Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Why is miss rate not a good metric for evaluating cache performance? What is the appropriate metric?
- The ultimate metric for cache performance is average access time:

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

  - The miss rate is only one component of this equation.
  - A cache may have a low miss rate, but an extremely high penalty per miss, making it lower-performing than a cache with a higher miss rate but a substantially lower miss penalty
  - Alternatively, it may have a low miss rate but a high hit time (this is true for large fully associative caches, for instance).

Multilevel Caches- cont'd

- Multiple levels of cache are used for exactly this reason.
  - Not all of the performance factors can be optimized in a single cache.
  - Specifically, with miss penalty (memory latency) given, it is difficult to design a cache which is both fast in the common case (a hit) and minimizes the costly uncommon case by having a low miss rate.

- The first level cache minimizes the hit time, therefore it is usually small with a low-associativity.

- The second level cache minimizes the miss rate, it is usually large with large blocks and a higher associativity.

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
- Total CPI=Base CPI+ Mem-stall cycles/ins
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

(2%-0.5%)x20 = 0.3 L2 hit portion

0.5(20+420)= 2.1, the sum, 1+0.3+2.1=3.4

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

How is the Hierarchy Managed?

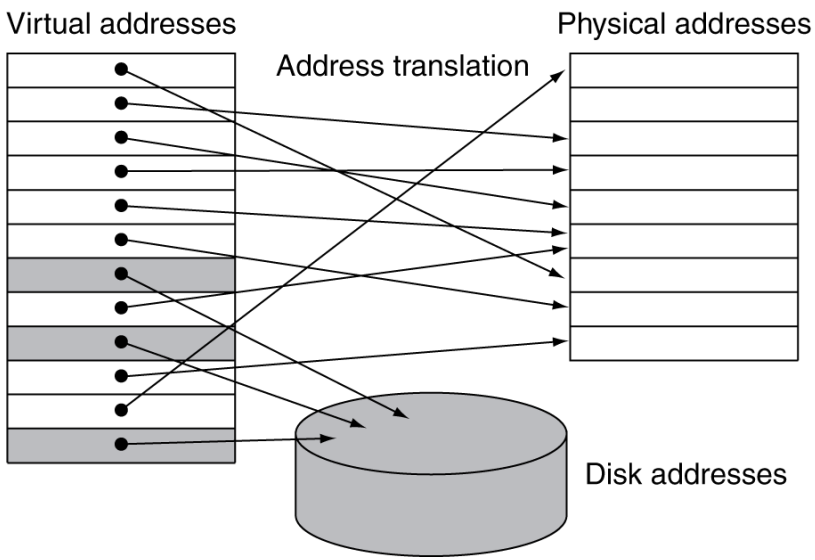
- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

