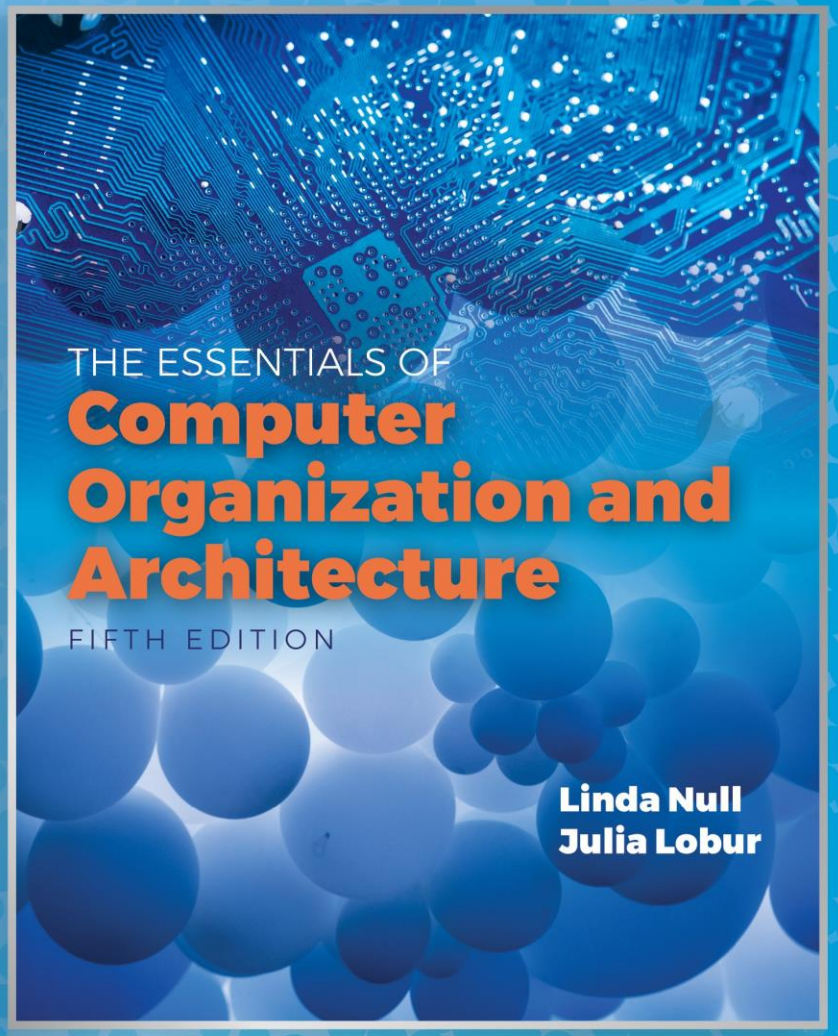
This is the second lecture of Chapter 8

Chapter 8

System Software (B)



Quick review of last lecture

- Operating Systems
  - Operating Systems History
  - Operating Systems Design
  - Operating System Services
- Protected Environments
  - Virtual Machines
  - Subsystems
  - Logical Partitions

