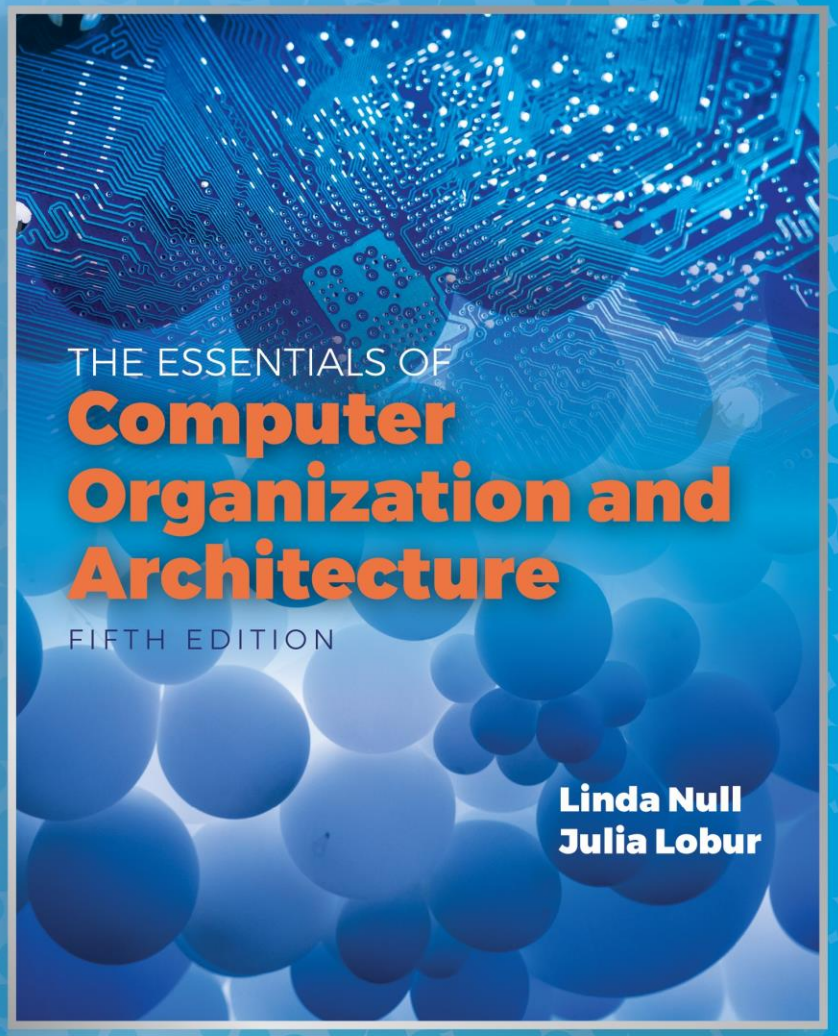
This is the seventh lecture of Chapter 6

Chapter 6

Memory (G)



Quick review of last lecture

- Virtual Memory
  - Paging, Virtual Page, Page Frame, Page Fault
  - Virtual Address, Physical Address
  - Page Table, Valid bit
  - Translate Virtual Address into Physical Address

6.5 Virtual Memory (12 of 26)

- We said earlier that effective access time (EAT) takes all levels of memory into consideration.
- Thus, virtual memory is also a factor in the calculation, and we also have to consider page table access time.
- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk. We have:
  - EAT = 0.99(200ns + 200ns) + 0.01(10ms) = 10,396ns

