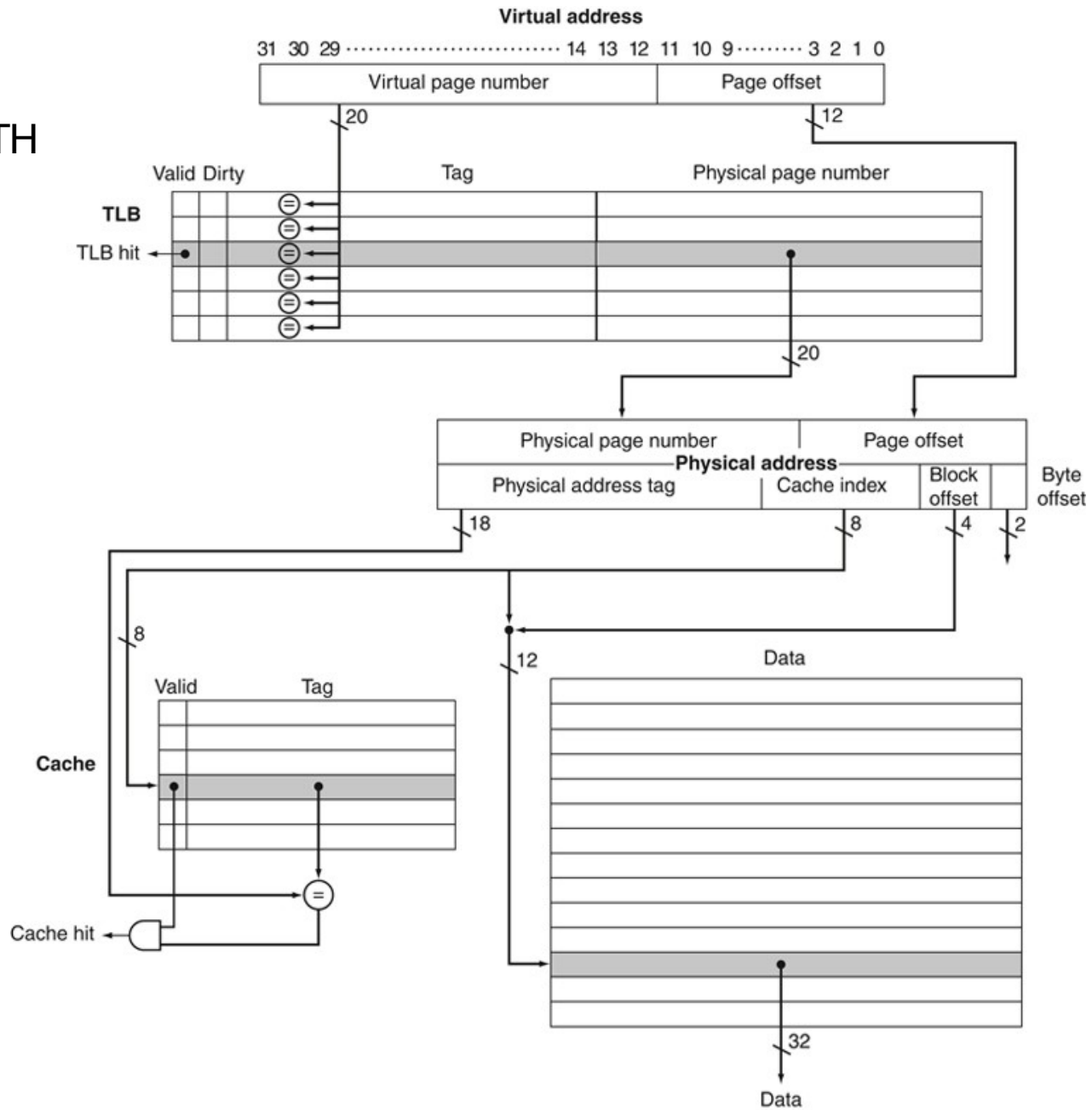
- If cache tag uses physical address
 - Need to translate before cache lookup
- Alternative: use virtual address tag
 - Complications due to aliasing
 - Different virtual addresses for shared physical address

