ECSE 425 Lecture 23: Virtual Memory

H&P Appendix C

Last Time

- Basic cache optimizations
  - AMAT = Hit Time + Miss Rate × Miss Penalty
  - Reducing the miss rate
  - Reducing the miss penalty
  - Reducing the time to hit

Today

- Virtual Memory
  - Appendix C.4

Why Virtual Memory?

- Original motivation: increase memory capacity beyond the size of main memory
- The problem:
  - If a program became too large to fit into memory…
- The pre-VM solution:
  - Programmer divides the program into mutually exclusive parts that each fit into main memory
  - Programmer ensures the required data are loaded at the appropriate time

Virtual Memory Basics

- With "virtual memory," the disk is used as the lowest level in the memory hierarchy
  - Main memory stores a subset of *pages*
  - Pages are moved in and out of memory by the OS
- The address space is usually much larger than the capacity of main memory, or even of the disk
  - E.g. 40-bit addresses => $2^{40} \sim 1 \times 10^{12}$ addresses

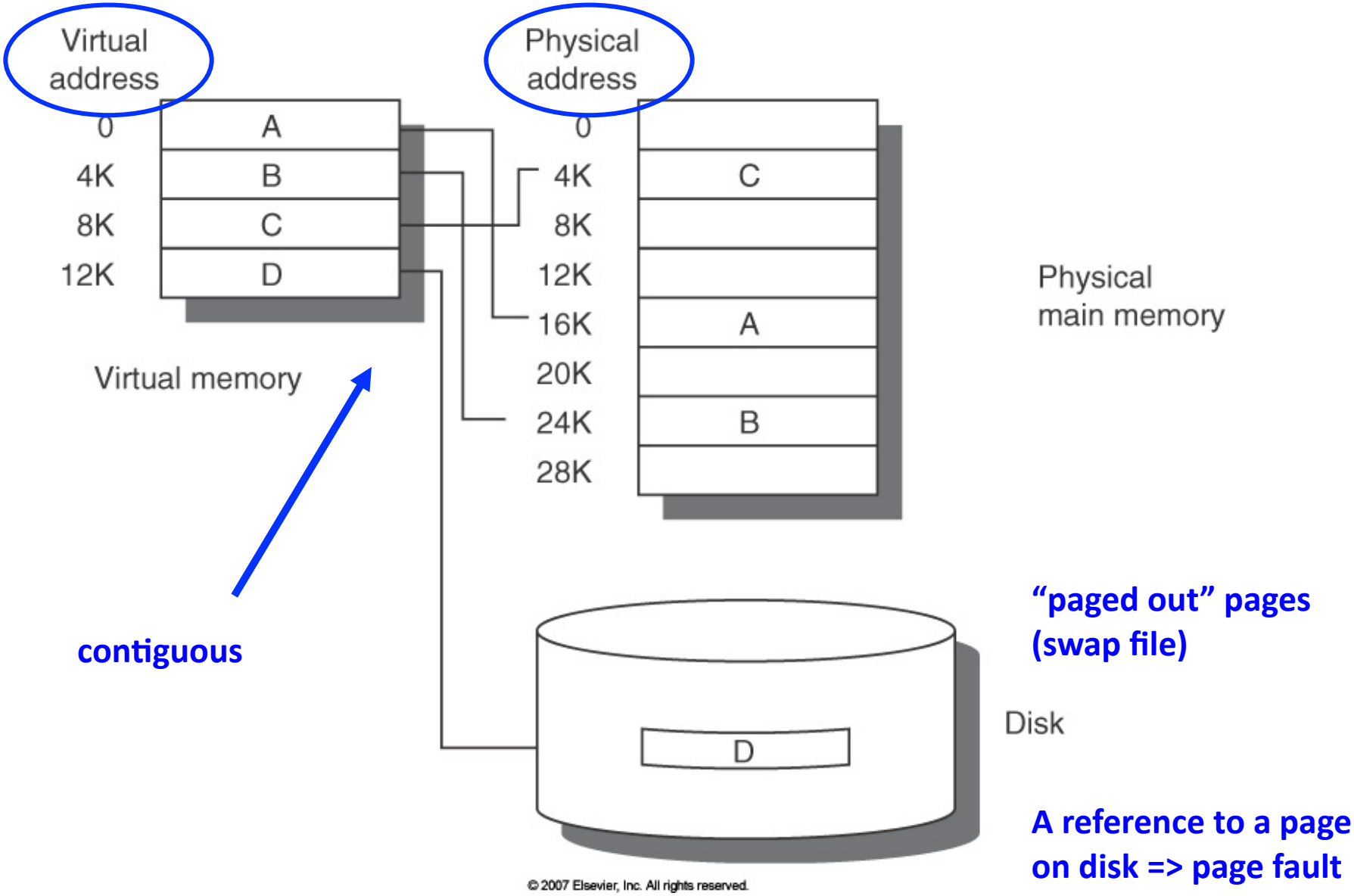
Other Motivations for Virtual Memory

- Multitasking
  - Each thinks it owns the entire memory space
  - But many processes share the memory space
  - Memory protection => processes can't access each others' memory
- Relocation
  - A program can run anywhere in memory
  - Virtual memory maps addresses generated by the compiler to real addresses in memory or on disk

Cache vs. Virtual Memory

Parameter	First-level cache	Virtual memory
Block (page) size	16-128 bytes	4096-65,536 bytes
Hit time	1-3 clock cycles	100-200 clock cycles
Miss penalty	8-200 clock cycles	1M – 1M cycles
(access time)	(6-160 clock cycles)	(800K-8M clock cycles)
(transfer time)	(2-40 clock cycles)	(200K-2M clock cycles)
Miss rate	0.1-10%	0.00001-0.001%
Address mapping	25-45 bit physical address to 14-20 bit cache address	32-64 bit virtual addr to 25-45 bit physical addr

Mapping Virtual to Physical Addresses



Pages and Segments

- Chunks of memory are called pages or segments

	Page	Segment
Size	Fixed	Variable
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial	Hard
Memory use inefficiency	Internal fragmentation	External fragmentation
Efficient disk traffic	Yes	Not always

- Hybrid approaches
  - Paged segments: segments with *n* pages
  - Multiple page sizes: e.g., from 1 KB to 4 MB