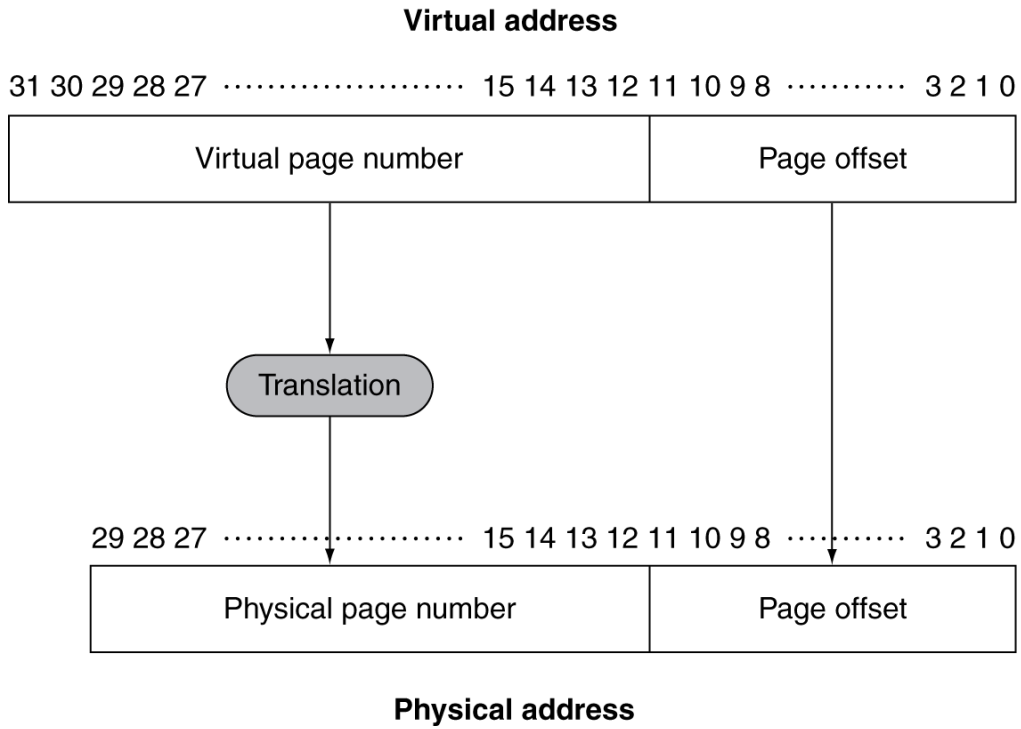
Virtual Memory

- 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)





-Pages should be large enough to amortize access time – typically 4 to 16K, servers 32-64k, embedded systems 1K
-Organization that reduces the page fault- fully associative placement-pages->mem
-Page faults can be handled by software : clever algorithm– to reduce miss rate
-Write through will not work- takes very long time, use write-back

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(PTEs), indexed by virtual page number
  - Page table register(PTR) 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