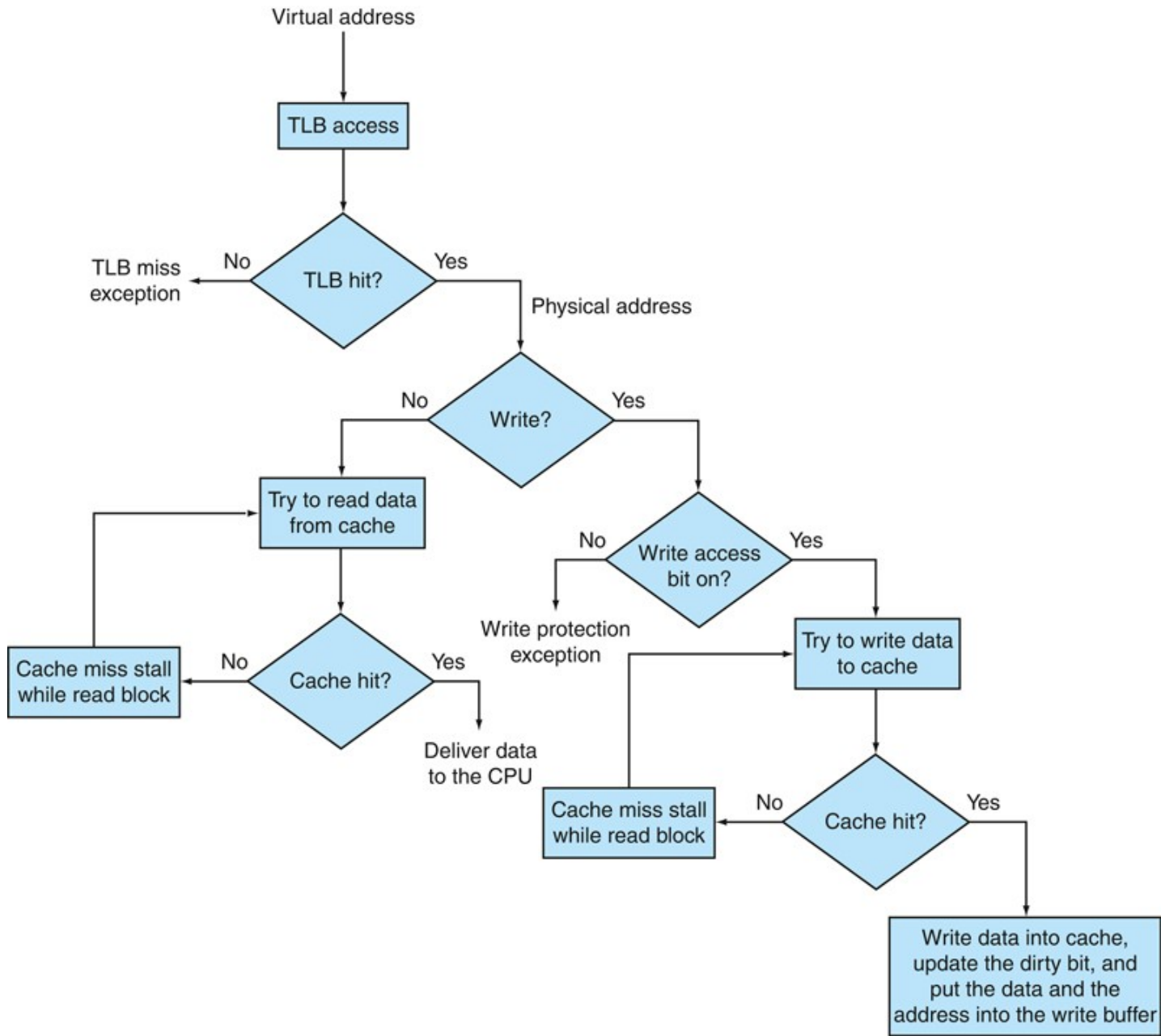


Copyright © 2021 Elsevier Inc. All rights reserved

Fig 5.12 FastMATH cache 16KiB

Fig 5.29
FastMATH
cpu





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
 - But 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

The BIG Picture

- 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

- Hardware caches
 - Reduce comparisons to reduce cost
- 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

