the essentials of

Computer Organization and Architecture

Linda Null and Julia Lobur

Chapter 8

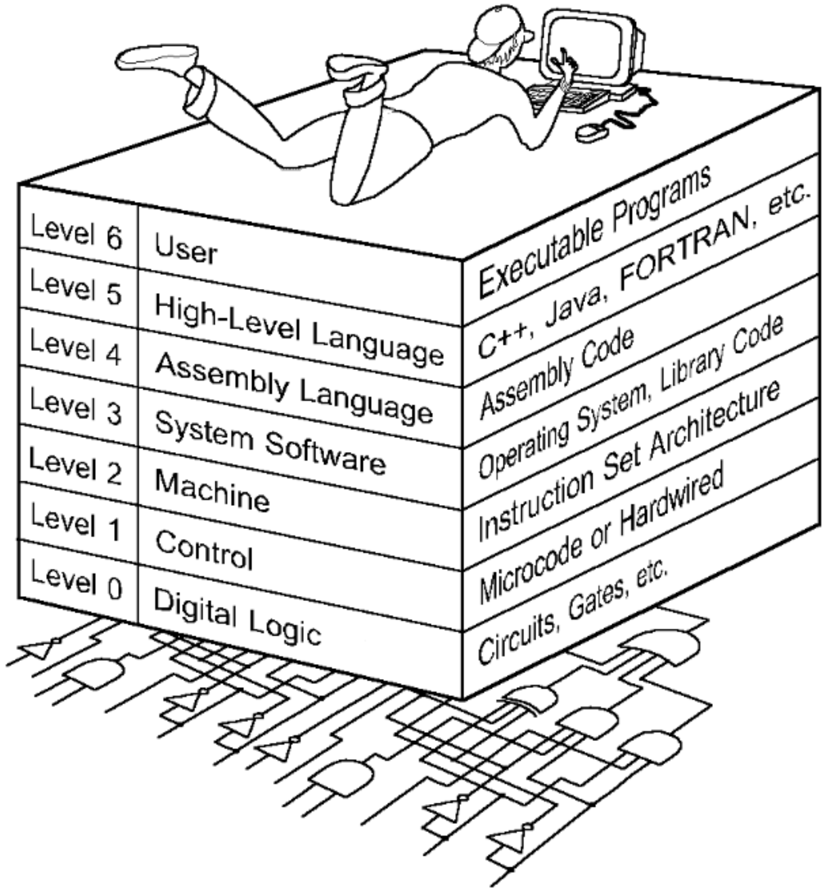
System Software

Chapter 8 Objectives

- Become familiar with the functions provided by operating systems, programming tools, database software and transaction managers.
- Understand the role played by each software component in maintaining the integrity of a computer system and its data.

8.1 Introduction

- Each virtual machine layer is an abstraction of the level below it.
- The machines at each level execute their own particular instructions, **calling upon** machines at lower levels to perform tasks as required.
- Computer circuits ultimately carry out the work.



Level 6 User Executable Programs
Level 5 High-Level Language C++, Java, FORTRAN, etc.
Level 4 Assembly Language Assembly Code
Level 3 System Software Operating System, Library Code
Level 2 Machine Instruction Set Architecture
Level 1 Control Microcode or Hardwired
Level 0 Digital Logic Circuits, Gates, etc.

8.1 Introduction

- The biggest and fastest computer in the world is of no use if it cannot efficiently provide beneficial services to its users.
- Users see the computer through their application programs. These programs are ultimately executed by computer hardware.
- System software-- in the form of operating systems and middleware-- is the glue that holds everything together.

8.2 Operating Systems

- The main role: to help various applications interact with the computer hardware
- Provides a necessary set of functions allowing software packages to control the computer’s hardware
- Without an operating system, each program you run would need its own driver for the video card, sound card, hard drive,…

8.2 Operating Systems

- From the user’s perspective, they expect easy managing the system and its system, ‘drag and drop’ file management, ‘plug and play’ device management
- From the programmer’s perspective, OS hides the details of the system’s lower architectural levels.
- It is difficult to program at the machine level or the assembly language level

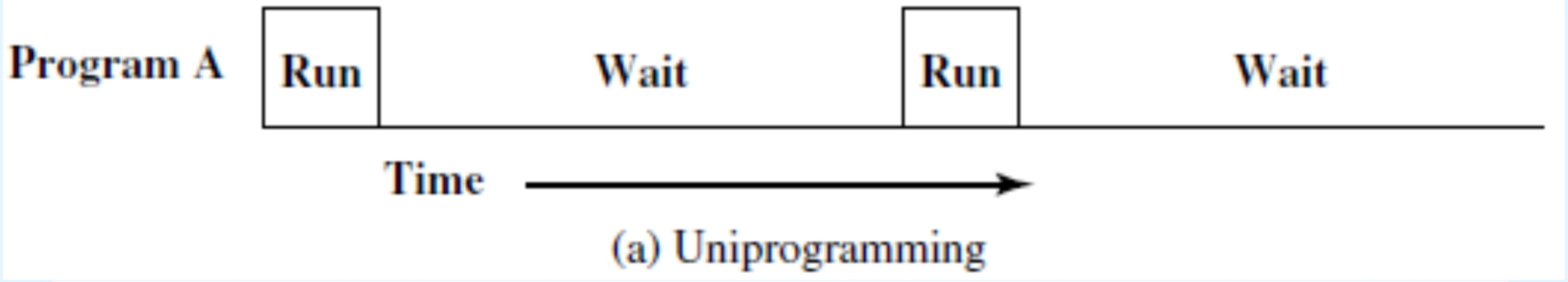
8.2 Operating Systems

- The evolution of operating systems has paralleled the evolution of computer hardware.
  - First generation computers were quite slow, there was no need for an OS. Human operators performed task management.
  - Second generation were built with transistors, increasing in speed and CPU capacity, batch processing was introduced to keep CPU busy. Monitors helped with the processing, providing minimal protection and handling interrupts.
  - The third generation was marked by use of IC. Time sharing was introduced

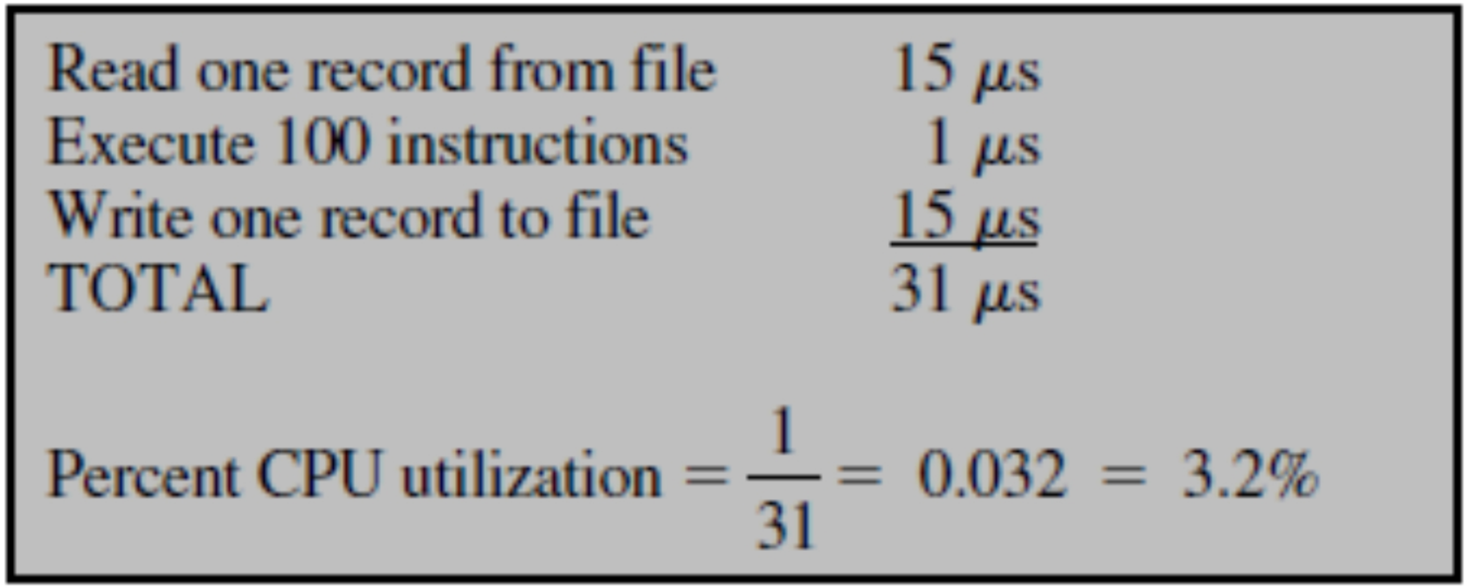
8.2 Operating Systems

- In the 1960s, hardware has become powerful enough to accommodate *multiprogramming(多道程序)*, the concurrent execution of more than one task.
- Multiprogramming is achieved by allocating each process a given portion of CPU time (a *timeslice*).

