Introduction to System Programming

Dr. Wassim Ahmad

BOOKS:

- 1.Computer Systems_ A Programmer's Perspective 3rd Edition
- By: Brayant O'Hallaron

- 2. System Programming with C and Unix
- By: Adam Hoover

- 3. System Programming
- By: D. M. Dhamdhere

Computer Software:

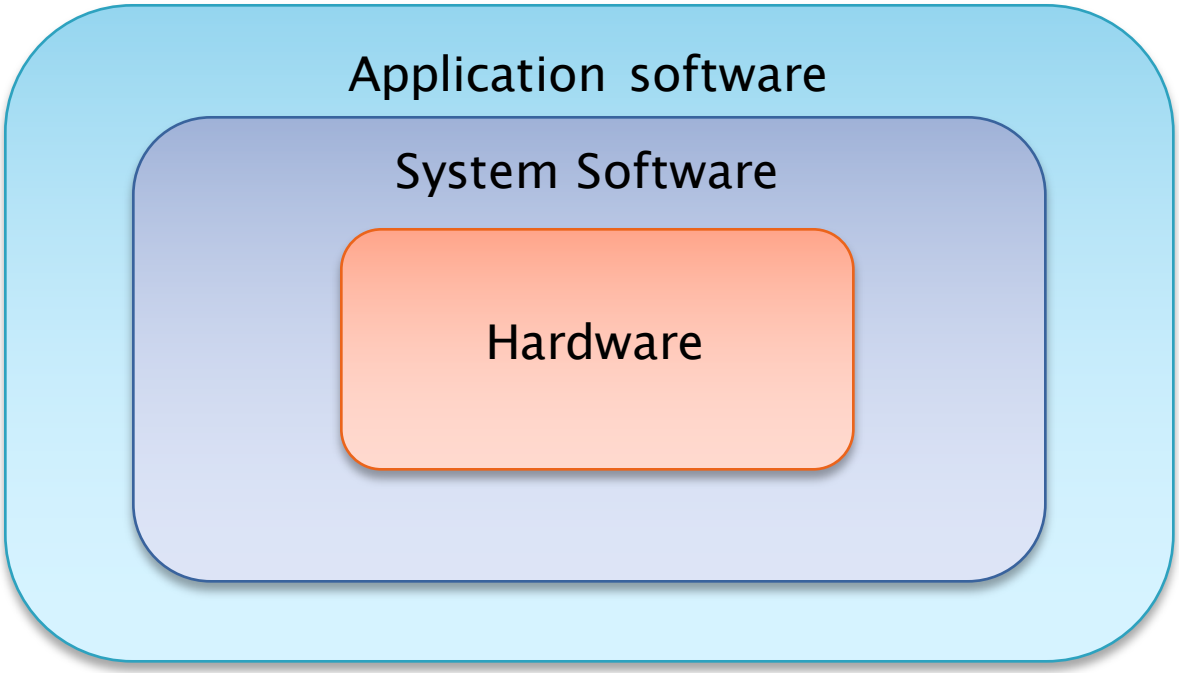
- **Computer software**, or simply **software**, refers to the non-tangible components of computers, known as computer programs. The term is used to contrast with computer hardware, which denotes the physical tangible components of computers.

Software classification

- Software can be classified into
  - **System software** is ***Computer Software*** designed to operate and control the ***Computer Hardware*** and to provide a platform for running ***Application Software***.
    - **System software** is collection of software program that perform a variety of functions like IO management, storage management, generation and execution of programs etc.
      - Operating Systems
      - Compiler / Assembler (utility software)
      - Device Drivers
  - **Application software**:
    - Application software is kind of software which is designed for fulfillment specialized user requirement.
      - MS Office
      - Adobe Photoshop

Cont.

- The system software work as middleware between application software and hardware.