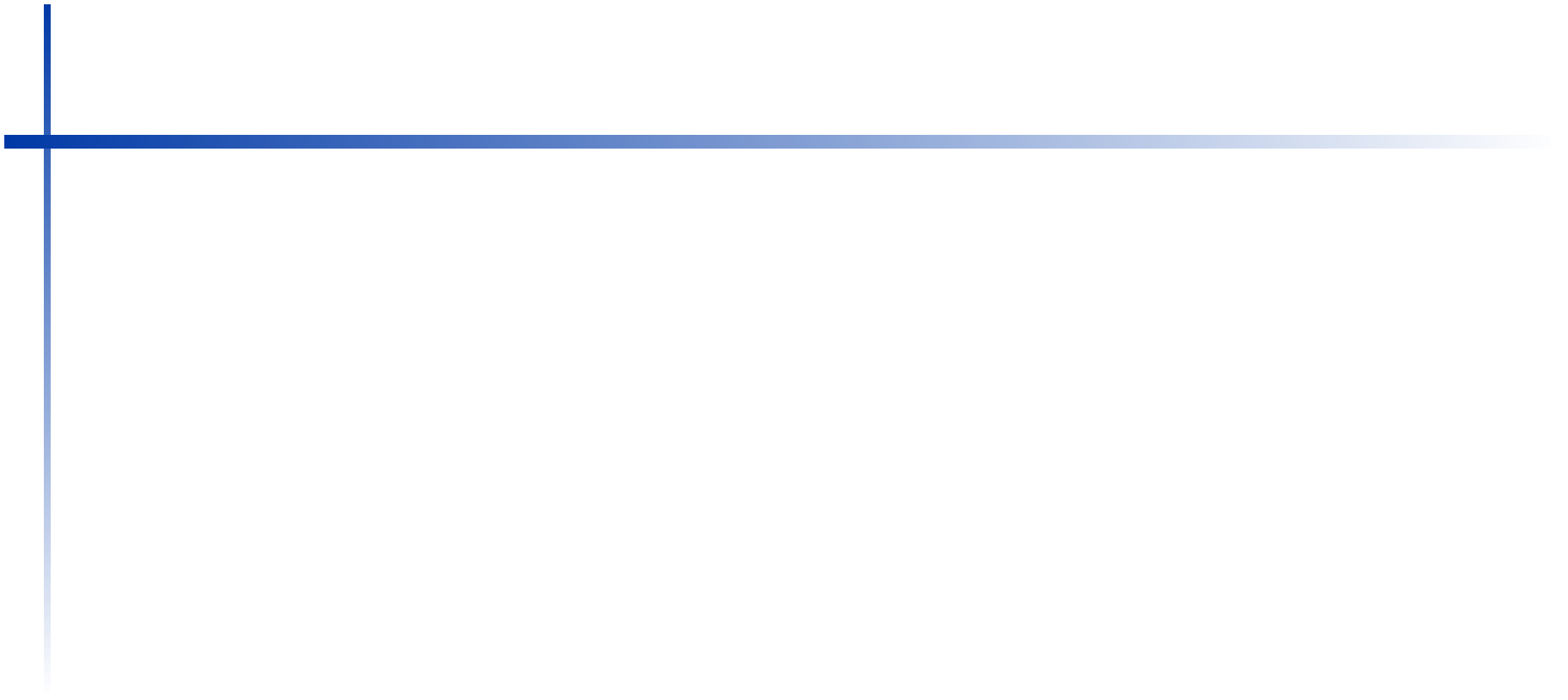
- Write-through
 - Update both upper and lower levels
 - Simplifies replacement, but may require write buffer
- Write-back
 - Update upper level only
 - Update lower level when block is replaced
 - Need to keep more state
- Virtual memory
 - Only write-back is feasible, given disk write latency

Sources of Misses

- Compulsory misses (aka cold start misses)
 - First access to a block
- Capacity misses
 - Due to finite cache size
 - A replaced block is later accessed again
- Conflict misses (aka collision misses)
 - In a non-fully associative cache
 - Due to competition for entries in a set
 - Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.



Pitfalls

- Byte vs. word addressing
 - Example: 32-byte direct-mapped cache, 4-byte blocks
 - Byte 36 maps to block 1
 - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
 - Example: iterating over rows vs. columns of arrays
 - Large strides result in poor locality

Pitfalls

- In multiprocessor with shared L2 or L3 cache
 - Less associativity than cores results in conflict misses
 - More cores \Rightarrow need to increase associativity
- Using AMAT to evaluate performance of out-of-order processors
 - Ignores effect of non-blocked accesses
 - Instead, evaluate performance by simulation

Pitfalls

- Extending address range using segments
 - E.g., Intel 80286
 - But a segment is not always big enough
 - Makes address arithmetic complicated
- Implementing a VMM on an ISA not designed for virtualization
 - E.g., non-privileged instructions accessing hardware resources
 - Either extend ISA, or require guest OS not to use problematic instructions

Concluding Remarks

- 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