8.4 Programming Tools (1 of 13)

8.4.1 Assembler and Assembly

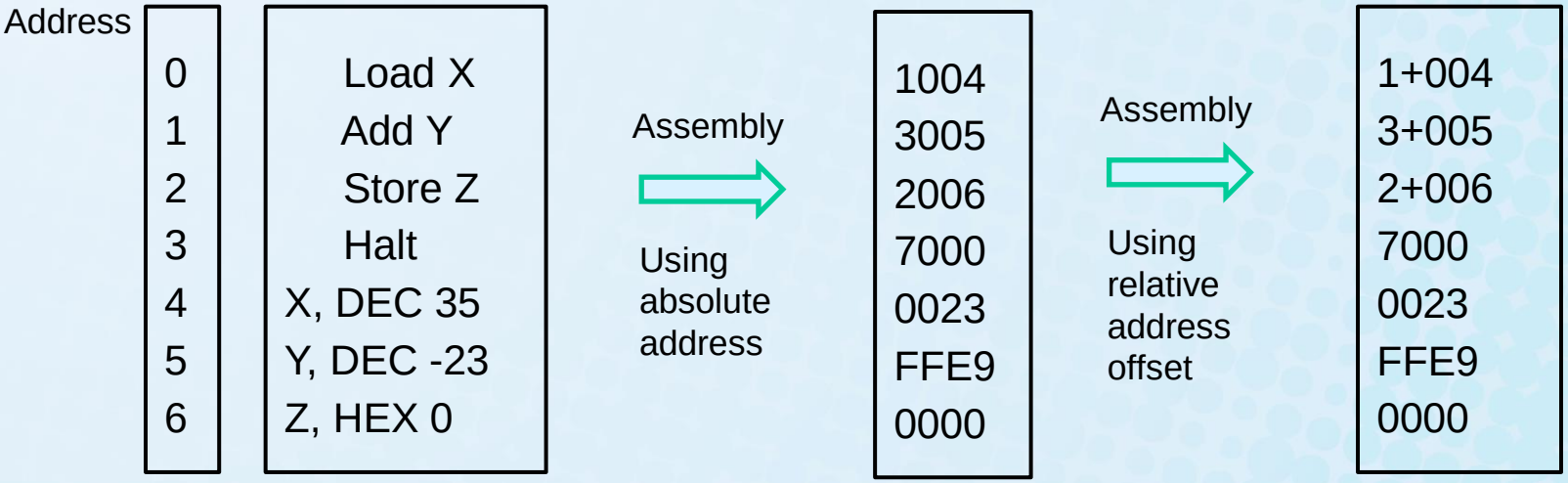
- Programming tools carry out the mechanics of software creation within the confines of the operating system and hardware environment.
- Assemblers are the simplest of all programming tools. They translate mnemonic instructions to machine code.
- Most assemblers carry out this translation in two passes 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 (2 of 13)

- The output of most assemblers is a stream of relocatable binary code.
  - In relocatable code, operand addresses are relative to where the operating system chooses to load the program.
  - Absolute (nonrelocatable) code is most suitable for device and operating system control programming.
- When relocatable code is loaded for execution, special registers provide the base addressing.
- Addresses specified within the program are interpreted as offsets from the base address.

The next slide illustrates this idea.

- Example: MARIE Code



Address	Memory Contents
0x250	1254
0x251	3255
0x252	2256
0x253	7000
0x254	0023
0x255	FFE9
0x256	0000

Address	Memory Contents
0x400	1404
0x401	3405
0x402	2406
0x403	7000
0x404	0023
0x405	FFE9
0x406	0000

