Lecture 9: Virtual Memory

Summary

1. Computer system overview (Chapter 1)
2. Basic of virtual memory; i.e. segmentation and paging (Chapter 7 and part of 8)
3. Process (Chapter 3)
4. Mutual Exclusion and Synchronization (Chapter 5 section 1-4)
   - Conditions for race avoidance.
   - Strict alternation.
   - Semaphores.
   - Producers and Consumers problem.
   - Hardware support for mutual exclusion.
   - Monitors.
5. Threading (user mode and kernel mode).
6. Deadlock and Starvation.
7. Memory management.

Virtual Memory

1. Lessons learned from memory management.
   (a) All memory references are logical addresses that are dynamically translated into physical address at run time. Thus, a process can be swapped in and out of memory and occupy different regions of main memory during the course of execution.
   (b) A process may be broken up into pieces and these pieces need not be contiguously located in main memory during execution.
2. Break through: Based on the two preceding characteristics, it is not necessary that all pages or segments be presented in the main memory during execution. Only the pieces that hold the next instructions and necessary data need to be in the **resident set**.
3. If the issued logical address is not in main memory:
   - Generate an interrupt to indicate that the desired address is not in main memory (access fault).
   - The OS puts the interrupted process in a blocking state and takes control.
   - The OS issues a disk I/O request to bring the needed piece to main memory.
   - The OS dispatches another process to run.
   - Once the desired piece has been brought into main memory, an I/O interrupt occurs.
   - The OS takes control and put the blocked process into a ready state.
4. Implications:

- More processes can be maintained in main memory.
- The maximum process image (virtual memory) may be larger than all of main memory (real memory).

5. Does virtual memory work?
   - Small working set.
   - If the OS frequently throws out the pieces about to be used, then **thrashing** can occur.
   - Principle of locality: programs and data references within a process tend to cluster.
6. Paging: page table revisited.
   - From last lecture, a process' page table contains information about the mapping of page to page frame.
     - It is per process and loaded into the memory when all the process' pages are loaded.
     - To support VM, additional bits are needed: P (present bit) and M (modified bit). (See Figure 8.2)
     - Because page tables can be large, they are also subject to paging. Thus, the goal is to break a page table into smaller chunks, and each chunk is equal to the page size (e.g. 4KB).
   - Multilevel page table (see Figure 8.4 and 8.5).
     (a) Load root page table (aka. page table directory). This table is the first that get loaded and stays in the memory throughout the execution of that process.
     (b) User the first n bit (10 in the example) to index the root page table for the secondary page table.
     (c) Index into the secondary page table and find the page frame.
       - If the P bit is set, the page is in the main memory, done.
       - If the P bit is clear, the page must be brought into the main memory.
         * Identify the location in the main memory to place the page. If eviction is needed, check the M bit of the evicted page to determine if writing to disk is needed before replacement. Note that page replacement policy will be discussed a little later.
         * Bring in the new page to this location then done.
   - Inverted page table.
     - An entry for every page frame. Thus, the table size is proportional to the amount of main memory and not virtual memory.
     - Entire table stay in the memory.
   - Translation Lookaside Buffer (TLB) contains the page table entries that have be most recently used (see Figure 8.7 and Figure 8.8).
7. Replacement policy:
   - Optimal.
   - Least Recently Used (LRU).
   - First-in-first-out (FIFO).
   - Clock (see Figure 8.16).

8. Other considerations.
    - Resident set size.
    - Page size.
    - TLB size.
9. Segmentation (see Figure 8.13).

Scheduling Algorithms

1. Decision mode: preemptive or non-preemptive.
2. Policies (see Figure 9.5):
    - First-come-first-serve (FCFS): Schedule processes based on the order of arrival.
    - Round Robin: Use time slice to schedule processes that are ready to run.
    - Shortest Process Next (SPN). The process with the shortest expected running time is selected next. It is non-preemptive.
    - Shortest Remaining Time (SRT). It is a preemptive version of SPN. The process with the shortest remaining time is picked next.
    - Highest Response Ratio Next (HRRN): $R = \frac{w+s}{s}$, w = wait time, s = expected service time. The process with the greatest R is picked next.
    - Feedback. Combine preemptive scheduling based on time-slice with dynamic priority. Process loses one level of priority after each run. The lowest queue uses round-robin while others use FCFS.