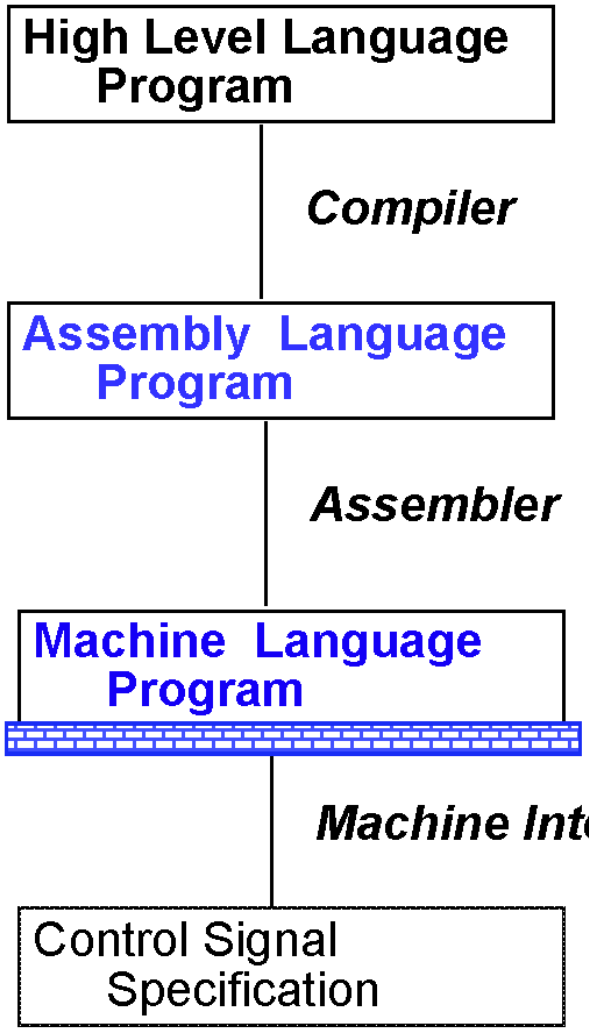
Programming and Problem Solving

- Specify the input:
  What values come into the program?
  What is the data type of each value?
  Where does each value come from?
- Computations:
  What new values must be computed?
  What is the data type of each new value?
  How are the values computed? (Algorithms)
- Specify the output:
  Which of the computed values are produced as output by the program.
  In what format is the output presented?

C: A High-Level Language

- Gives symbolic names to values
  - don't need to know which register or memory location
- Provides abstraction of underlying hardware
  - operations do not depend on instruction set
  - example: can write "a = b * c", even though
- Provides expressiveness
  - use meaningful symbols that convey meaning
  - simple expressions for common control patterns (if-then-else)
- Enhances code readability
- Safeguards against bugs
  - can enforce rules or conditions at compile-time or run-time