8.4 Programming Tools (4 of 13)

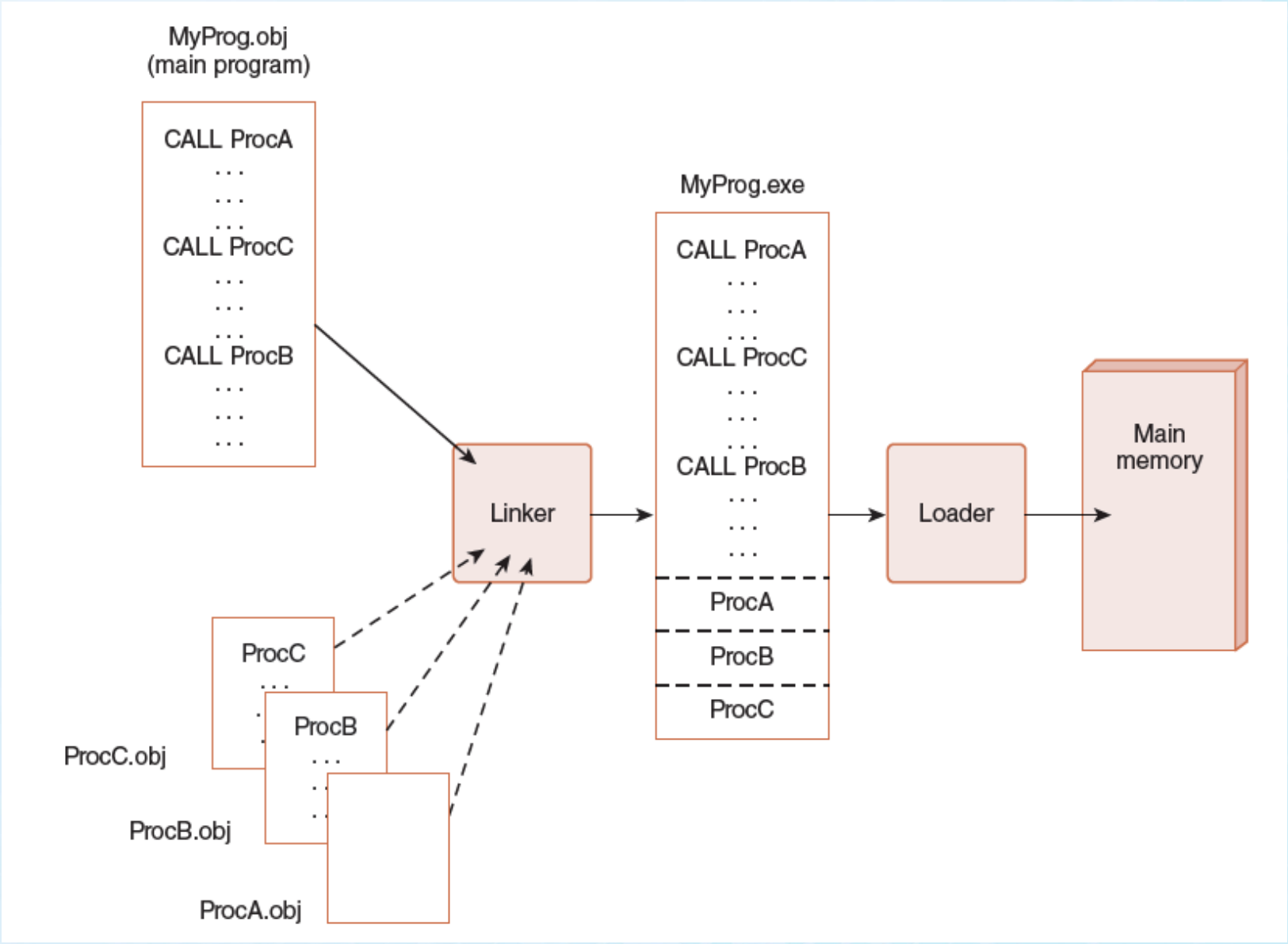
- The process of assigning physical addresses to program variables is called *binding*.
- Binding can occur at compile time, load time, or run time.
- Compile time binding gives us absolute code.
- Load time binding assigns physical addresses as the program is loaded into memory.
  - With load time,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

8.4 Programming Tools (5 of 13)

8.4.2 Link Editors

- On most systems, binary instructions must pass through a link editor (or linker) to create an executable module.
- Link editors incorporate various binary routines into a single executable file as called for by a program's 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.

The next slide shows this process schematically.