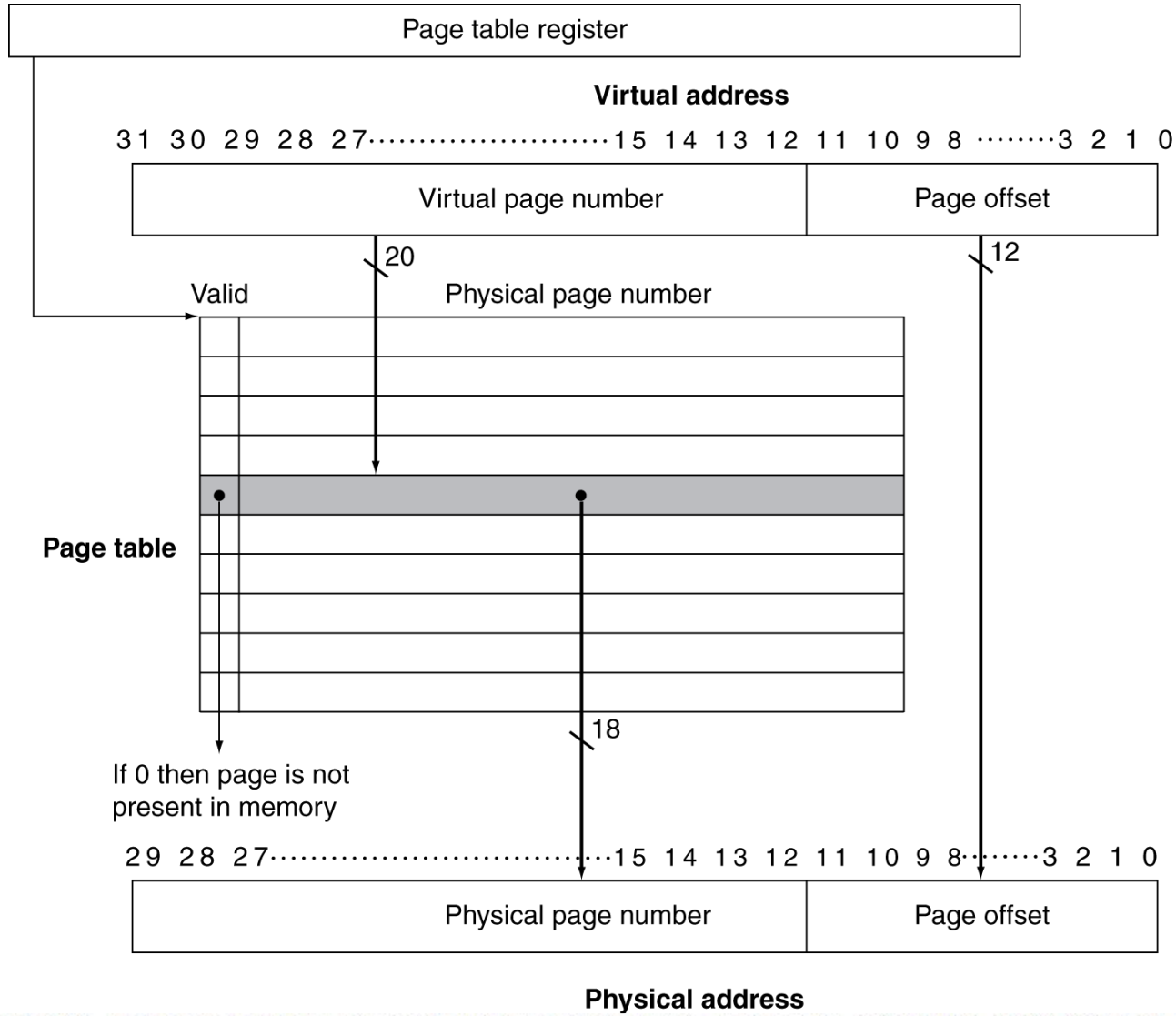
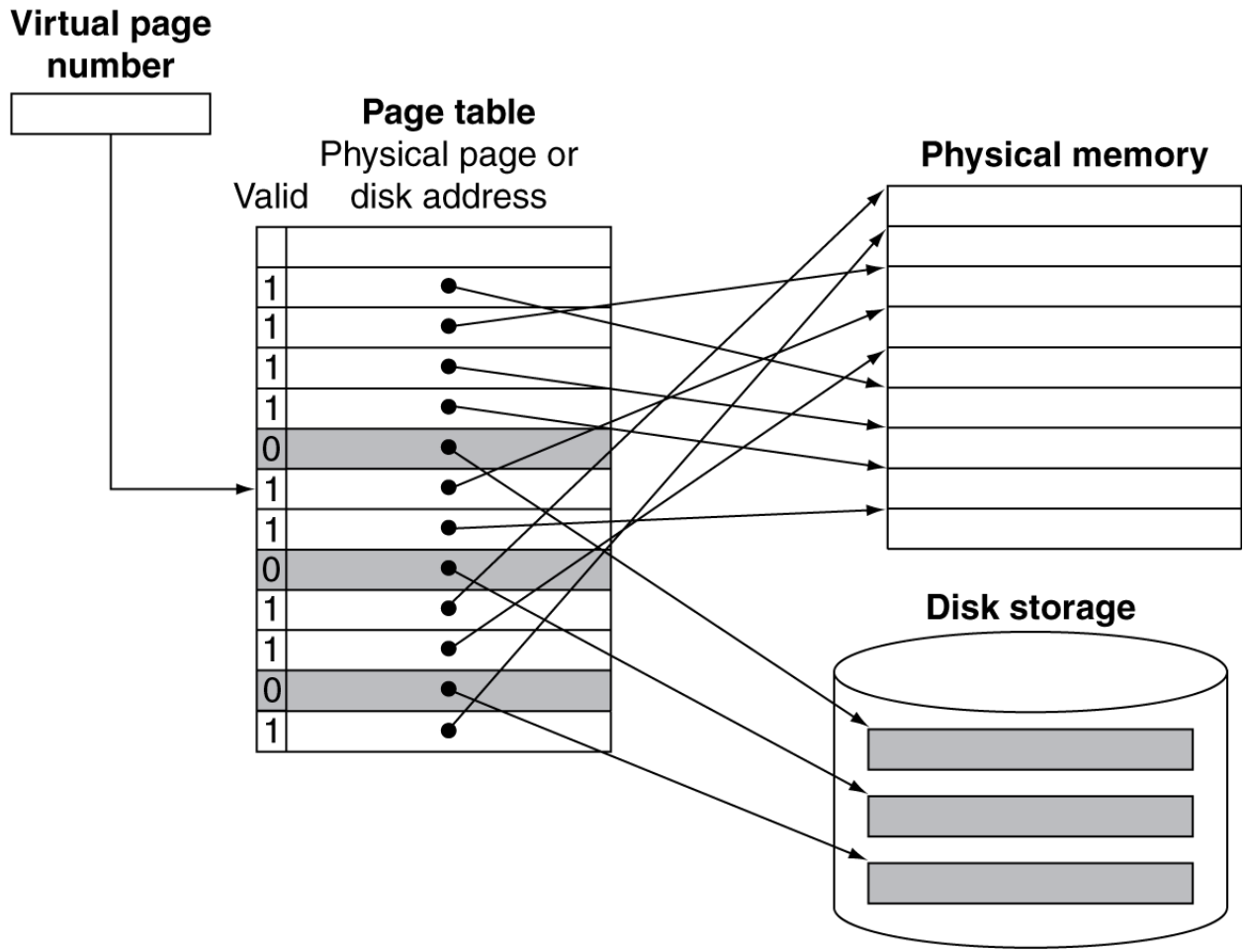


FIGURE 5.21 The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address. We assume a 32-bit address. The starting address of the page table is given by the page table pointer. In this figure, the page size is 2^{12} bytes, or 4 KB. The virtual address space is 2^{32} bytes, or 4 GB, and the physical address space is 2^{30} bytes, which allows main memory of up to 1 GB. The number of entries in the page table is 2^{20}, or 1 million entries. The valid bit for each entry indicates whether the mapping is legal. If it is off, then the page is not present in memory. Although the page table entry shown here need only be 19 bits wide, it would typically be rounded up to 32 bits for ease of indexing. The extra bits would be used to store additional information that needs to be kept on a per-page basis, such as protection.