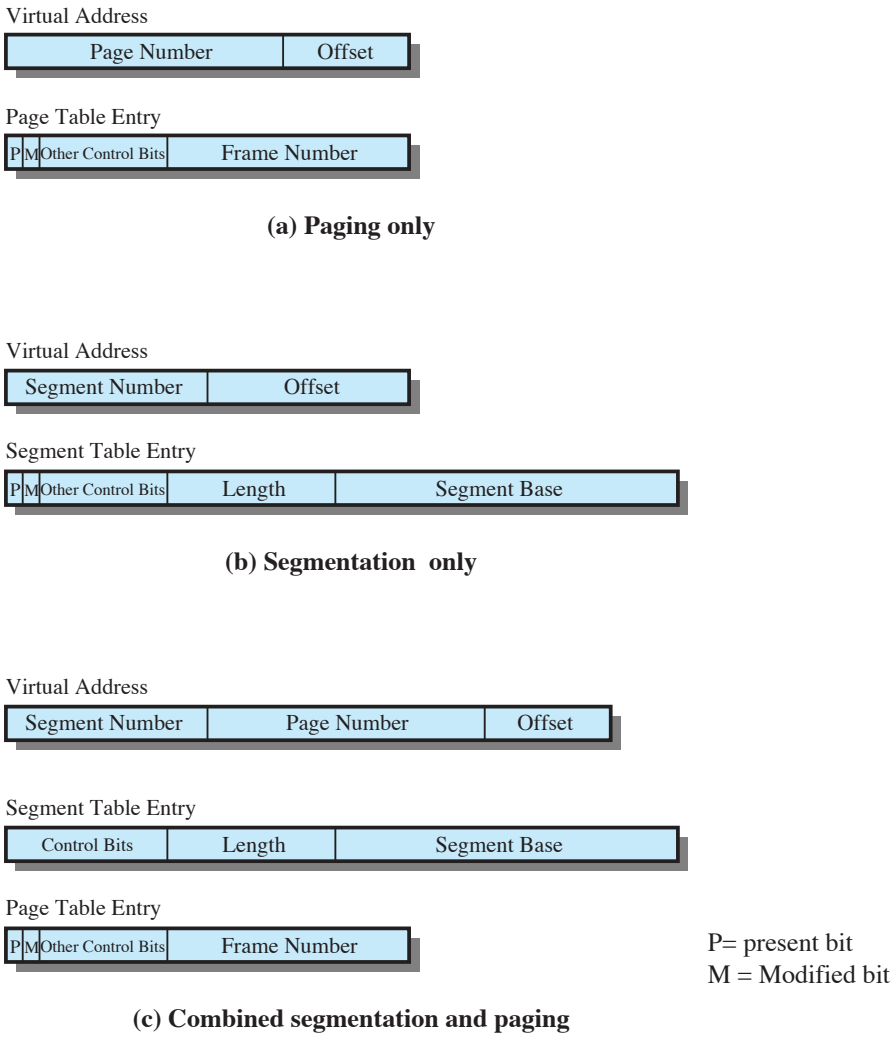


Figure 8.2 Typical Memory Management Formats

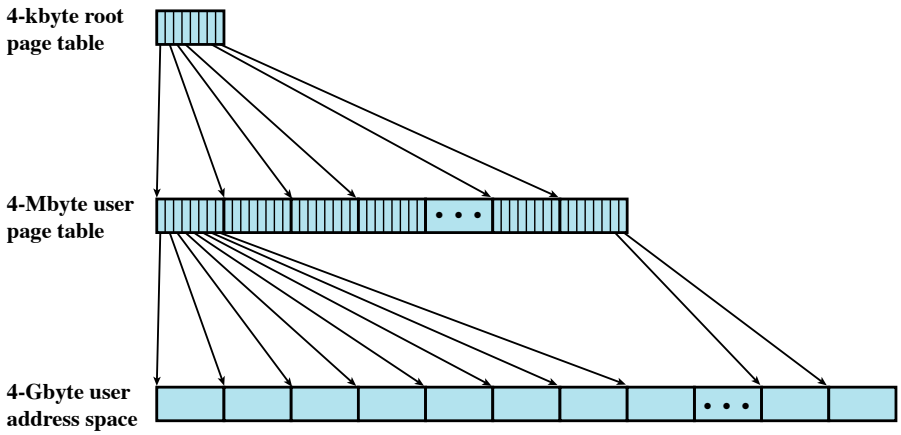


