CHAPTER 9
COMPUTER LANGUAGES

Introduction

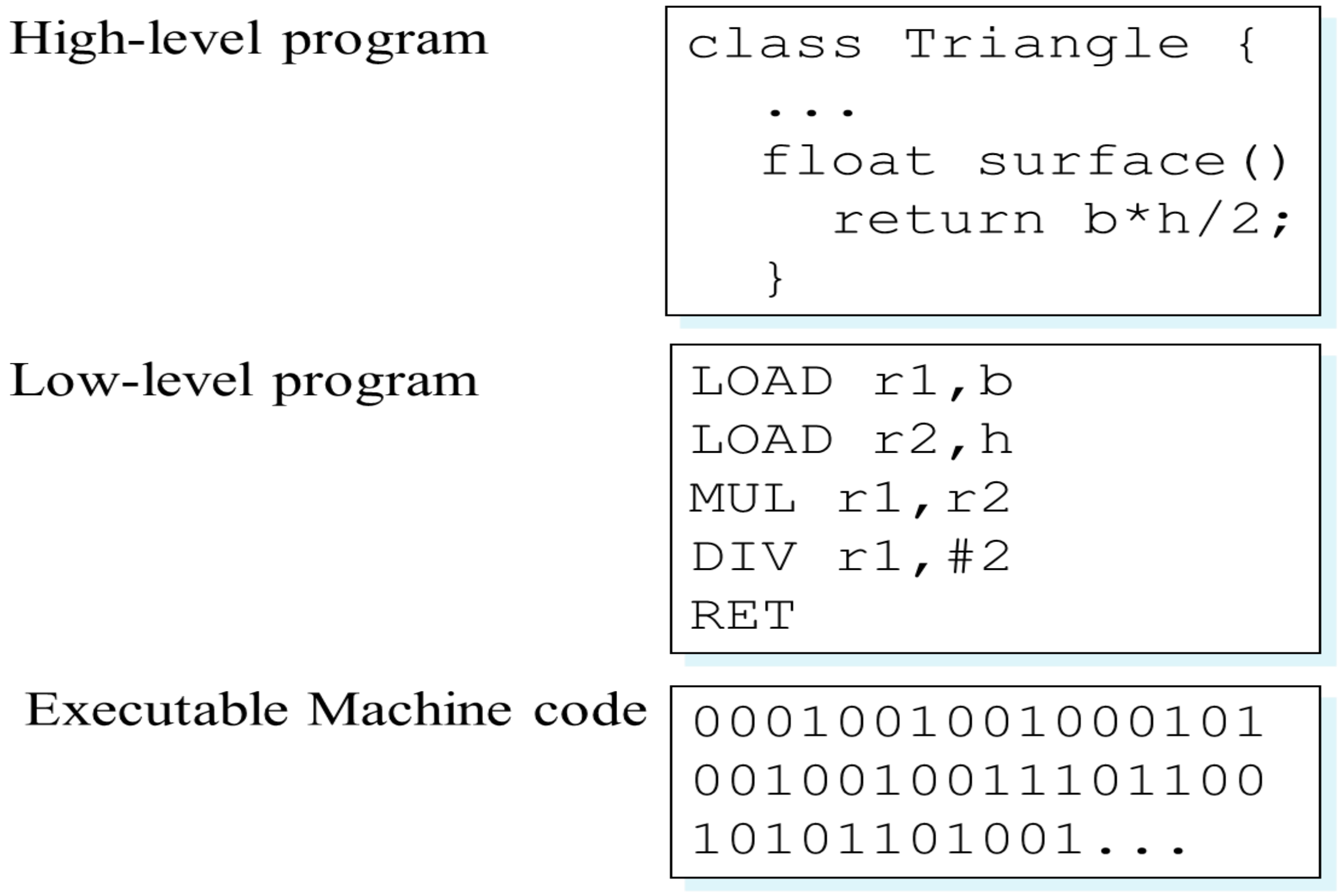
- Human languages are known as natural languages.
  - Unfortunately, computers can not understand natural languages (English, Gujarati, Spanish,.. etc),
  - as a result we must communicate with computers using computer languages (programming languages)

- Programming languages can be used to create programs that control the behavior of a computer and serve any purpose
  - A programming language is a set of rules that provides a way of telling a computer what operations to perform.