Mapping Pages to Storage



Elaboration: With a 32-bit virtual address, 4 KB pages, and 4 bytes per page table entry, we can compute the total page table size:

$$\text{Number of page table entries} = \frac{2^{32}}{2^{12}} = 2^{20}$$

$$\text{Size of page table} = 2^{20} \text{ page table entries} \times 2^{2} \frac{\text{bytes}}{\text{page table entry}} = 4 \text{ MB}$$

Replacement and Writes

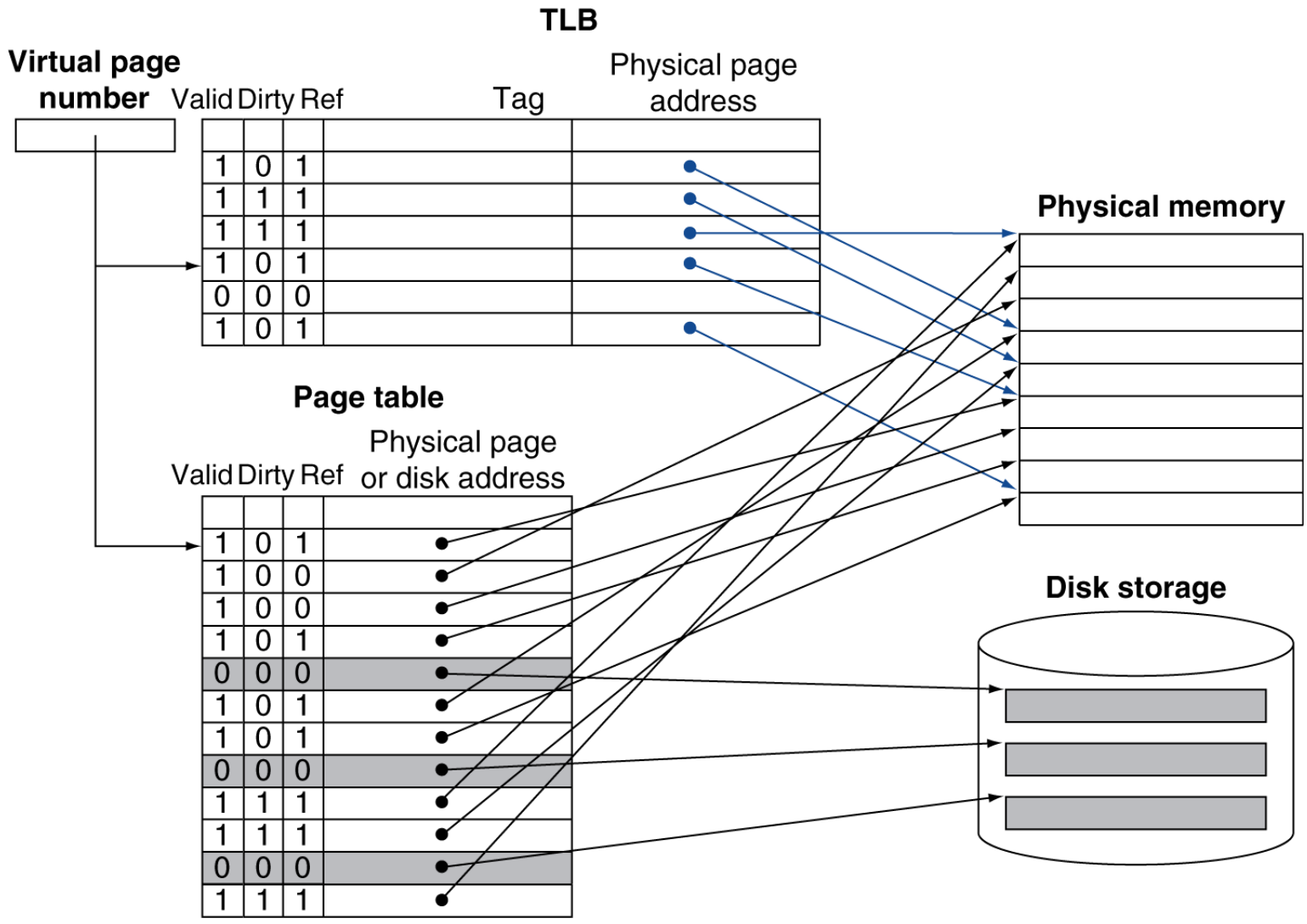
- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - *Reference bit* (aka *use bit*) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical-Use write-back
    - (Disk transfer time is smaller than access time-copying entire page is more efficient than writing individual words)
  - *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