Four Memory Hierarchy Questions, rev. 2

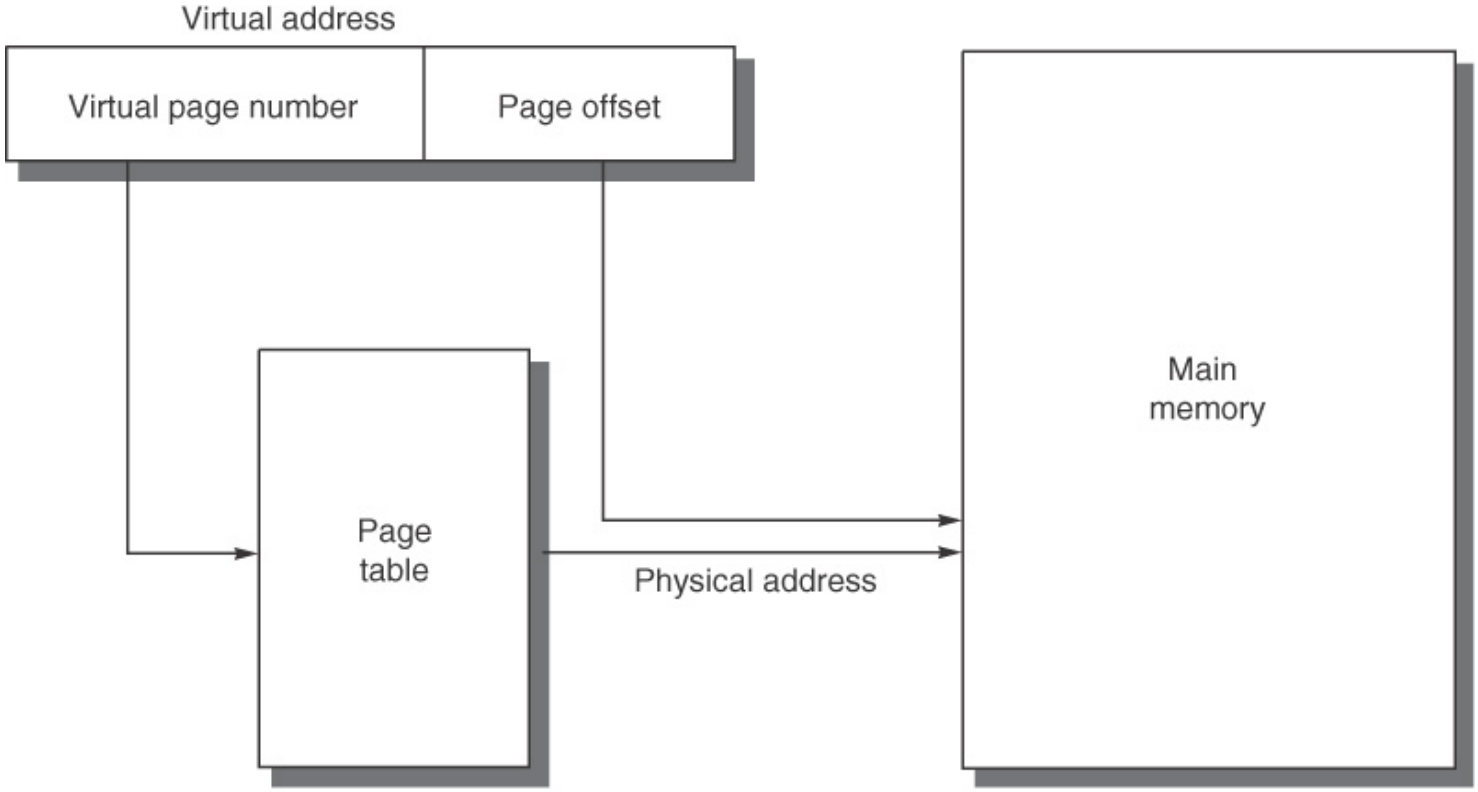
- Block placement
  - Miss penalty is huge! (1M to 10M cycles)
  - Target lower miss rates at expense of implementing more complex algorithms
  - Virtual memory is fully associative strategy
- Block identification
  - Translate from virtual addresses to physical addresses
  - The mapping is stored in the *page table*
  - The page table is stored in main memory
  - Optimize address translation for the common case

Four Questions, rev. 2, Cont'd

- Block replacement
  - Minimize page faults!
  - LRU replacement
    - Set "use bit" when a page is accessed
    - O/S periodically clears them
    - On replacement, find a page without the "use bit" set
- Write strategy
  - Avoid writing to disk whenever possible
  - Always use write-back (write-through is too slow)
  - Set the "dirty bit" when a page is modified