8.2 Operating Systems



(a) Uniprogramming

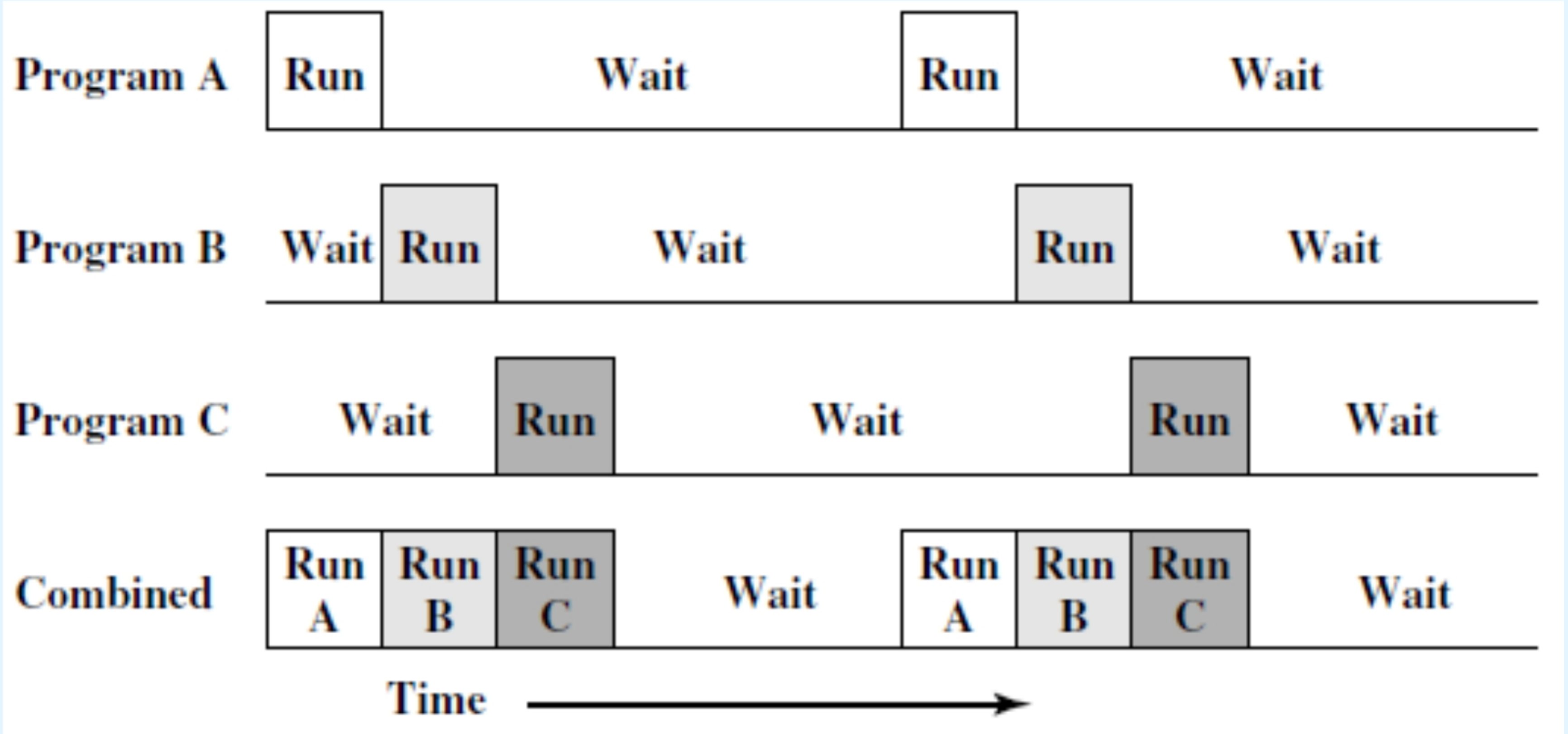


Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 8.4 System Utilization Example

8.2 Operating Systems



(c) Multiprogramming with three programs

8.2 Operating Systems

- When a process is taken from the CPU and replaced by another, we say that a *context switch* has occurred
- During a context switch, all pertinent information about the currently executing process must be saved.
- When the process is scheduled to use CPU again, it can be restored to the exact state in which it was interrupted
  – Page tables, CPU registers, stacks, etc.

8.2 Operating Systems

- Today, multiprocessor systems have become commonplace.
  - They present an array of challenges to the operating system designer, including the manner in which the processors will be synchronized, and how to keep their activities from interfering with each other.
- *Tightly coupled* multiprocessor systems share a common memory and the same set of I/O devices
  - All processors perform the same functions, with the processing load being distributed among all of them

8.2 Operating Systems

- Loosely coupled multiprocessor systems have physically separate memory.
  – These are often called *distributed systems*.
  – Another type of distributed system is a networked system, which consists of a collection of interconnected, collaborating workstations.
- *Real time* operating systems control computers that respond to their environment.
  – *Hard real time* systems have tight timing constraints, *soft real time* systems do not.

8.2 Operating Systems

- Personal computer operating systems are designed for ease of use rather than high performance.
  - One main objective: make the system user friendly
- First personal computer(intel 8080) OS, CP/M(control program for Microprocessor), is based on BIOS(basic input-output operating system)
  - BIOS allowed CP/M to be exported to different types of PCs easily because it provided necessary interactions with input and output devices.
  - OS could remain the same for various machines, only BIOS had to be altered

8.2 Operating Systems

- OS for early personal computers operated on commands typed form the keyboard(QDOS, MS-DOS, PC DOS)
- Invention of GUI, and invention of the mouse changed the face of operating system
- Command prompts were replaced by windows, icons, and drop-down menus.
- Windows 1.x, 2.x,95,98,me, xp, MacOS(Macintosh operating system, preceded the windows GUI by several years)

8.2 Operating Systems

- At one time, these systems were considered appropriate only for desktop publishing and games. Today they are seen as technology enablers for users with little formal computer education.

8.2 Operating Systems

- An operating system differs from most other software: it is event driven
  - Performs tasks in response to command, application programs, I/O devices, and interrupts
- Four main factors drive operating system design: performance, power, cost and compatibility.

8.2 Operating Systems

- Two operating system components are crucial: The *kernel* and the system programs.
- As the core of the operating system, the kernel performs **scheduling**, **synchronization**, **memory management**, **interrupt handling** and it provides **security and protection**.
- *Microkernel* systems provide minimal functionality, with most services carried out by external programs.
- *Monolithic* systems(单内核) provide most of their services within a single operating system program.

8.2 Operating Systems

- Microkernel systems provide better security, easier maintenance, and portability at the expense of execution speed.
  - Services running at the user level, restricted access to system resources.
  - Permits many services to be restarted or reconfigured without restarting the entire operating system
  - Services runs above the kernel, instead of in the kernel itself.
  - Additional communication between the kernel and the other modules is necessary
  - Examples are Windows 2000, and QNX.

8.2 Operating Systems

- Monolithic systems give faster execution speed, but are difficult to port from one architecture to another.
  - Provide all of their essential functionality through a single process
  - They are significantly larger than microkernels.
  - It interacts directly with the hardware, not easily portable
  - Examples are Linux, MacOS, and DOS.

8.2 Operating Systems

- Operating system services
  - Human interface
  - Process management
  - Security and Protection
  - Resource management
  - Interaction with I/O devices

8.2 Operating Systems

- The Human interface
  - Hardware developers: OS as an interface to the hardware
  - Applications developers: OS as an interface to various application programs and services
  - Ordinary users: most interested in graphical interface
  - Command line interface and graphical user interface
  - User interface is a program, or small set of programs

8.2 Operating Systems

- Process management lies at the heart of operating system services.
  - The operating system **creates** processes, **schedules** their access to resources, **deletes** processes, and **deallocates** resources that were allocated during process execution.
- OS keeps track of each process, its actual state-running, ready or waiting, the resources it is using, and those it requires
- The operating system monitors the activities of each process to avoid **synchronization** problems that can occur when processes use shared resources.
- If processes need to communicate with one another, the operating system provides the services.

8.2 Operating Systems

- The operating system schedules process execution.
- First, the operating system determines which process shall be granted access to the CPU.
  - This is *long-term scheduling*.
- After a number of processes have been admitted, the operating system determines which one will have access to the CPU at any particular moment.
  - This is *short-term scheduling*.
  - **OS maintains a list of ready processes**.
  - Waiting list
  - Process priority
  - PCB(Process Control Block)

8.2 Operating Systems

- Context switches occur when a process is taken from the CPU and replaced by another process.
  - Information relating to the state of the process is preserved during a context switch.
  - All CPU registers, page tables, others associated with virtual memory

8.2 Operating Systems

- Short-term scheduling can be **nonpreemptive** or preemptive.
- In nonpreemptive scheduling, a process has use of the CPU until either it terminates, or must wait for resources that are temporarily unavailable.
- In preemptive scheduling, each process is allocated a time slice. When the time slice expires, a context switch occurs.
- Or higher priority process will preempt lower priority process.

8.2 Operating Systems

- Four approaches to CPU scheduling are:
  - **First-come, first-served** where jobs are serviced in arrival sequence and run to completion if they have all of the resources they need. Nonpreemptive, easy to implement, high variance in the average time will cause delays
  - **Shortest job first** where the smallest jobs get scheduled first. (The trouble is in knowing which jobs are shortest!) , non preemptive or preemptive
  - **Round robin** scheduling where each job is allotted a certain amount of CPU time. A context switch occurs when the time expires.
  - **Priority** scheduling preempts a job with a lower priority when a higher-priority job needs the CPU.