6.5 Virtual Memory (13 of 26)

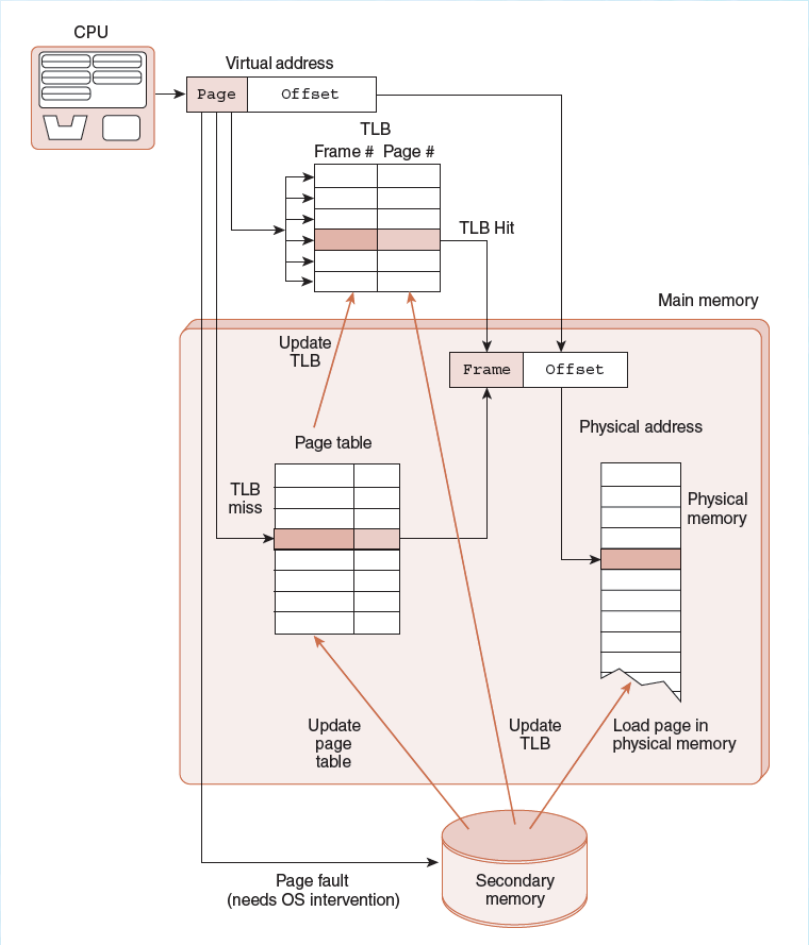
- Even if we had no page faults, the EAT would be 400ns because memory is always read twice: First to access the page table, and second to load the data from memory.
- Because page tables are read constantly, it makes sense to keep them in a special cache called a *translation look-aside buffer* (TLB).
- TLBs are a special associative cache that stores the mapping of virtual pages to physical pages.

The next slide shows address lookup steps when a TLB is involved.