Address Translation

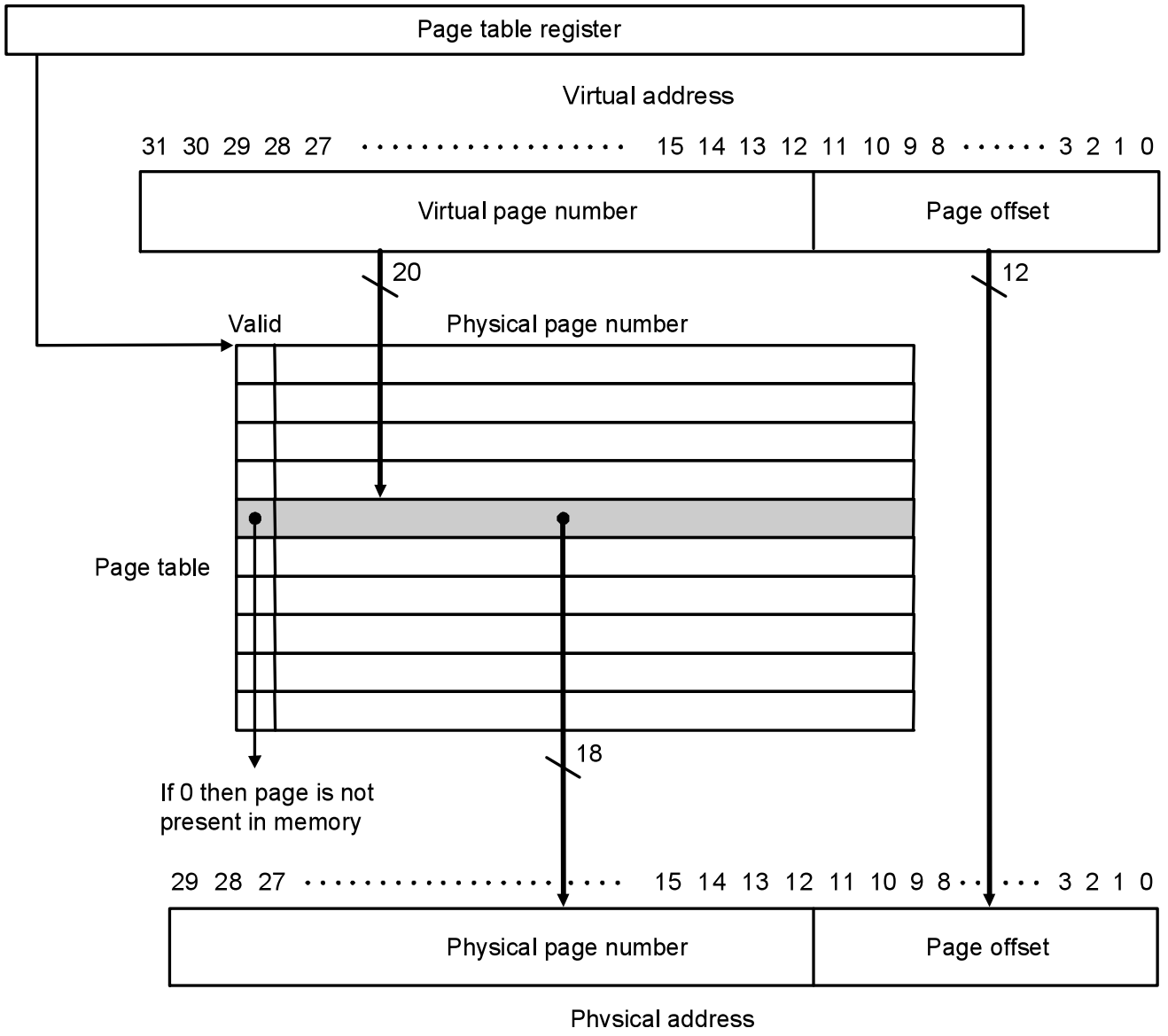
- Virtual addresses map to physical addresses
  - Page offset is concatenated to the page PA
  - Segment offset is added to the segment PA



Page Tables

- Page tables store the translation from VA to PA
  - Indexed by the virtual page #
- Page tables are stored in main memory
  - And can therefore be in the cache
- Each process gets a page table
- Page tables can be large
  - E.g.: 32-bit VA, 4KB pages, 4 bytes per PTE
  - Resulting page table is $2^{32}/2^{12} \times 2^2 = 2^{22}$ or 4MB