Levels of Representation



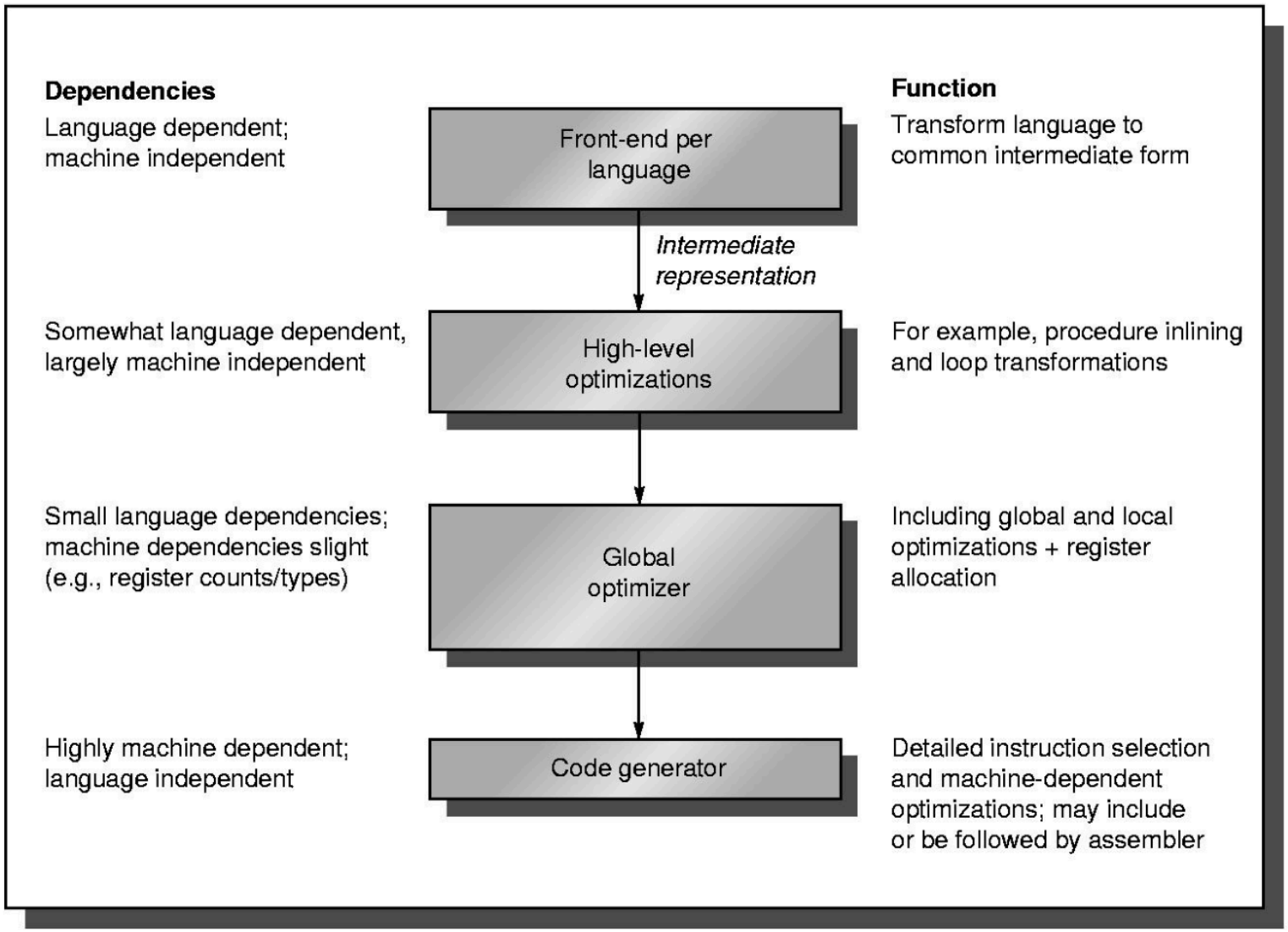

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
load    $15,  0($2)
load    $16,  4($2)
store   $16,  0($2)
store   $15,  4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Compiler structure