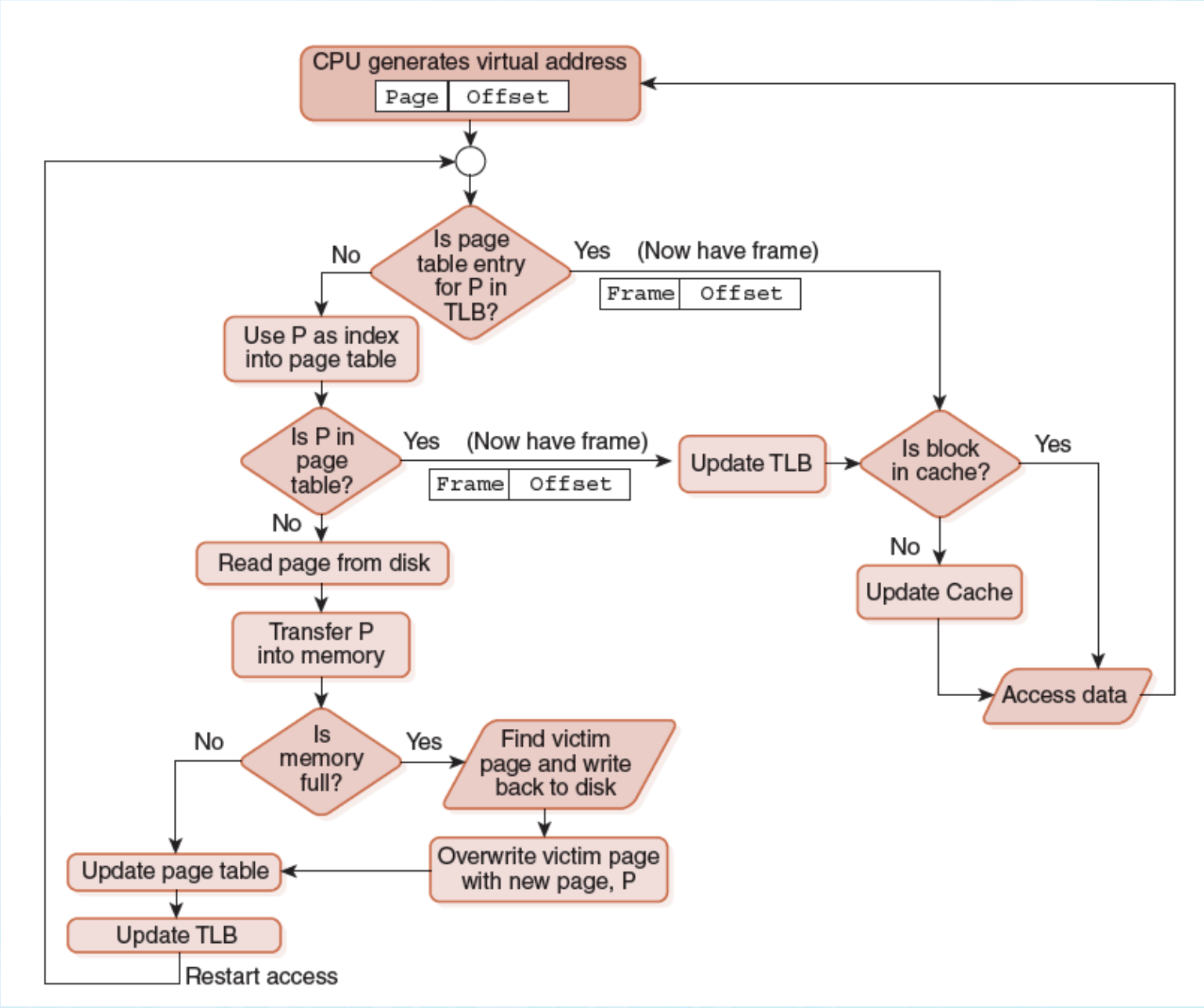
6.5 Virtual Memory (14 of 26)

TLB lookup process

- Extract the page number from the virtual address.
- Extract the offset from the virtual address.
- Search for the virtual page number in the TLB.
- If the (virtual page #, page frame #) pair is found in the TLB, add the offset to the physical frame number and access the memory location.
- If there is a TLB miss, go to the page table to get the necessary frame number. If the page is in memory, use the corresponding frame number and add the offset to yield the physical address.
- If the page is not in main memory, generate a page fault and restart the access when the page fault is complete.



Putting it all together: The TLB, Page Table, and Main Memory



6.5 Virtual Memory (16 of 26)

- Another approach to virtual memory is the use of *segmentation*.
- Instead of dividing memory into equal-sized pages, virtual address space is divided into variable-length segments, often under the control of the programmer.
  - Segmentation can facilitate sharing code or data and protection
- A segment table is used, instead of page table.
  - It contains the segment's memory location and a bounds limit that indicates its size.
- When a new segment is needed, the operating system searches for a location in memory large enough to hold the segment that is retrieved from disk.