Figure 8.4 A Two-Level Hierarchical Page Table

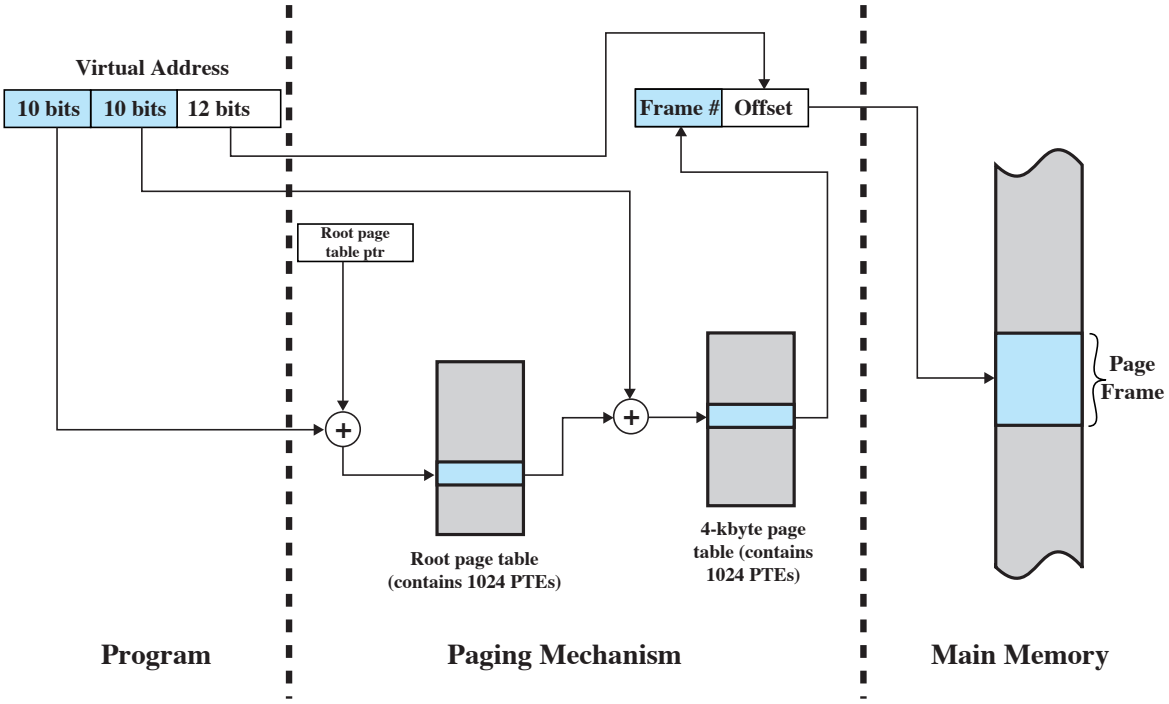


Figure 8.5 Address Translation in a Two-Level Paging System