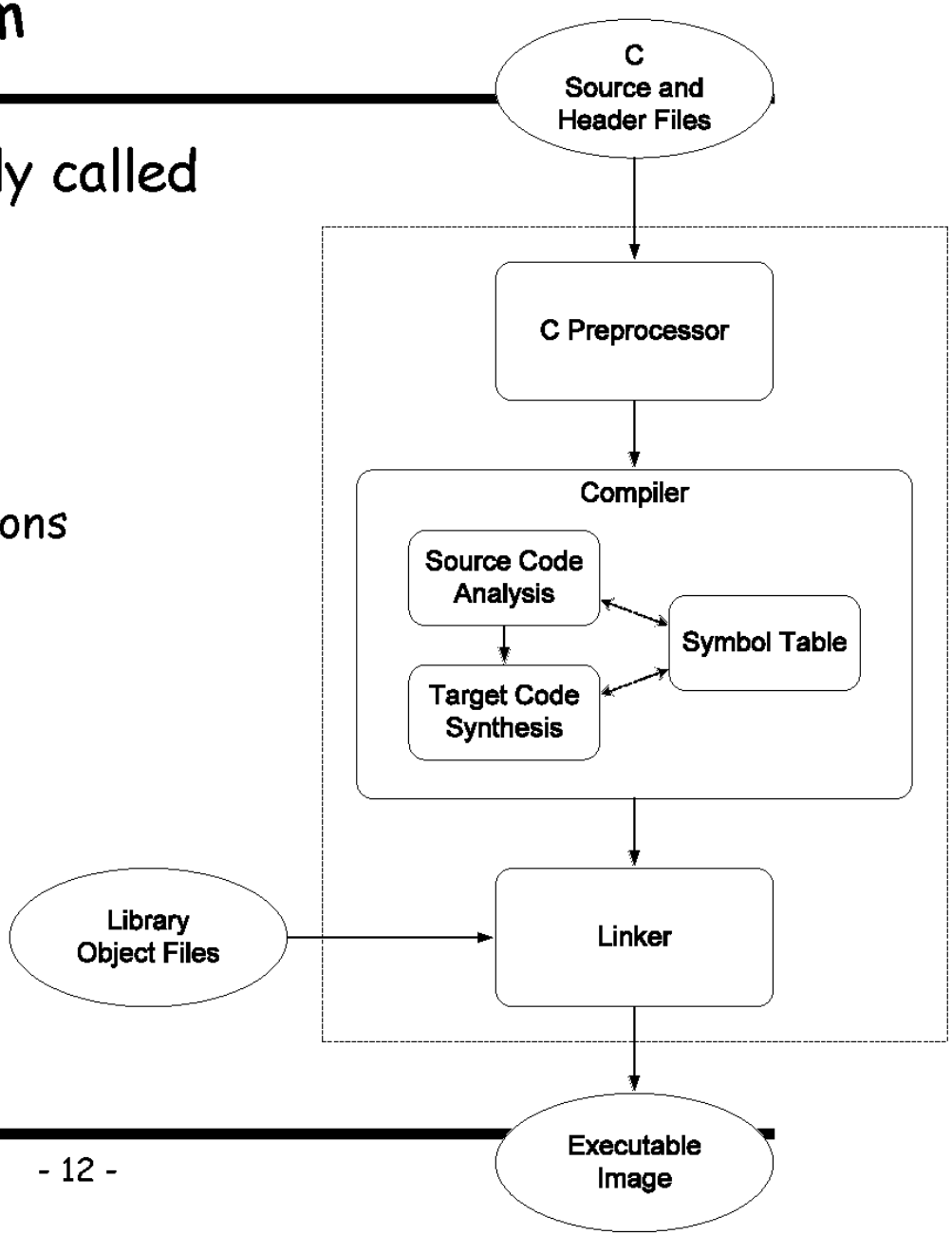


Compilation vs. Interpretation

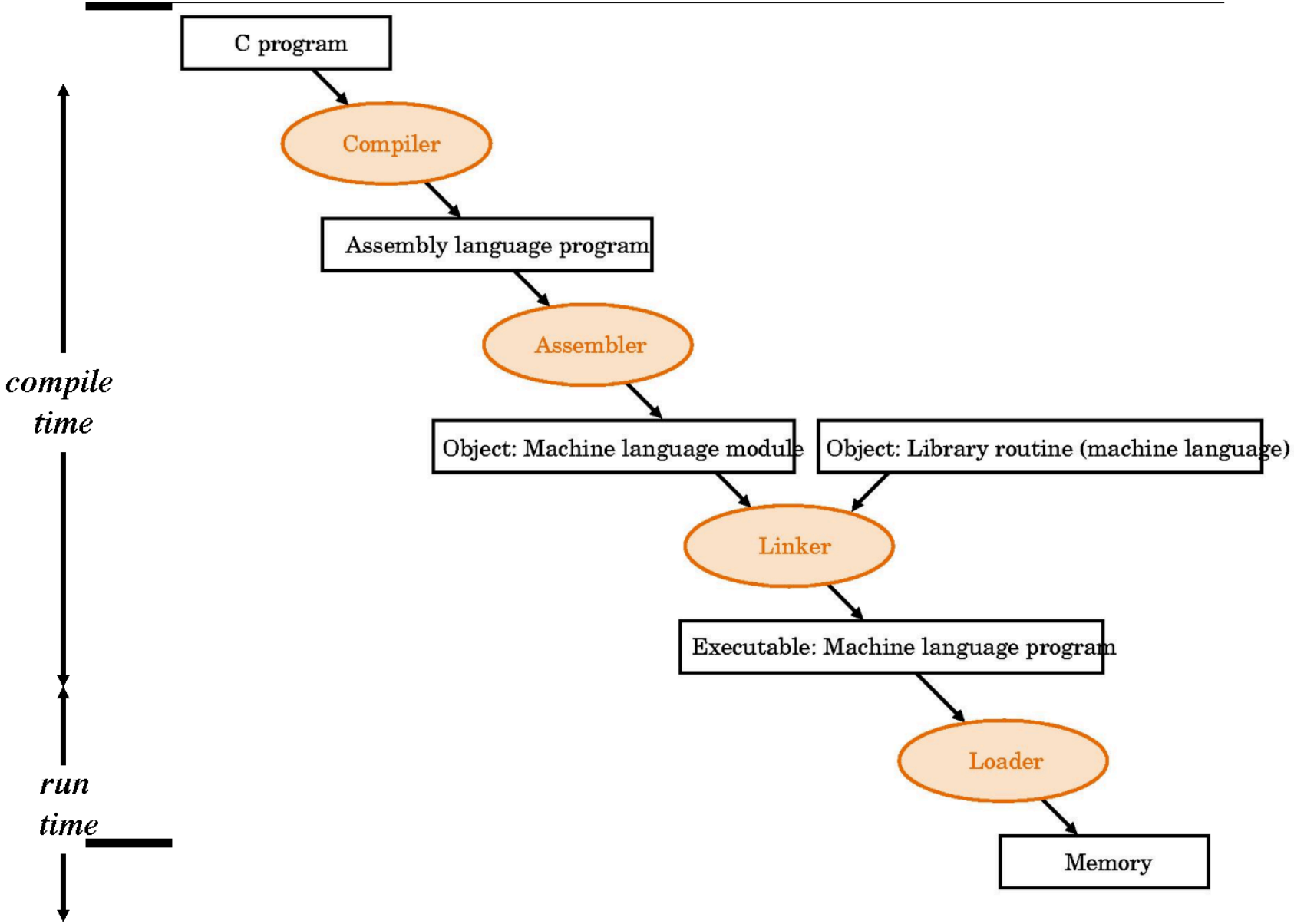
- Different ways of translating high-level language
- *Interpretation*
  - interpreter = program that executes program statements
  - generally one line/command at a time
  - limited processing
  - easy to debug, make changes, view intermediate results
  - languages: BASIC, LISP, Perl, Java, Matlab, C-shell
- *Compilation*
  - translates statements into machine language
    - does not execute, but creates executable program
  - performs optimization over multiple statements
  - change requires recompilation
    - can be harder to debug, since executed code may be different
  - languages: C, C++, Fortran, Pascal

Compiling a C Program

- Entire mechanism is usually called the "compiler"
- Preprocessor
  - macro substitution
  - conditional compilation
  - "source-level" transformations
    - output is still C
- Compiler
  - generates object file
    - machine instructions
- Linker
  - combine object files (including libraries) into executable image



Assembling, linking, loading programs



Compiling and Linking

- Various compilers available
  - cc, gcc, g++
  - includes preprocessor, compiler, and linker
- Lots and lots of options!
  - level of optimization, debugging
  - preprocessor, linker options
  - intermediate files --
    object (.o), assembler (.s), preprocessor (.i), etc.