Page Table for 32-bit VA, 512 MB of RAM

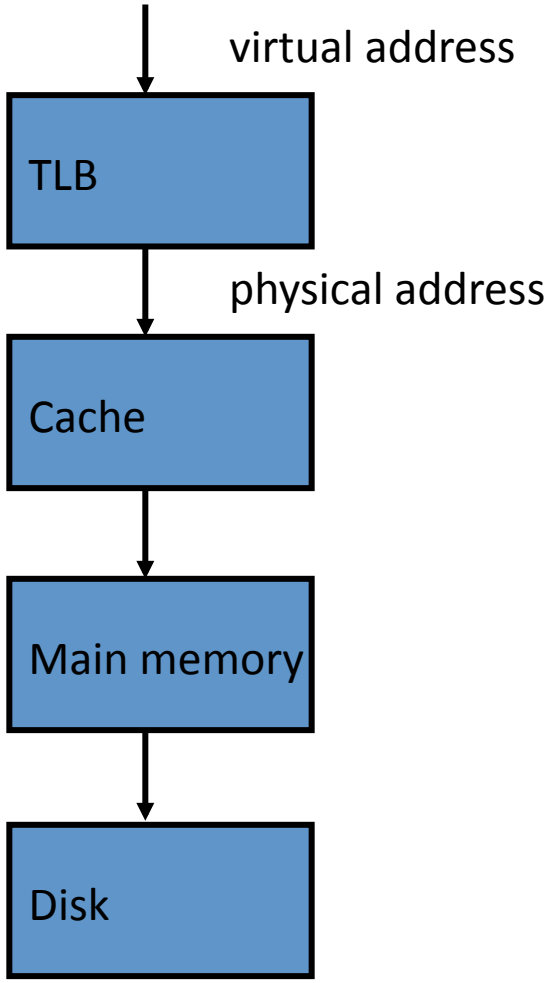


Fast Address Translation

- Paging: every LD requires two memory accesses
  - One to read the page table to get the PA
  - One to get the data
  - Expensive!
- Principle of locality:
  - Consecutive accesses are likely to the same page
  - Why recalculate the address translation every time?
- Solution => cache recent translations
  - Use a translation look-aside buffer (TLB)

Translation Lookaside Buffers (TLB)

- Just like a cache
  - Tag: a portion of the virtual address
  - Data: physical page number
  - Other utility fields: protection, valid, use, and dirty
- TLB sits "between" the CPU, L1
  - On the critical path
  - TLB speed influences cycle time
- Split L1? Two TLBs



Hit Time Reduction

- Avoiding address translation while indexing cache reduces hit time
  - Take the TLB off the critical path
- Caches can be tagged and index using virtual or physical addresses
- Hits are much more common than misses, so
  - Use virtual addresses for caches
  - Save on address translation time

Why Not Virtually Indexed Caches?