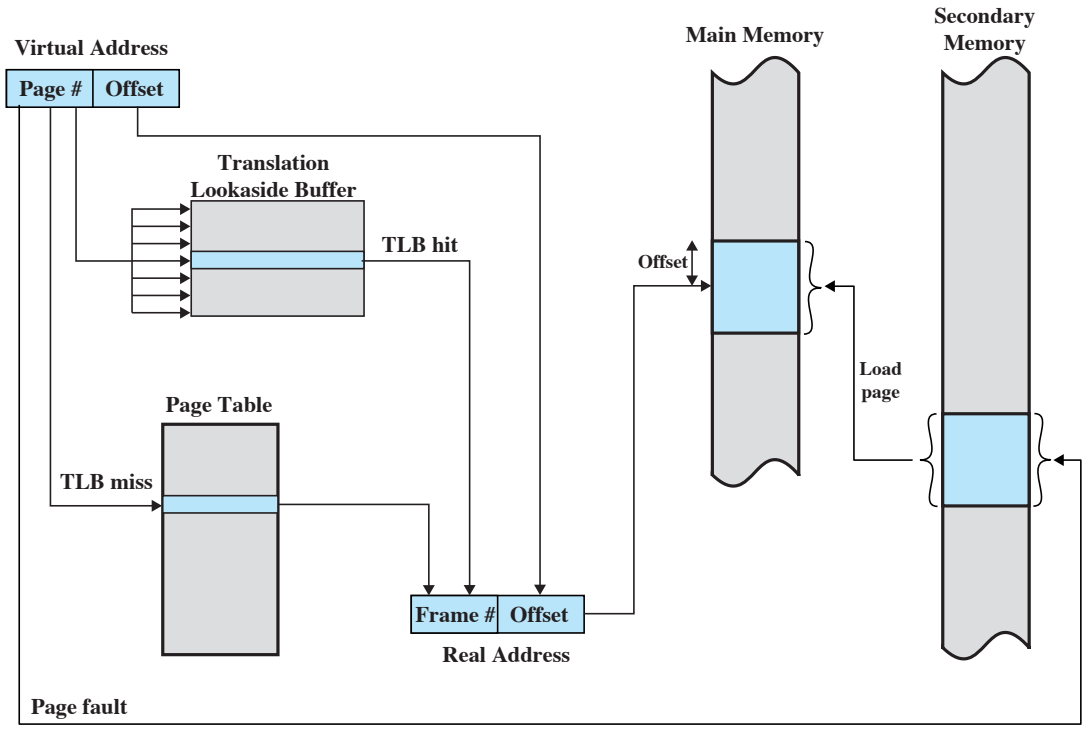


Figure 8.7 Use of a Translation Lookaside Buffer

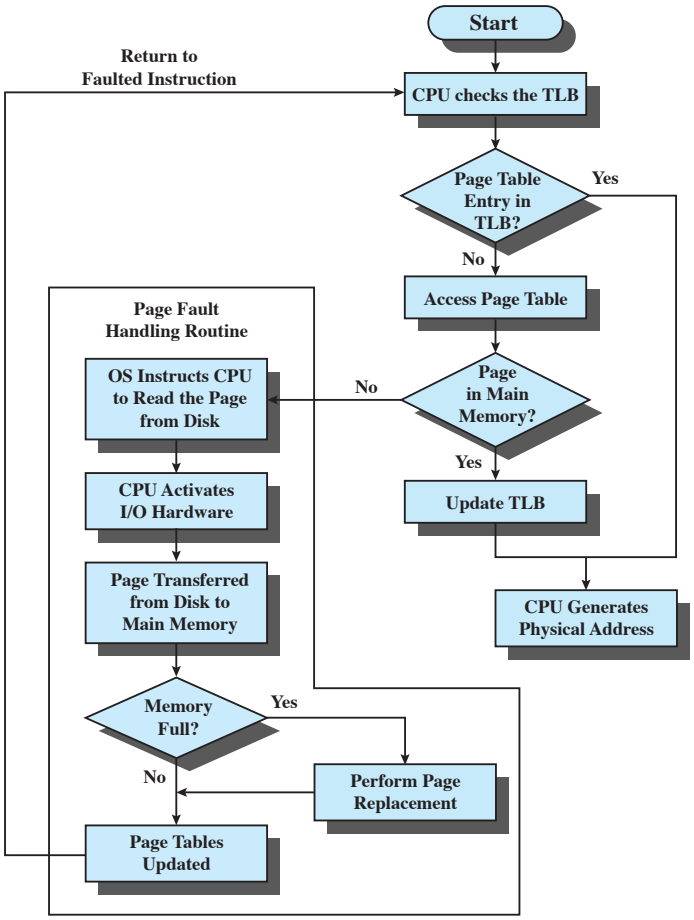


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