8.3 Protected Environments

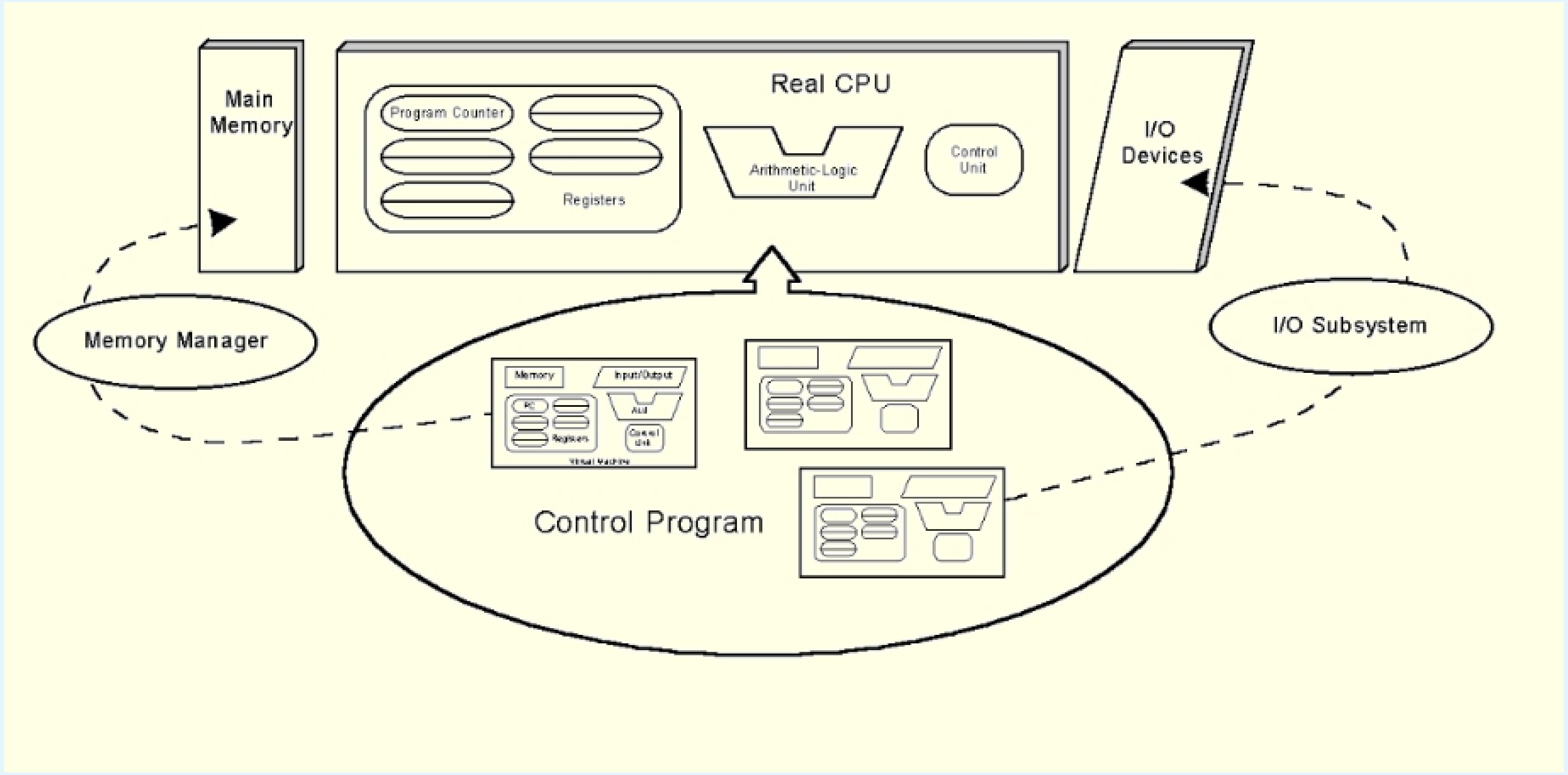
- In their role as resource managers and protectors, many operating systems provide protected environments that isolate processes, or groups of processes from each other.
- Three common approaches to establishing protected environments are virtual machines, subsystems, and partitions.
- These environments simplify system management and control, and can provide emulated machines(模拟机) to enable execution of programs that the system would otherwise be unable to run.

8.3 Protected Environments

- Virtual machines are a protected environment that presents an image of itself -- or the image of a totally different architecture -- to the processes that run within the environment.
- A virtual machine is exactly that: an imaginary computer.
- The underlying real machine is under the control of the kernel. The kernel receives and manages all resource requests that emit from processes running in the virtual environment.

The next slide provides an illustration.

8.3 Protected Environments



8.3 Protected Environments

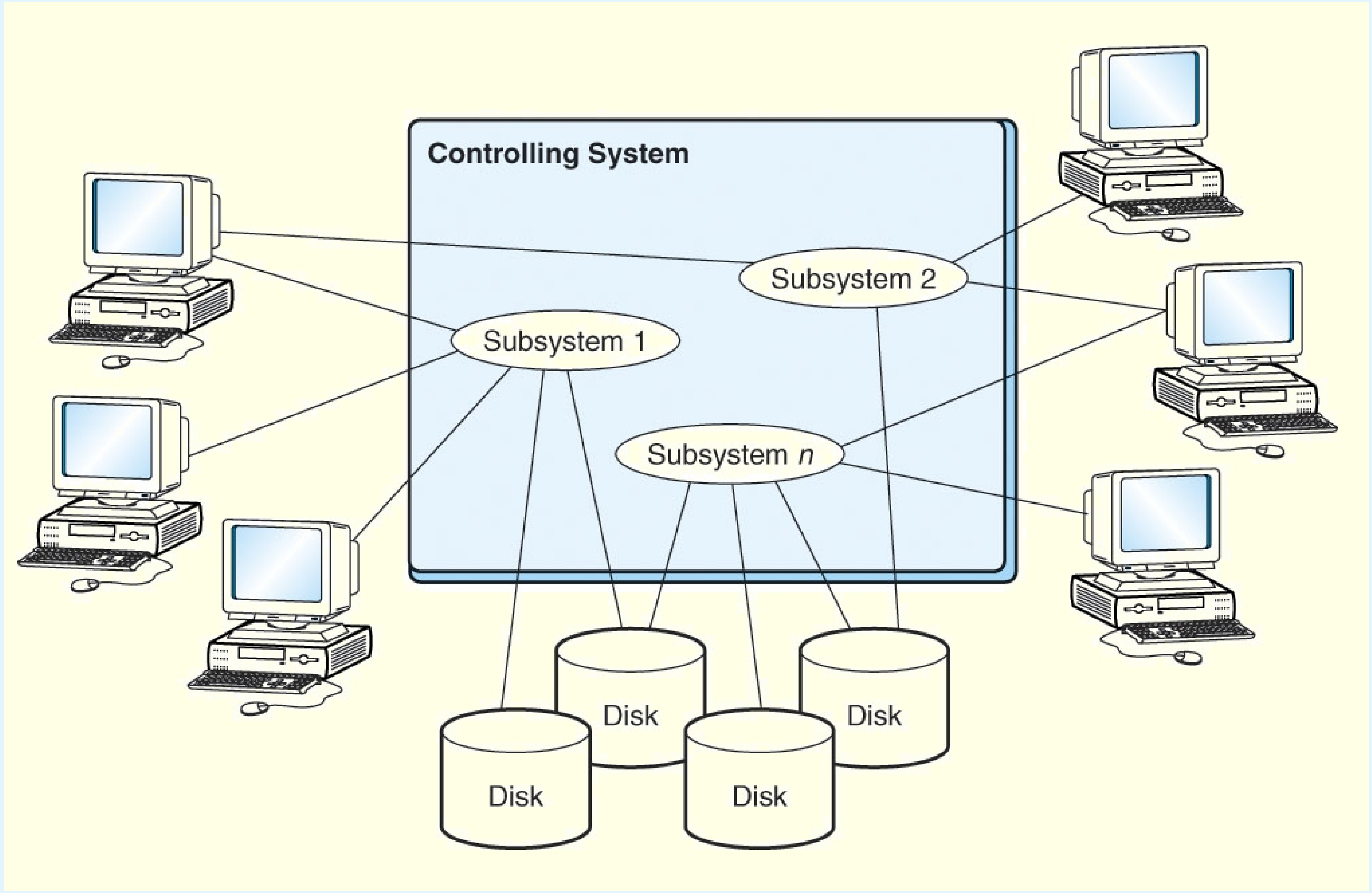
- MS-DOS prompt on a Microsoft windows uses a virtual machine environment
- The virtual machine manager(VMM) is a 32 bit protected-mode subsystem, that creates, runs, monitors and terminates virtual machine
- VMM creates an “MS-DOS” machine running under a virtual image of a 16-bit Intel 8086/8088 processor
- The VMM controlling program converts the 16-bit instructions to 32-bit instructions before they are executed on the real system processor

8.3 Protected Environments

- Subsystems are another type of protected environment.
- They provide logically distinct environments that can be individually controlled and managed. They can be stopped and started independent on each other.
  - Subsystems can have special purposes, such as controlling I/O or virtual machines. Others partition large application systems to make them more manageable.
  - In many cases, resources must be made visible to the subsystem before they can be accessed by the processes running within it.