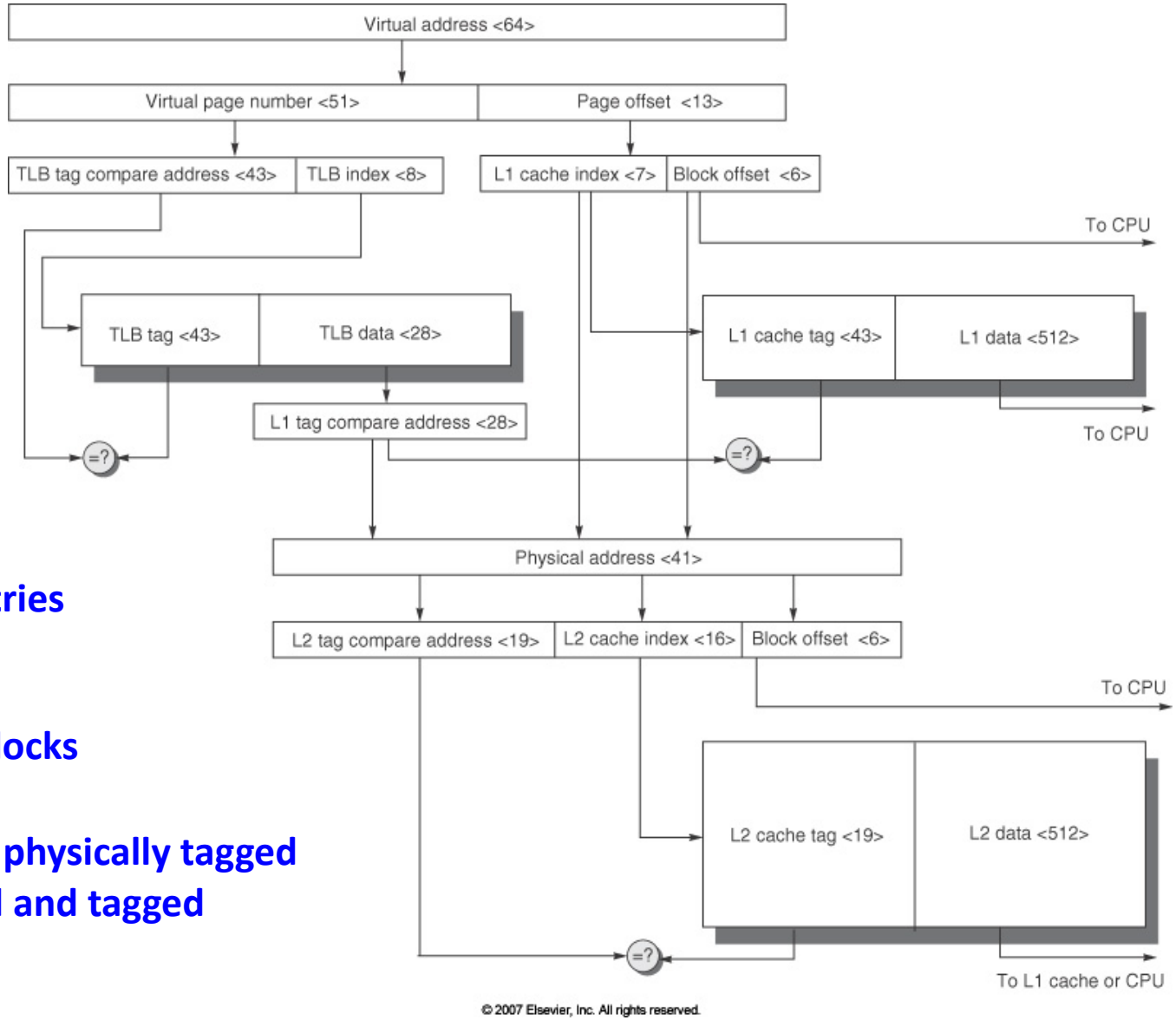
- Protection must be enforced for every cache access
  - This is usually done in the TLB
  - Copy protection bits into the cache on TLB miss
- Cache must be flushed on every process switch
  - Because different processes have different mappings of physical addresses to virtual addresses
- Synonyms or aliases (two different virtual addresses for the same physical address)
  - If software modifies one in cache, the other is invalid
- I/O uses physical addresses

Virtually Indexed, Physically Tagged

- Index into cache using part of the page offset
- While indexing, perform translation to match the tag with the physical address
- Limits the size of direct-mapped caches
  - Low-order bits must be equal in both VA and PA
  - Cache must be no bigger than the page size
  - Adding associativity allows caches to be larger (IBM uses a 16-way cache)

$$2^{\text{Index}} = \text{Cache Size} / (\text{Block Size} \times \text{Associativity})$$

Example Memory Hierarchy



- **Page size: 8 KB**
- **TLB: DM with 256 entries**
- **L1: DM, 8 KB**
- **L2: DM, 4 MB**
- **Caches use 64-byte blocks**

- **L1: Virtually indexed, physically tagged**
- **L2: Physically indexed and tagged**

Summary

- Virtual Memory Basics
  - Automatic management of memory-disk swap
  - Memory protection
  - Program relocation
- Pages are swapped in and out of memory
- Page tables map virtual addresses to physical addresses
- Making Virtual Memory Fast
  - Translation Look-aside Buffer
  - Virtually indexed, physically tagged caches

Next Time

- Limitations of ILP
  - Read Chapter 3.1-3.3