8.4 Programming Tools (7 of 13)

8.4.3 Dynamic Link Libraries

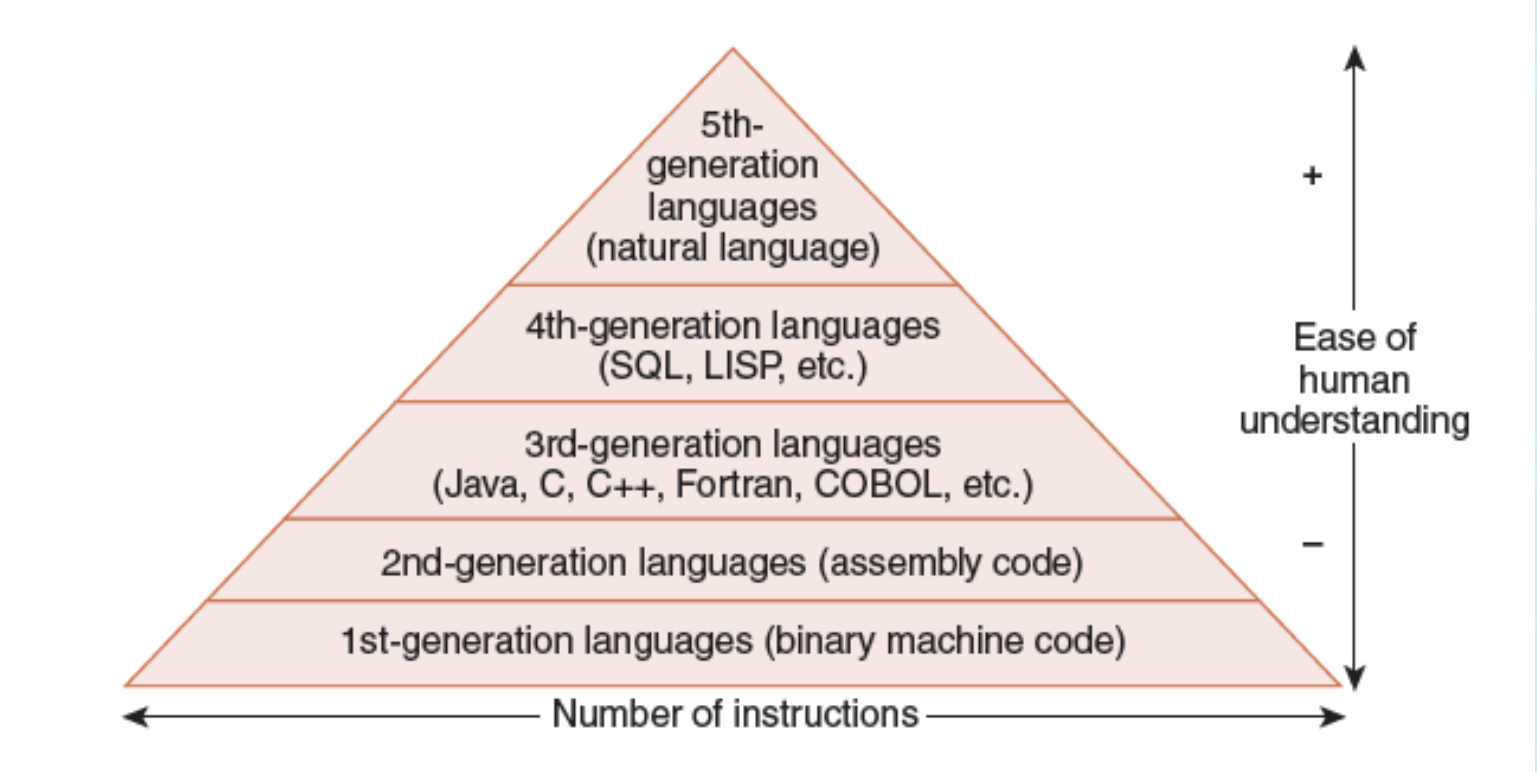
- Dynamic linking is when the link editing is delayed until load time or at run time.
- External modules are loaded from *dynamic link libraries* (DLLs).
- Load time dynamic linking slows down program loading, but calls to the DLLs are faster.
- Run time dynamic linking occurs when an external module is first called, causing slower execution time.
  - Dynamic linking makes program modules smaller, but carries the risk that the programmer may not have control over the DLL.

8.4 Programming Tools (8 of 13)

8.4.4 Compilers

- 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).
- Higher language generation presents problem solving tools that are closer to how people think and farther away from how the machine implements the solution.