The next slide provides an illustration.

8.3 Protected Environments

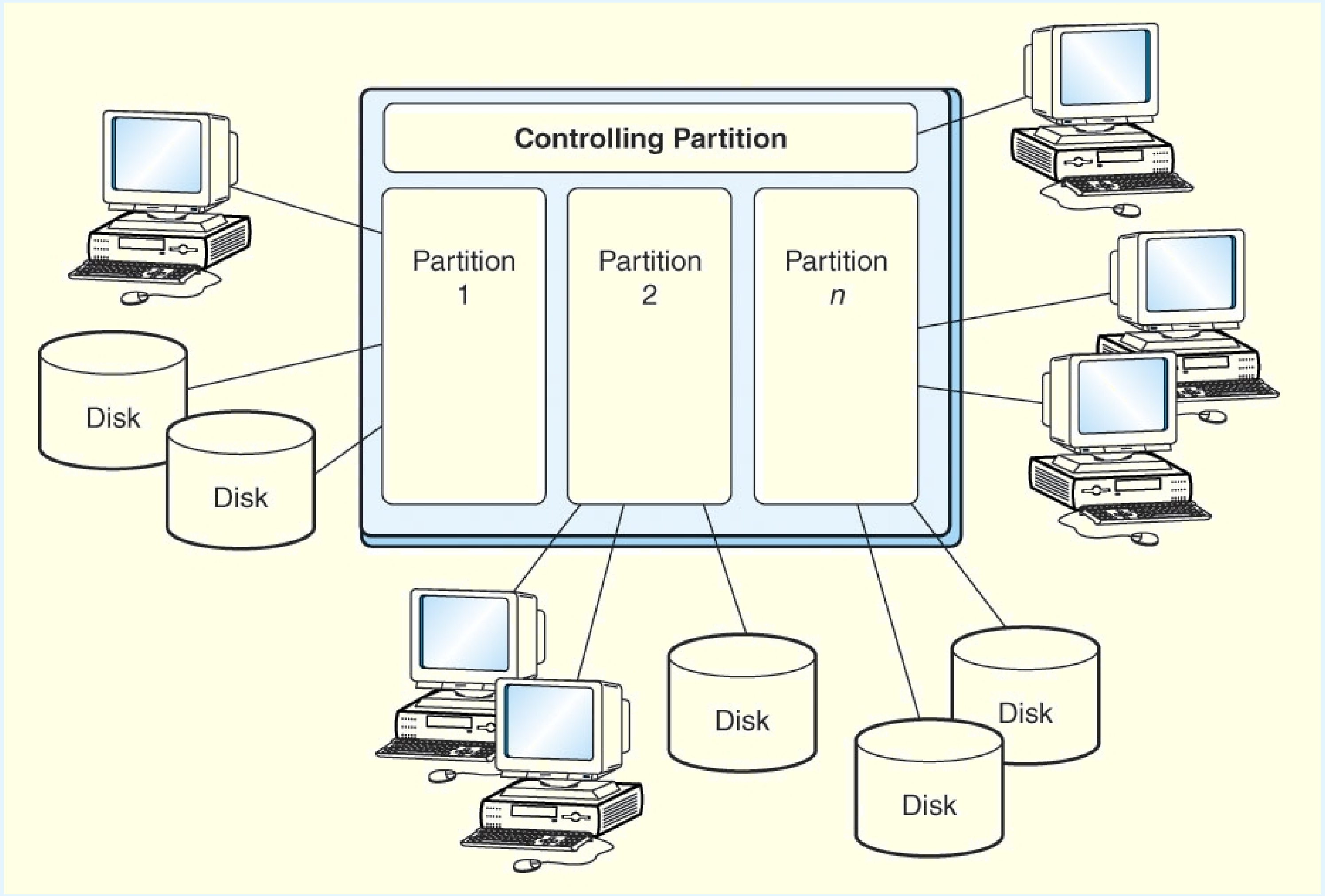


8.3 Protected Environments

- In very large computers, subsystems do not go far enough to establish a protected environment.
- Logical partitions (LPARs) provide much higher barriers: Processes running within a logical partition have no access to processes running in another partition unless a connection between them (e.g., FTP) is explicitly established.
- LPARs are an enabling technology for the recent trend of consolidating hundreds of small servers within the confines of a single large system.

The next slide provides an illustration.

8.3 Protected Environments

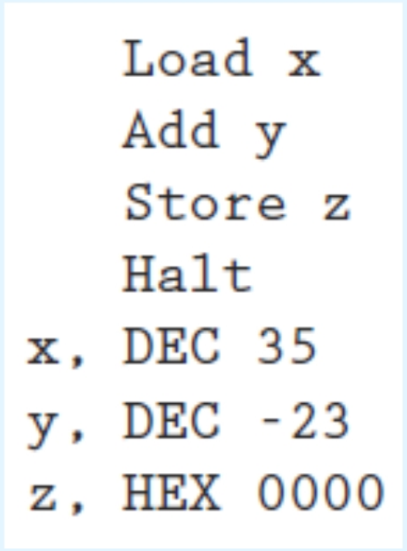


8.4 Programming Tools

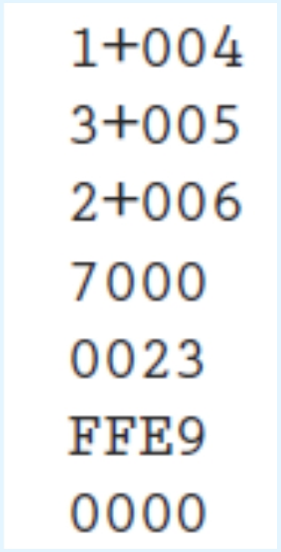
- Assemblers are the simplest of all programming tools. They translate mnemonic instructions to machine code.
- Most assemblers carry out this translation in two passes over the source code.
  - The first pass partially assembles the code and builds the symbol table
  - The second pass completes the instructions by supplying values stored in the symbol table.

8.4 Programming Tools

- The output of most assemblers is a stream of relocatable binary code.
- Binary code is relocatable when the addresses of the operands are relative to the location where the operating system has loaded the program in memory



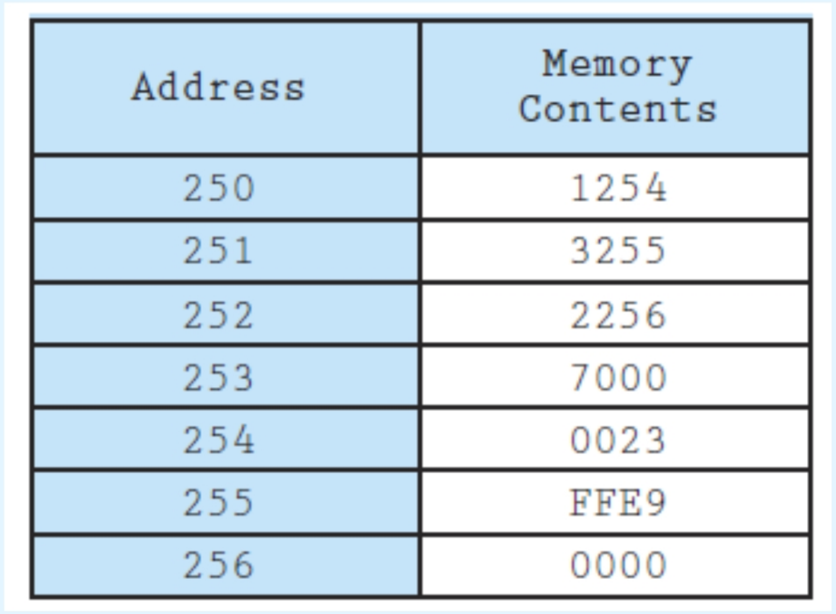
```
   Load x
   Add y
   Store z
   Halt
x, DEC 35
y, DEC -23
z, HEX 0000
```



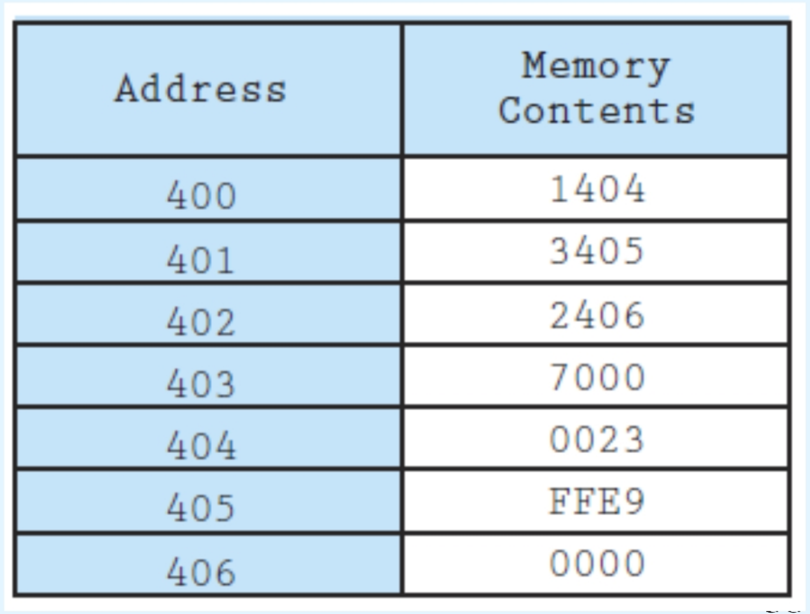
```
1+004
3+005
2+006
7000
0023
FFE9
0000
```