6.5 Virtual Memory (17 of 26)

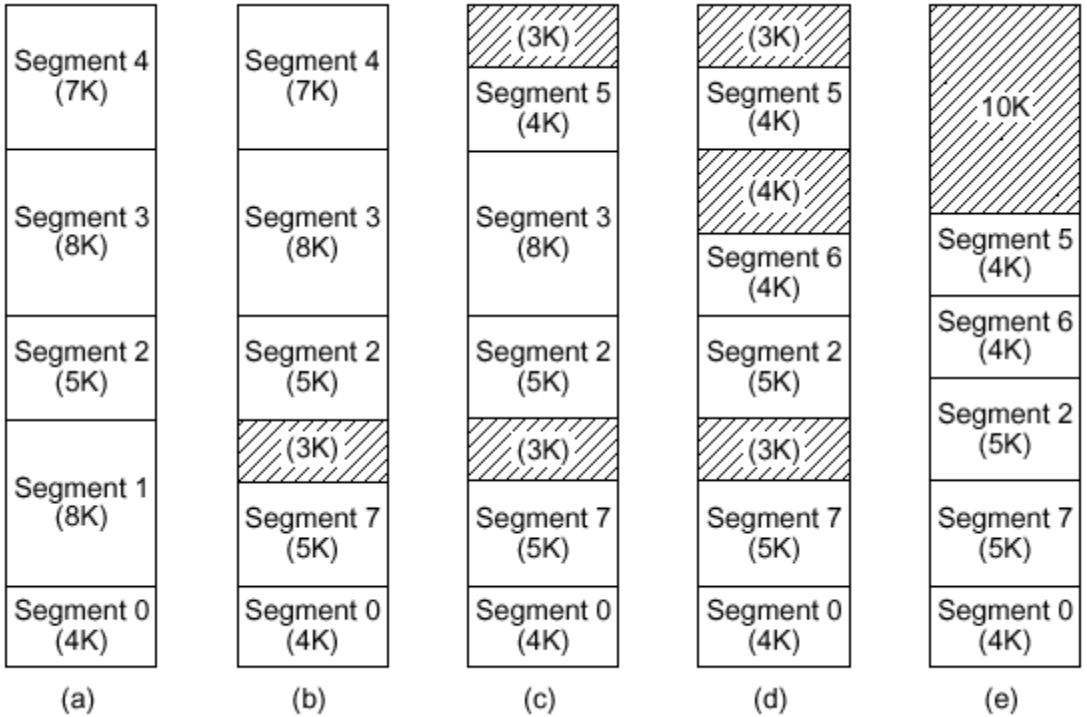
- Both paging and segmentation can cause fragmentation.
- Paging is subject to *internal* fragmentation because a process may not need the entire range of addresses contained within the page. Thus, there may be many pages containing unused fragments of memory.
- Segmentation is subject to *external* fragmentation, which occurs when contiguous chunks of memory become broken up as segments are allocated and deallocated over time.

Fragmentation

External fragmentation
- Unused space in between segments

Internal fragmentation
- Unused space within a page

Development of external fragmentation ((a)-(d))



Removal of external fragmentation by compaction((e))