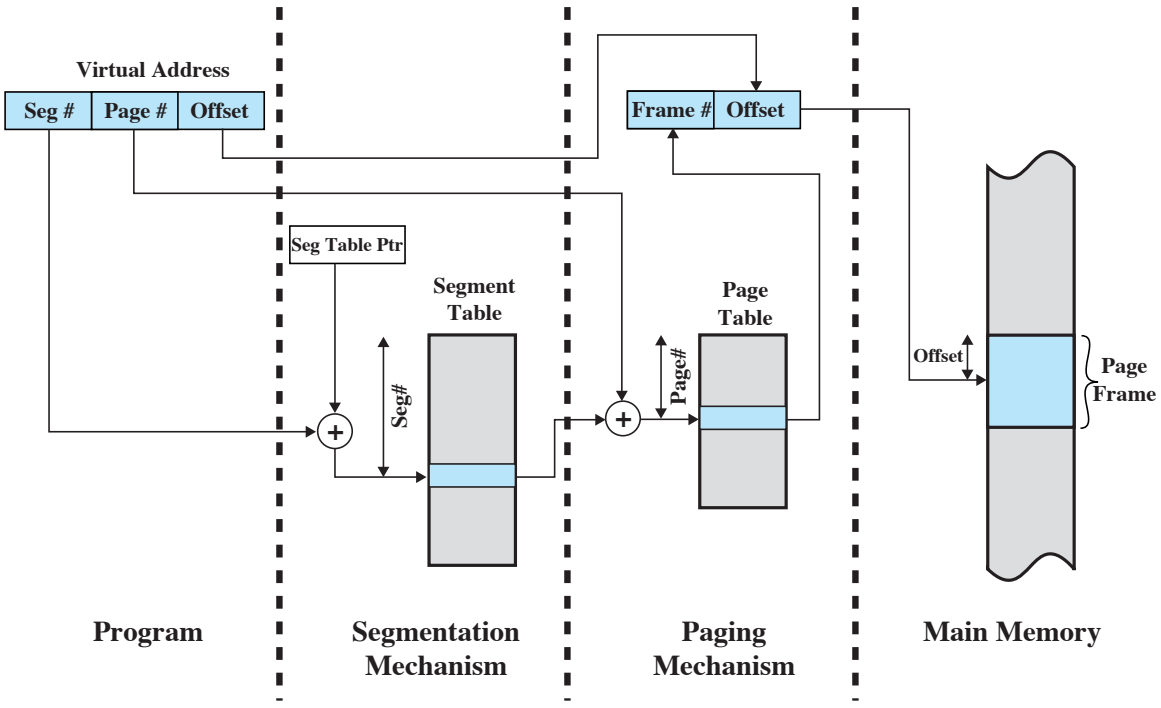
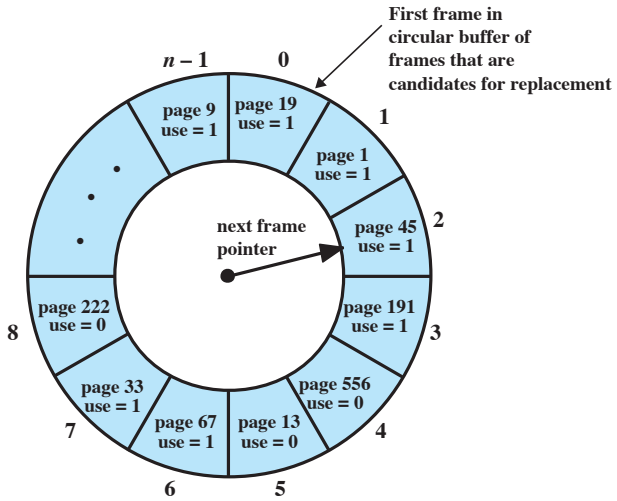
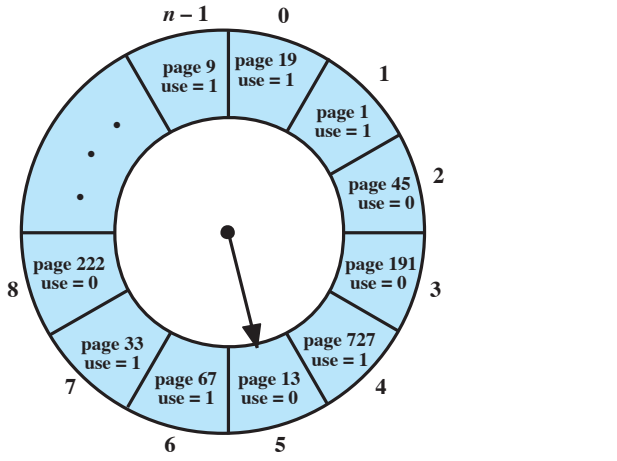