8.4 Programming Tools (9 of 13)



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

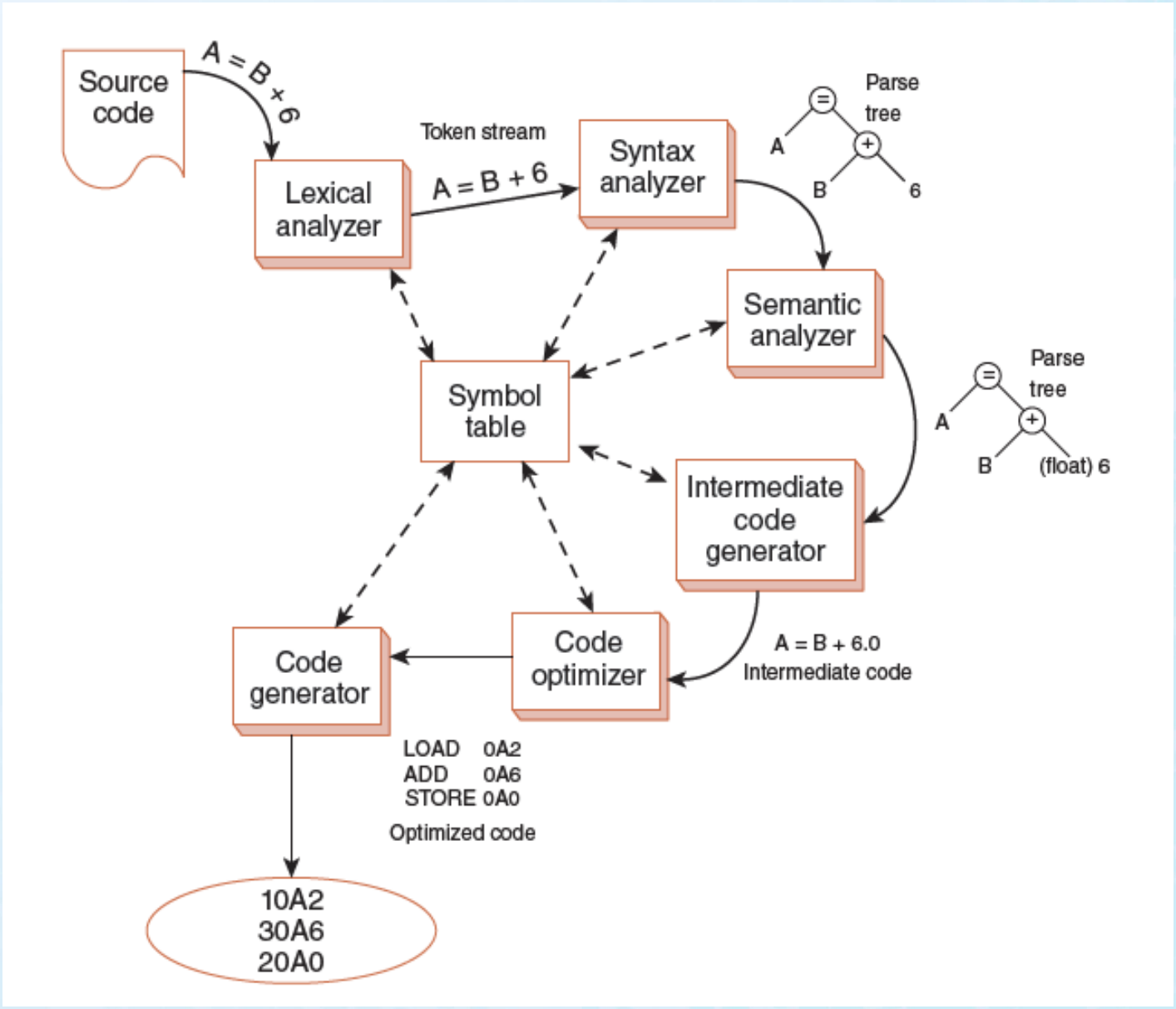
8.4 Programming Tools (10 of 13)

- 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 analysis extracts tokens (e.g., reserved words and variables).
  - 2. Syntax analysis (parsing) checks statement construction.
  - 3. Semantic analysis checks data types and the validity of operators.