6.5 Virtual Memory (26 of 26)

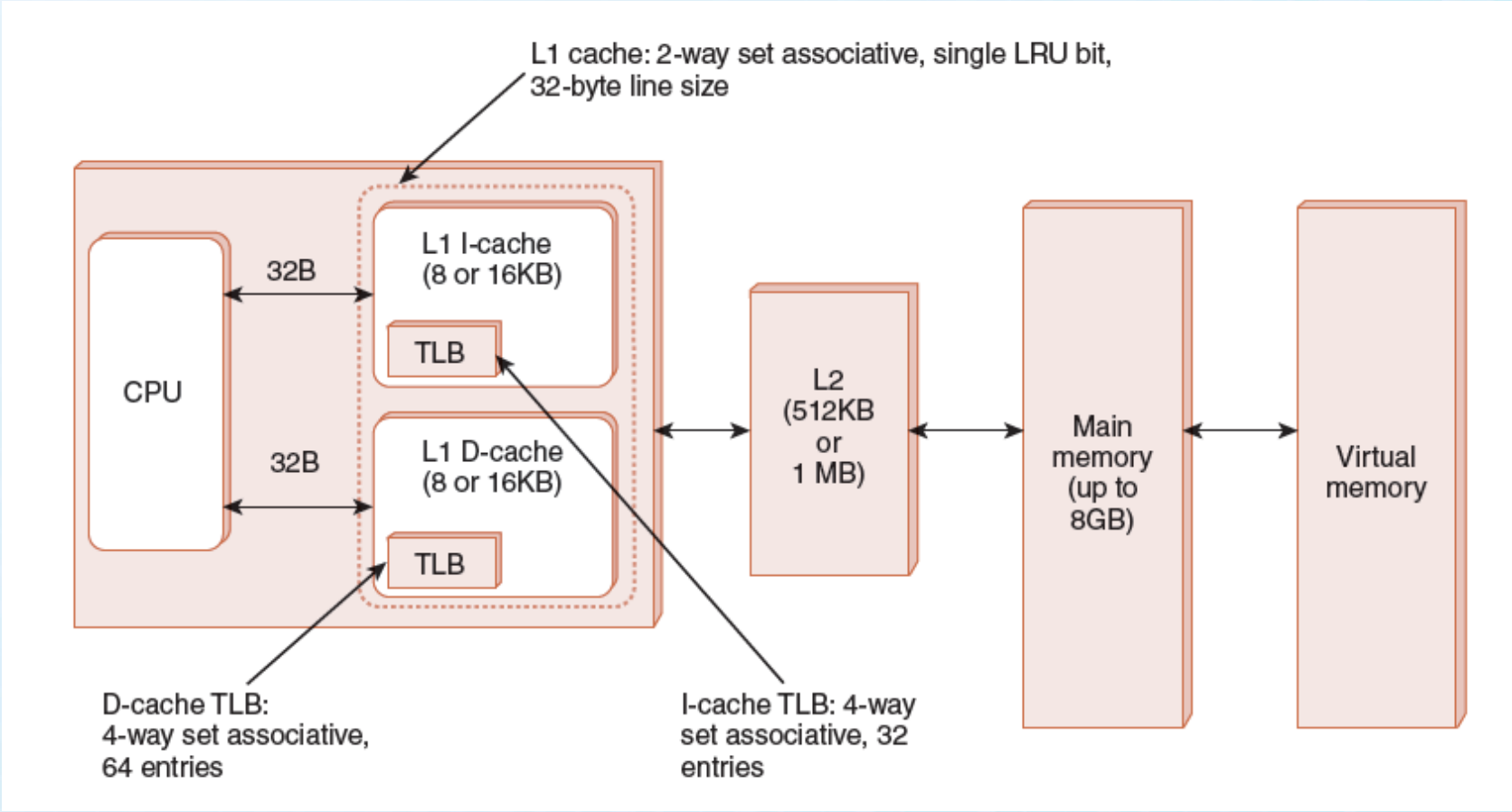
- Large page tables are cumbersome and slow, but with its uniform memory mapping, page operations are fast. Segmentation allows fast access to the segment table, but segment loading is labor-intensive.
- Paging and segmentation can be combined to take advantage of the best features of both by assigning fixed-size pages within variable-sized segments.
- Each segment has a page table. This means that a memory address will have three fields, one for the segment, another for the page, and a third for the offset.

6.6 A Real-World Example (1 of 2)

- The Pentium architecture supports both paging and segmentation, and they can be used in various combinations including unpaged unsegmented, segmented unpaged, and unsegmented paged.
- The processor supports two levels of cache (L1 and L2), both having a block size of 32 bytes.
- The L1 cache is next to the processor, and the L2 cache sits between the processor and memory.
- The L1 cache is in two parts: and instruction cache (I-cache) and a data cache (D-cache).

The next slide shows this organization schematically.

6.6 A Real-World Example (2 of 2)



Conclusion (1 of 2)

- Computer memory is organized in a hierarchy, with the smallest, fastest memory at the top and the largest, slowest memory at the bottom.
- Cache memory gives faster access to main memory, while virtual memory uses disk storage to give the illusion of having a large main memory.
- Cache maps blocks of main memory to blocks of cache memory. Virtual memory maps virtual pages to page frames.
- There are three general types of cache: direct mapped, fully associative, and set associative.

Conclusion (2 of 2)

- With fully associative and set associative cache, as well as with virtual memory, replacement policies must be established.
- Replacement policies include LRU, FIFO, or Random. These policies must also take into account what to do with dirty blocks.
- All virtual memory must deal with fragmentation, internal for paged memory, external for segmented memory.

Review of Essential Terms and Concepts

- Strongly encourage you to study Review of Essential Terms and Concepts.
  - I will not collect and grade them
  - You can discuss the answers to this review questions on our Course Questions Forum on Canvas.