8.4 Programming Tools

- The "+" sign signals the program loader (component of the operating system) that the 004 in the first instruction is relative to the starting address of the program



Address	Memory Contents
250	1254
251	3255
252	2256
253	7000
254	0023
255	FFE9
256	0000



Address	Memory Contents
400	1404
401	3405
402	2406
403	7000
404	0023
405	FFE9
406	0000

8.4 Programming Tools

- In contrast to relocatable code, *absolute code* is executable binary code that must always be loaded at a particular location in memory . Often used in embedded systems which have space constraints
- binary machine instructions cannot be provided with “+” signs to distinguish between relocatable and nonrelocatable code
- One of the simplest ways to distinguish between the two is to use different file types (extensions) for this purpose
- The MS-DOS operating system uses a .COM extension for nonrelocatable code and .EXE extension for relocatable code
- COM files always load at address 100h. EXE files can load anywhere and they don’t even have to occupy contiguous memory space

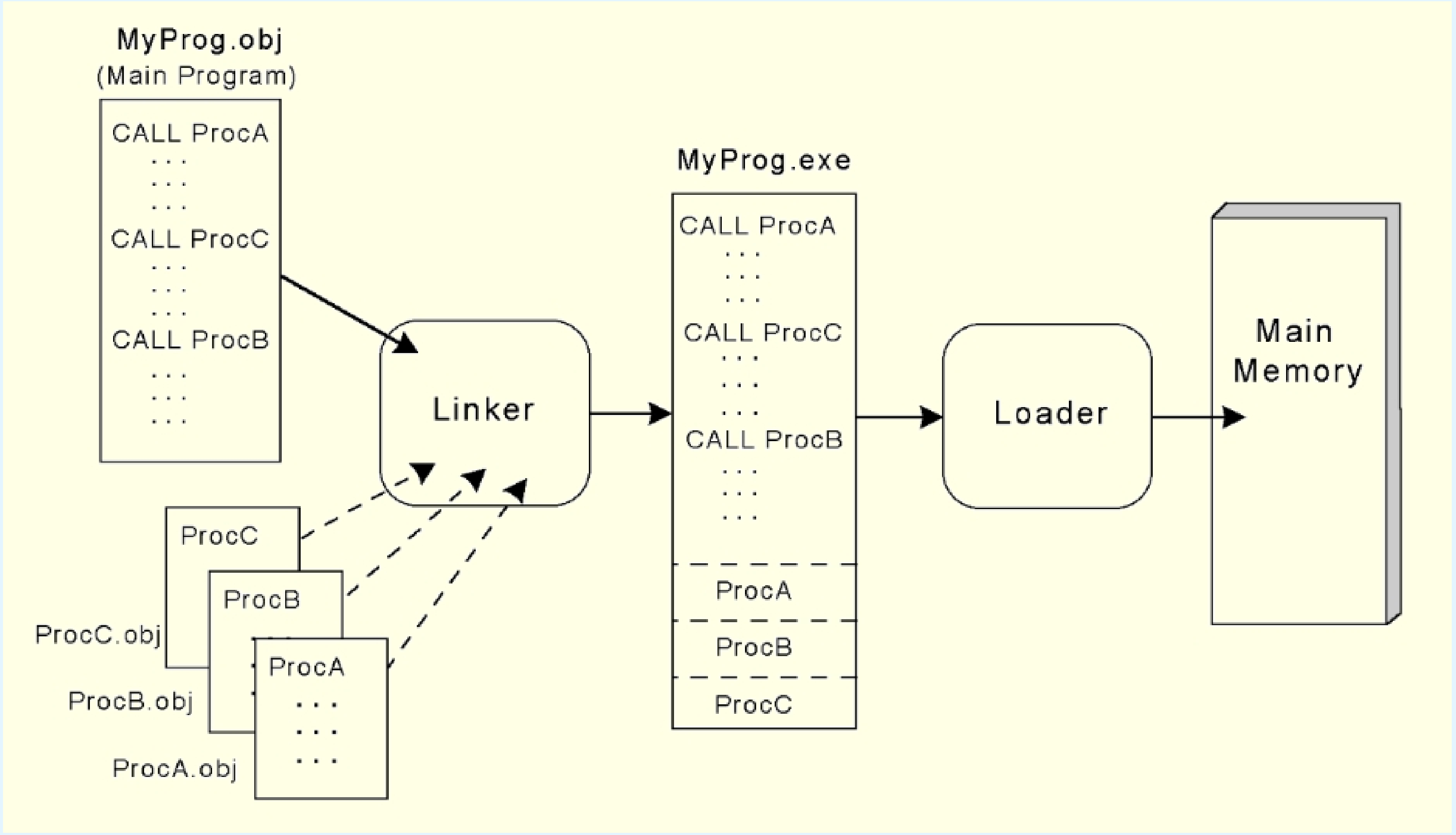
8.4 Programming Tools

- The process of assigning physical addresses to program variables is called *binding*.
- Binding can occur at compile time, load time, or run time.
- Compile time binding gives us absolute code.
- Load time binding assigns physical addresses as the program is loaded into memory.
  - With load time, binding the program cannot be moved!
- Run time binding requires a base register to carry out the address mapping.
  - Requires hardware support

8.4 Programming Tools

- On most systems, binary instructions must pass through a link editor (or linker) to create an executable module.
- Linking is the process of matching the external symbols of a program with all exported symbols from other files, producing a single binary file with no unresolved external symbols
- Like assemblers, link editors perform two passes: The first pass creates a symbol table and the second resolves references to the values in the symbol table.

8.4 Programming Tools



8.4 Programming Tools

- Dynamic linking is when the link editing is delayed until load time or at run time.
- External modules are loaded from *dynamic link libraries* (DLLs).
- Load time dynamic linking slows down program loading, but calls to the DLLs are faster.
- Run time dynamic linking occurs when an external module is first called, causing slower execution time.

8.4 Programming Tools

- Advantages：
  - if an external module is used repeatedly by several programs, static linking would require that each of these programs include a copy of the modules’ binary code. so we save space by linking at runtime
  - if the code in one of the external modules changes, then each module that has been linked with it does not need to be relinked to preserve the integrity of the program
  - dynamic linking provides the means whereby third parties can create common libraries

8.4 Programming Tools

- Assembly language programming can do many things that higher-level languages can’t do
  - assembly language gives the programmer direct access to the underlying machine architecture
  - Programs used to control and/or communicate with peripheral devices are typically written in assembly
  - A programmer doesn’t have to rely on operating system services to control a communications port
  - it produces the most timely and effective action upon the system