8.4 Programming Tools (11 of 13)

- 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 (13 of 13)

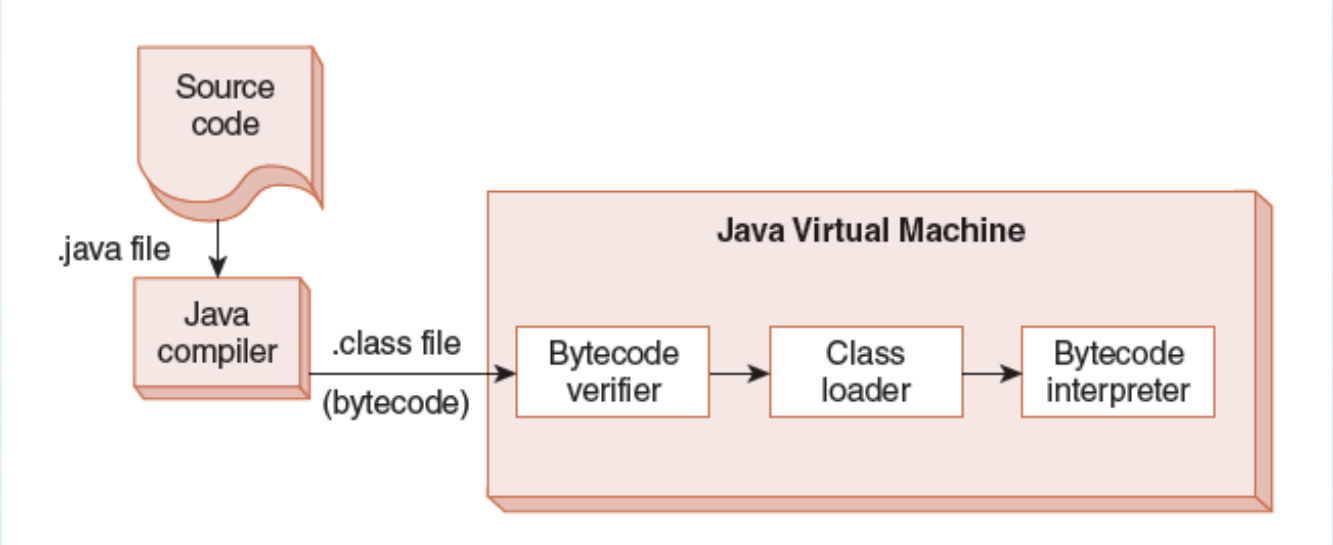
8.4.5 Interpreters

- 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 (1 of 5)

- 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.

- 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.

8.5 Java: All of the Above (3 of 5)

- At execution time, a Java Virtual Machine must be running on the host system.
- It loads and executes the bytecode class file.
- While loading the class file, the JVM verifies the integrity of the bytecode.
- The loader then performs a number of run-time checks as it places the bytecode in memory.
- The loader invokes the bytecode interpreter.

8.5 Java: All of the Above (4 of 5)

- The bytecode interpreter:
  - Performs a link edit of the bytecode instructions by asking the loader to supply all referenced classes and system binaries, if they are not already loaded.
  - Creates and initializes the main stack frame and local variables.
  - Creates and starts execution thread(s).
  - Manages heap storage by deallocating unused storage while the threads are executing.
  - Deallocates resources of terminated threads.
  - Upon program termination, kills any remaining threads and terminates the JVM.

8.5 Java: All of the Above (5 of 5)

- Because the JVM does so much as it loads and executes its bytecode, it can't match the performance of a compiled language.
  - This is true even when speedup software like Java's Just-In-Time (JIT) compiler is used.
- However class files can be created and stored on one platform and executed on a completely different platform.
- This "write once, run-anywhere" paradigm is of enormous benefit for enterprises with disparate and geographically separate systems.
- Given its portability and relative ease of use, the Java language and its virtual machine environment are the ideal middleware platform.

Conclusion (1 of 2)

- 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.

Conclusion (2 of 2)

- Programming languages are often classed into generations, with machine 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.
- The Java programming language incorporates the idea of a virtual machine, a compiler and an interpreter.