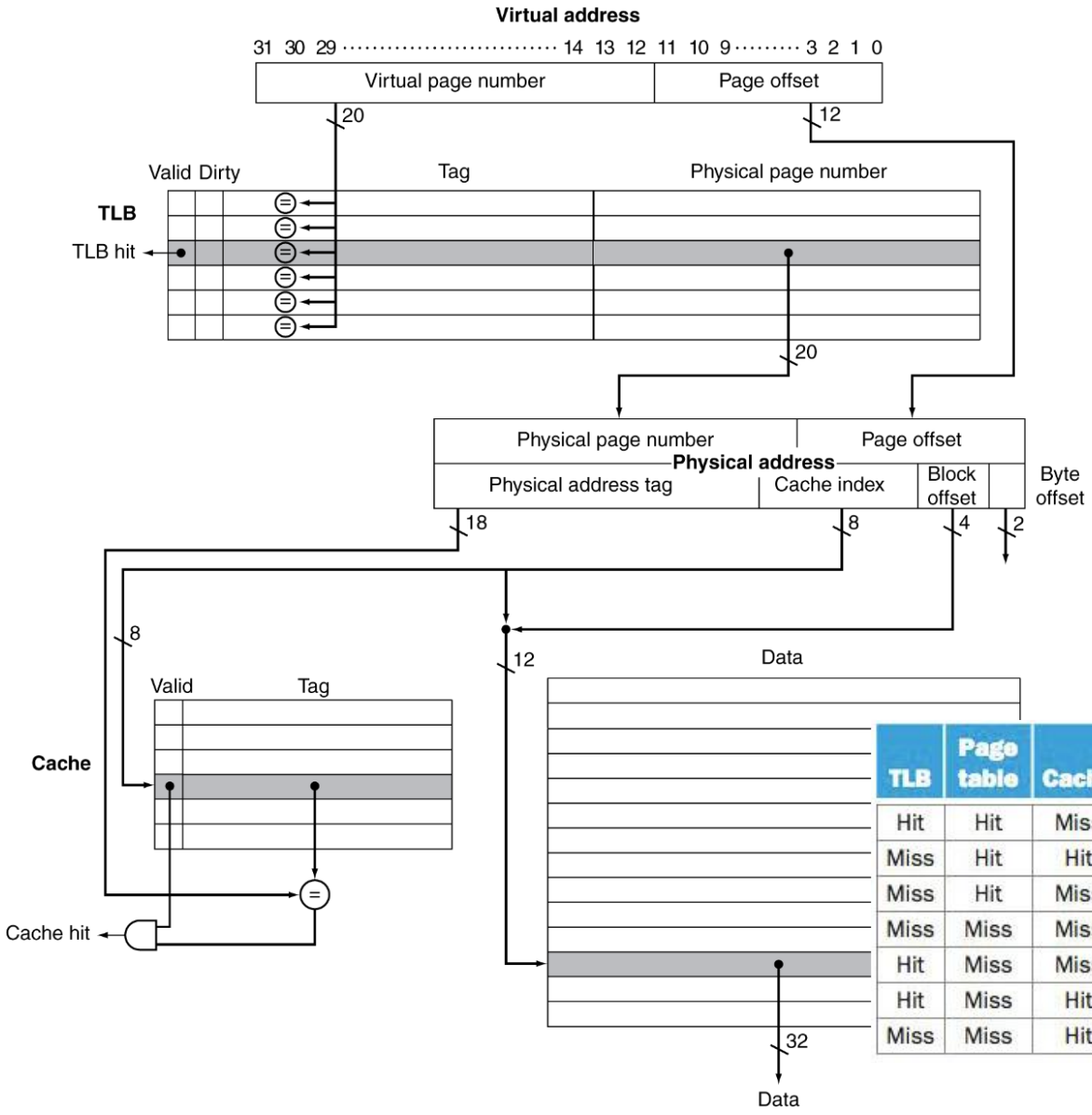
Fast Translation Using a TLB



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 and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag

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.

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

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