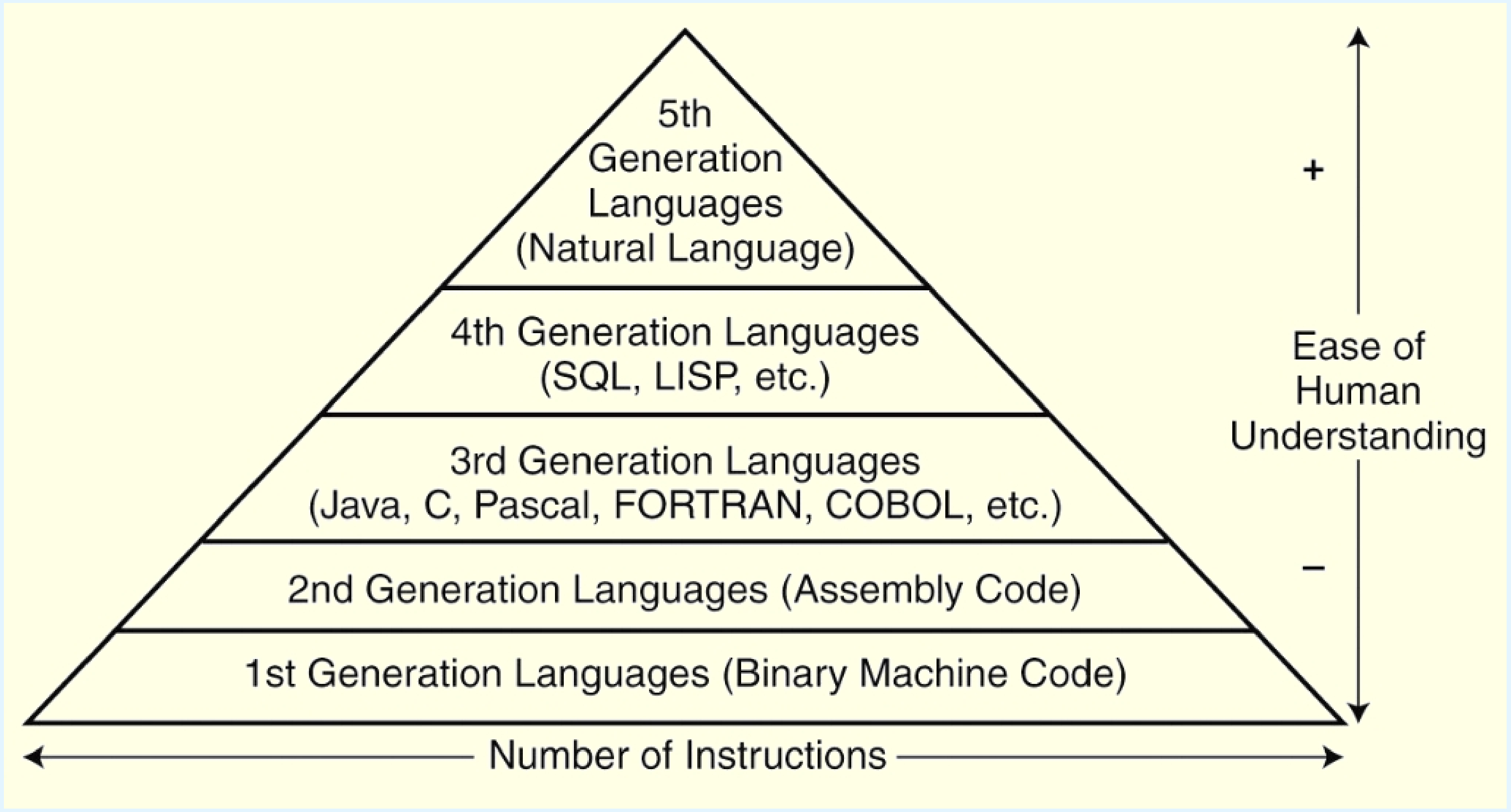
8.4 Programming Tools

- These advantages are not sufficiently compelling reasons to use assembly language for general application development.
  - programming in assembly language is difficult and error-prone
  - It is even more difficult to maintain than it is to write
  - assembly languages are not portable to different machine architectures

8.4 Programming Tools

- Assembly language is considered a “second generation” programming language (2GL).
- Compiled programming languages, such as C, C++, Pascal, and COBOL, are “third generation” languages (3GLs).
- Each “generation” of programming languages gets closer to how people think about problems and more distant from how machinery solves them
- simply tells the computer what to do, not how to do it

8.4 Programming Tools


5th Generation Languages (Natural Language)
4th Generation Languages (SQL, LISP, etc.)
3rd Generation Languages (Java, C, Pascal, FORTRAN, COBOL, etc.)
2nd Generation Languages (Assembly Code)
1st Generation Languages (Binary Machine Code)
Number of Instructions
+
Ease of Human Understanding
–

Keep in mind that the computer can understand only the 1GL!

8.4 Programming Tools

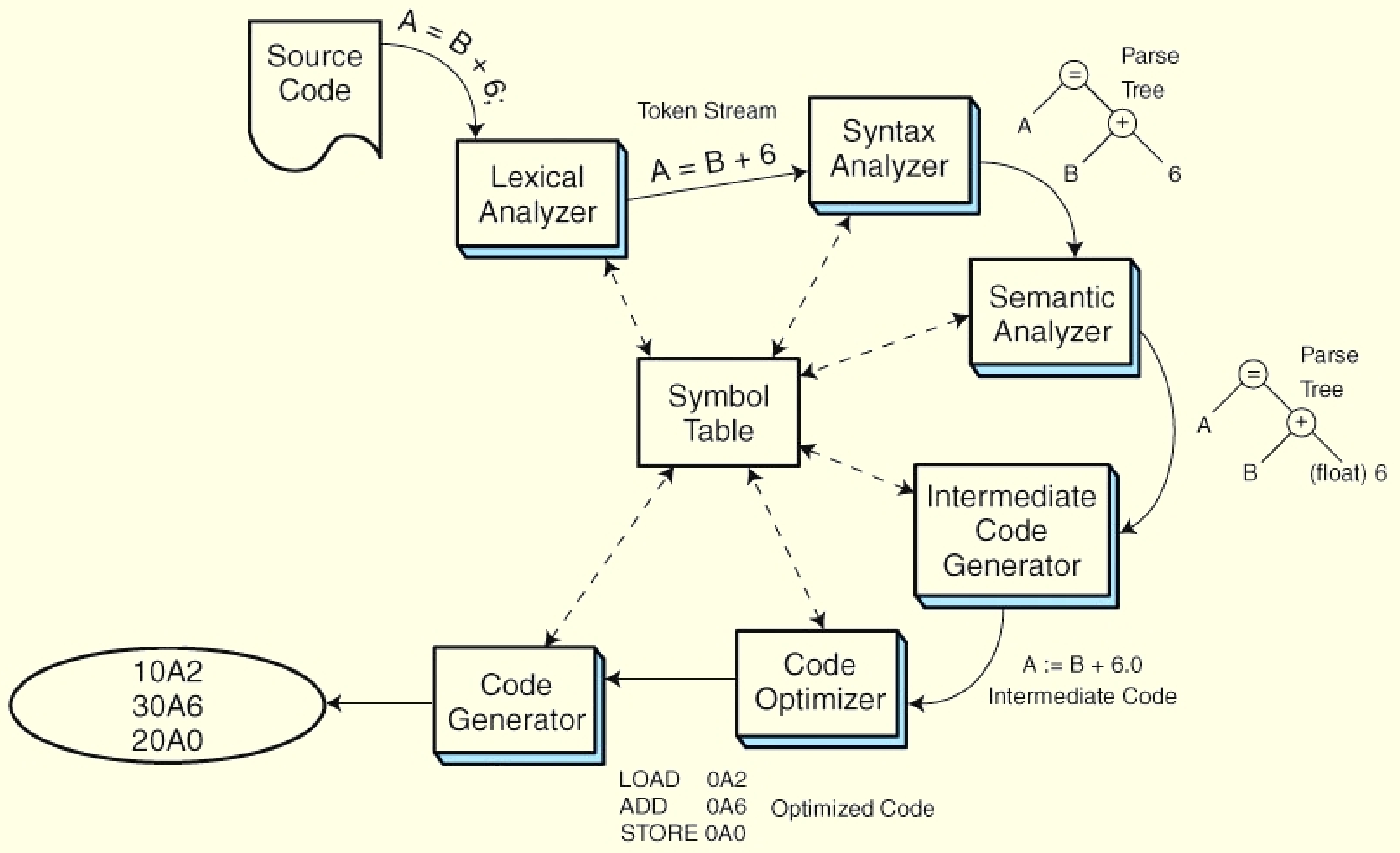
- Compilers bridge the semantic gap between the higher level language and the machine’s binary instructions.
- Most compilers effect this translation in a six-phase process. The first three are analysis phases:
  1. **Lexical(词汇) analysis** extracts tokens, e.g., reserved words and variables.
  2. **Syntax(语法) analysis** (parsing) checks statement construction.
  3. **Semantic(语义) analysis** checks data types and the validity of operators.

8.4 Programming Tools

- The last three compiler phases are synthesis phases:
  4. **Intermediate code generation** creates *three address code* to facilitate optimization and translation.
  5. **Optimization** creates assembly code while taking into account architectural features that can make the code efficient.
  6. **Code generation** creates binary code from the optimized assembly code.
- Through this modularity, compilers can be written for various platforms **by rewriting only the last two phases.**

The next slide shows this process graphically.

8.4 Programming Tools



8.4 Programming Tools

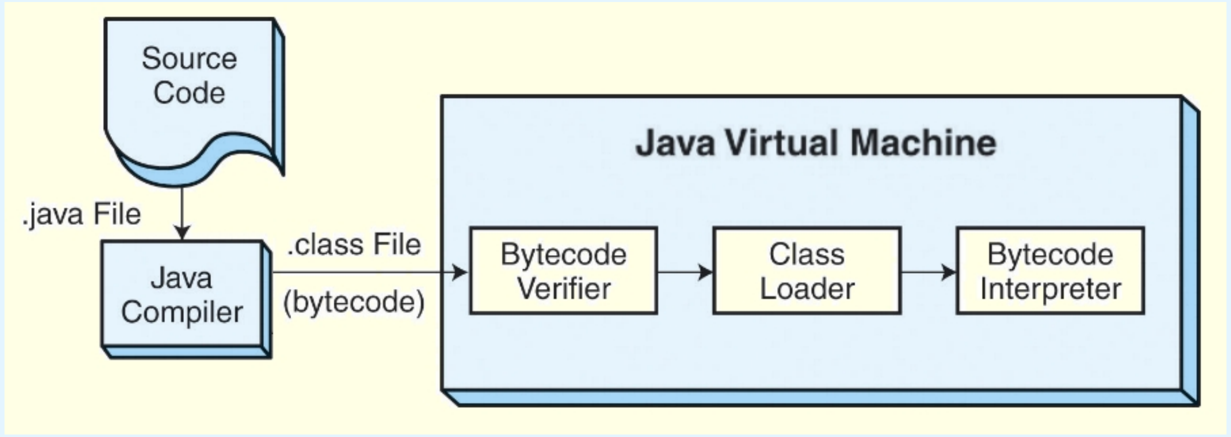
- Interpreters produce executable code from source code in real time, one line at a time.
- Consequently, this not only makes interpreted languages slower than compiled languages but it also affords less opportunity for error checking.
- Interpreted languages are, however, very useful for teaching programming concepts, because feedback is nearly instantaneous, and performance is rarely a concern.