Figure 8.13 Address Translation in a Segmentation/Paging System



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

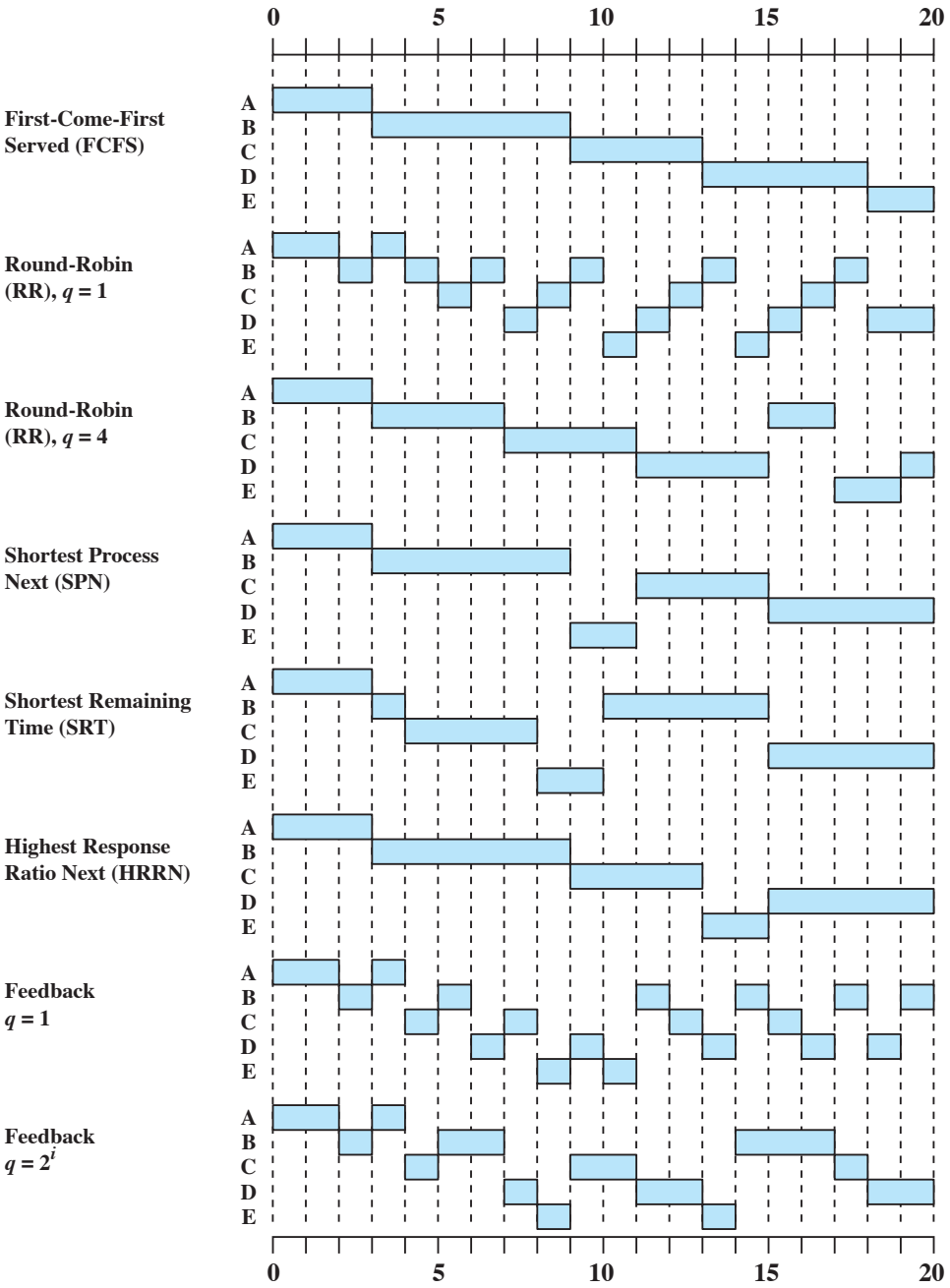


Figure 9.5 A Comparison of Scheduling Policies