8.5 Java: All of the Above

- The Java programming language exemplifies many of the concepts that we have discussed in this chapter.
- Java programs (*classes*) execute within a virtual machine, the *Java Virtual Machine* (JVM).
- This allows the language to run on any platform for which a virtual machine environment has been written.
- Java is both a **compiled** and an **interpreted** language. The output of the compilation process is an assembly-like intermediate code (*bytecode)* that is interpreted by the JVM.

8.5 Java: All of the Above

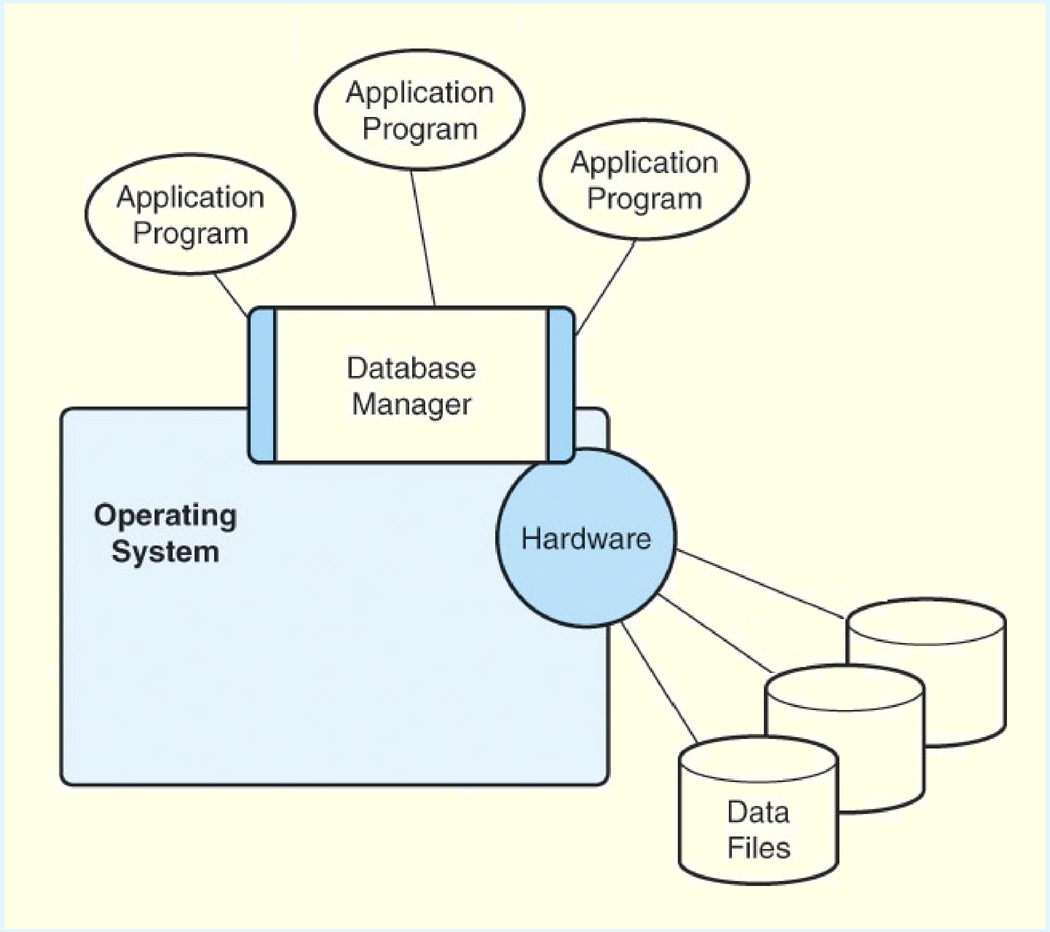
- The JVM is an operating system in miniature.
  - It loads programs, links them, starts execution threads, manages program resources, and deallocates resources when the programs terminate.



- Because the JVM performs so many tasks at run time, its performance cannot match the performance of a traditional compiled language.

8.6 Database Software

- Database systems contain the most valuable assets of an enterprise. They are the foundation upon which application systems are built.



8.6 Database Software

- Database systems provide a single definition, the database *schema*, for the data elements that are accessed by application programs.
  - A **physical schema** is the computer's view of the database that includes locations of physical files and indexes.
  - A **logical schema** is the application program's view of the database that defines field sizes and data types.
- Within the logical schema, certain data fields are designated as record keys that provide efficient access to records in the database.

8.6 Database Software

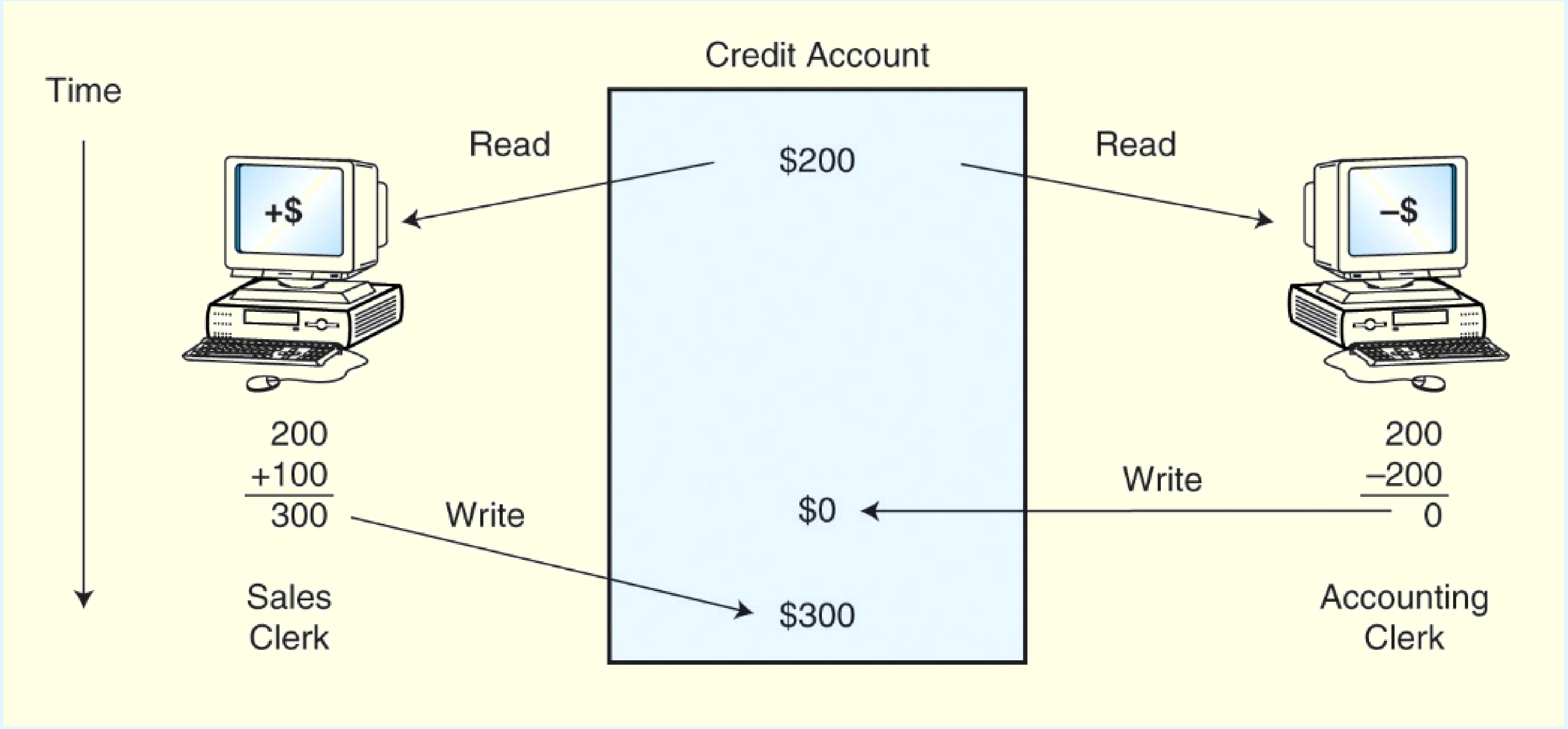
- Keys are stored in physical index file structures containing pointers to the location of the physical records.
- Many implementations use a variant of a B+ tree for index management because B+ trees can be optimized with consideration to the I/O system and the applications.
- In many cases, the “higher” nodes of the tree will persist in cache memory, requiring physical disk accesses only when traversing the lower levels of the index.

8.6 Database Software

- Most database systems also include transaction management components to assure that the database is always in a consistent state.
- Transaction management provides the following properties:
  – **Atomicity** - All related updates occur or no updates occur.
  – **Consistency** - All updates conform to defined data constraints.
  – **Isolation** - No transaction can interfere with another transaction.
  – **Durability** - Successful updates are written to durable media as soon as possible.
- These are the *ACID* properties of transaction management.

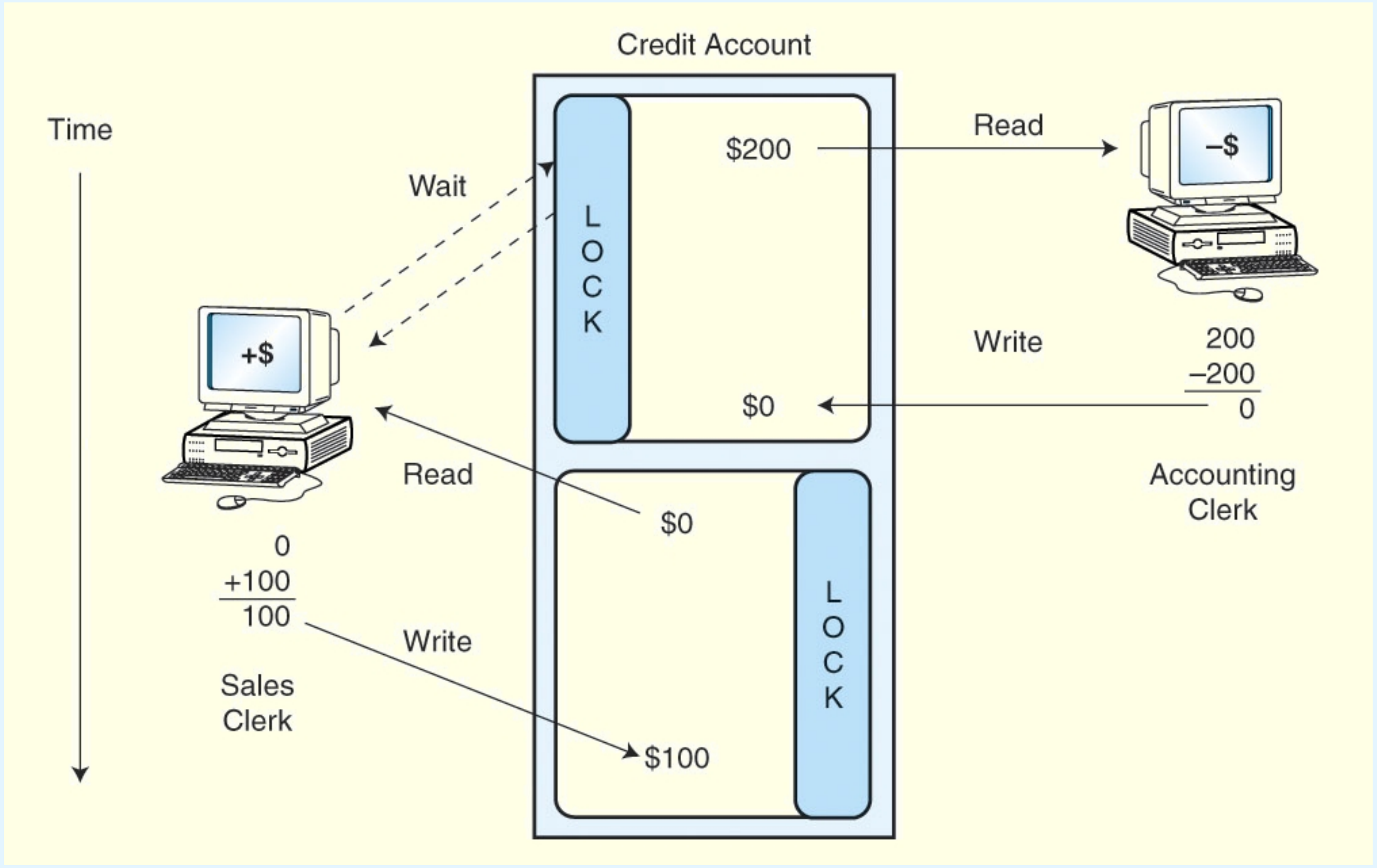
8.6 Database Software

- Without the ACID properties, *race conditions* can occur:



8.6 Database Software

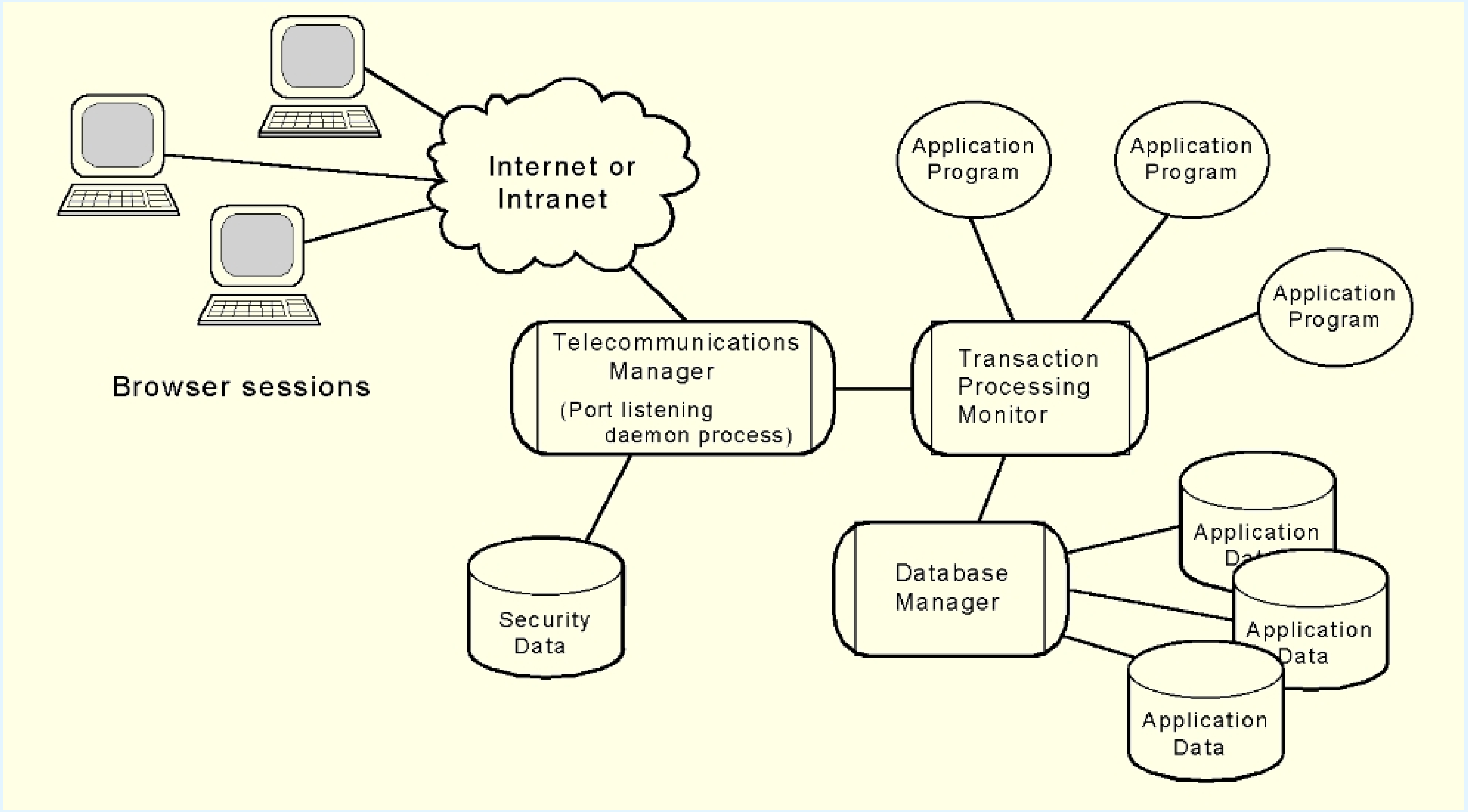
- Record locking mechanisms assure isolated, atomic database updates:



8.7 Transaction Managers

- One way to improve database performance is to ask it to do less work by moving some of its functions to specialized software.
- Transaction management is one component that is often partitioned from the core database system.
- Transaction managers are especially important when the transactions involve more than one physical database, or the application system spans more than one class of computer, as in a multitiered architecture.
- One of the most widely-used transaction management systems is CICS shown on the next slide.

8.7 Transaction Managers



Chapter 8 Conclusion

- The proper functioning and performance of a computer system depends as much on its software as its hardware.
- The operating system is the system software component upon which all other software rests.
- Operating systems control process execution, resource management, protection, and security.
- Subsystems and partitions provide compatibility and ease of management.

Chapter 8 Conclusion

- Programming languages are often classed into generations, with assembly language being the first generation.
- All languages above the machine level must be translated into machine code.
- Compilers bridge this semantic gap through a series of six steps.
- Link editors resolve system calls and external routines, creating a unified executable module.

Chapter 8 Conclusion

- The Java programming language incorporates the idea of a virtual machine, a compiler and an interpreter.
- Database software provides controlled access to data files through enforcement of ACID properties.
- Transaction managers provide high performance and cross-platform access to data.