Introduction

- English is a natural language. It has words, symbols and grammatical rules.
- A programming language also has words, symbols and rules of grammar.
- The grammatical rules are called syntax.
- Each programming language has a different set of syntax rules.

Levels of Programming Languages



High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

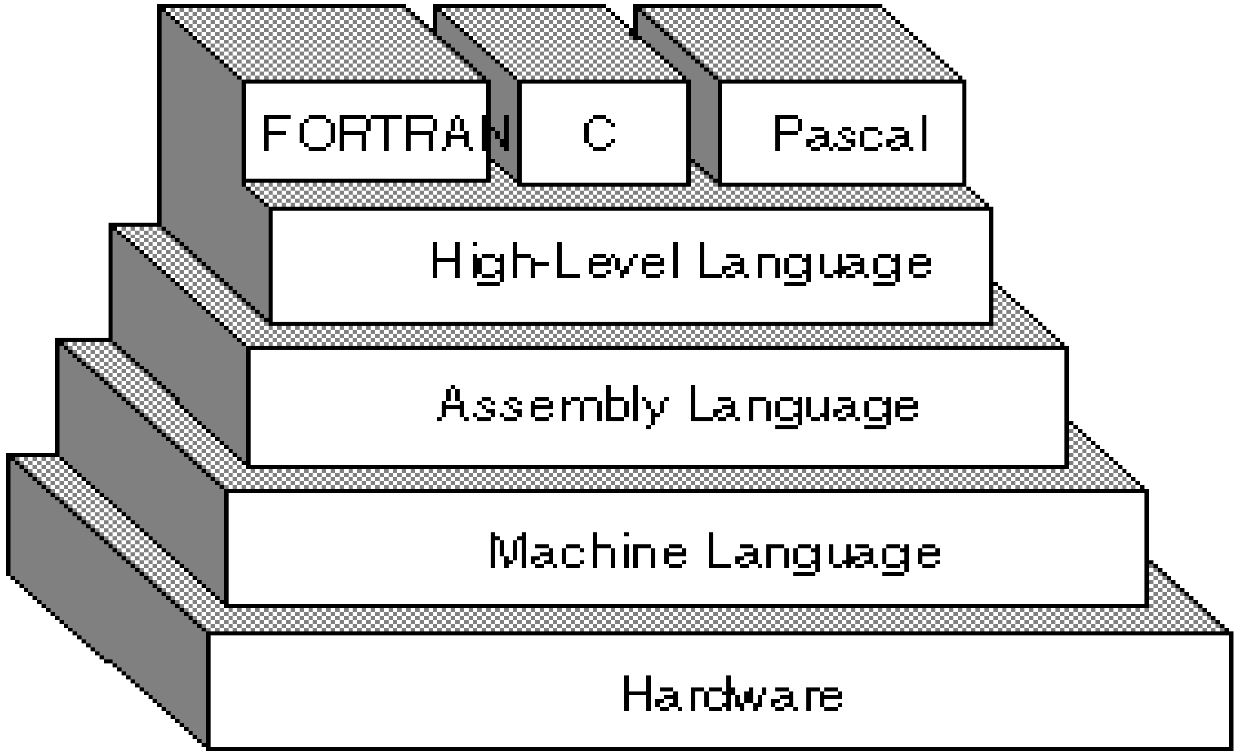
```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

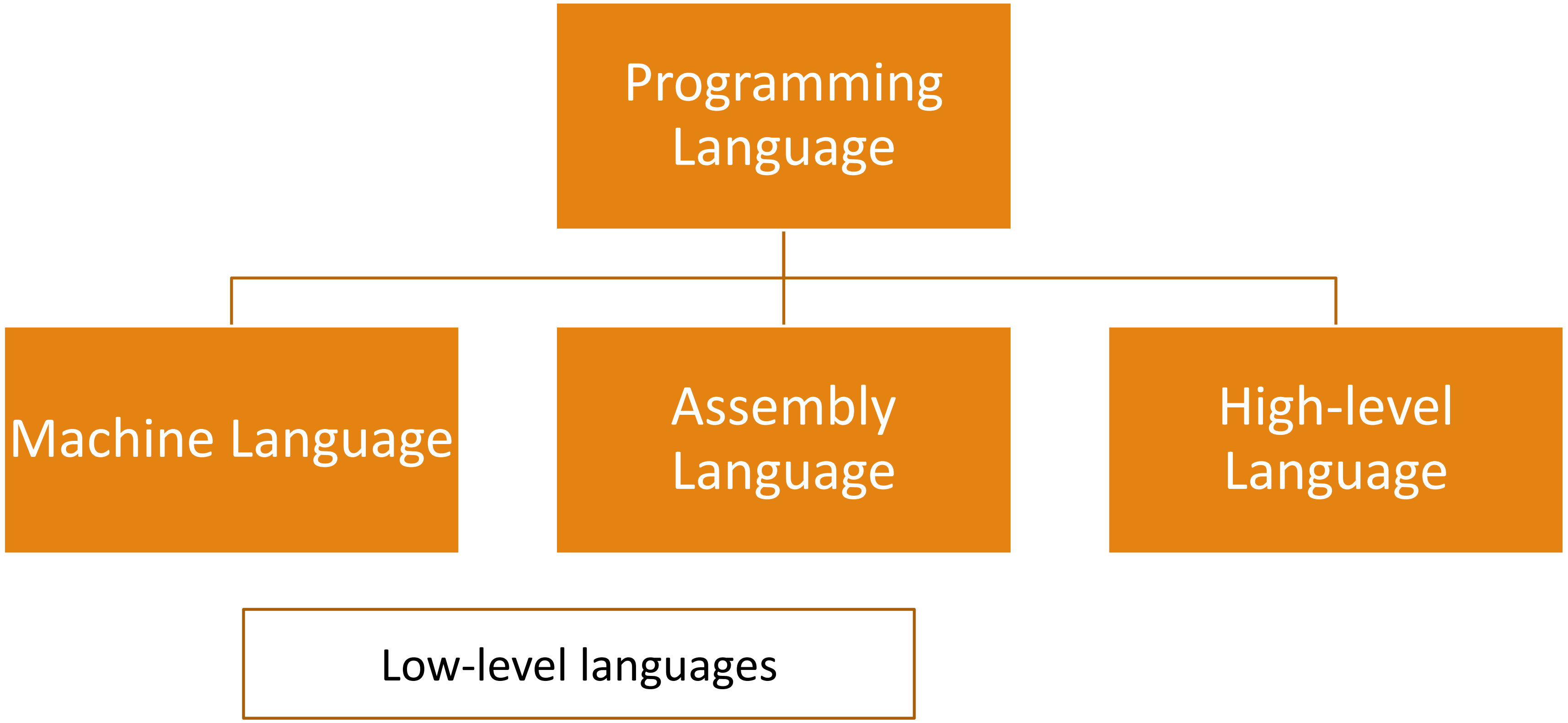
```
0001001001000101
0010010011101100
10101101001...
```

Architecture

- Computer understand only binary language (0 or 1) .
- Binary language also known as machine or low level language
- All the instructions given in binary form only – hard to understand by people
  - High-level language were developed



Classification of Programming Languages



Generations of Programming Languages

- Computer languages has the same history as the computers itself history,

- There are five generations of languages when programming method and techniques could be developed as far as development in hardware occurred.

First Generation (1GL) – Machine Language

- First-generation language was machine language
  - the level of instructions and data that the processor is actually given to work on **binary numbers 0s and 1s.**

- In the 1940s and 1950s, computers were programmed by scientists sitting before control panels equipped with toggle switches so that they could input instructions as strings of zeros and ones.

(1GL) – Machine Language

- Machine language Format
  - Operation code – instruct computer what functions are to be performed (such as addition or subtraction).
  - Operands – instruct the computer where to find or store the data on which the desired operation is to be performed
- Machine language is machine dependent as it is the only language the computer can understand.
  - Very efficient code but very difficult to write.

0101	**00111**	**11110**
Opcode	**Operand1**	**Operand2**

(1GL) – Machine Language

- **Advantages:**

1. Translation Free:
   - computer can directly execute without the need for conversion
2. High Speed:
   - Since no conversion is needed, applications developed using machine languages are extremely fast

(1GL) – Machine Language

- **Disadvantages:**

1. **Machine Dependent**
   - Based on computer architecture,
   - application developed for one type of computer may not run on others
2. **Complex Language**
   - Difficult to read and write
3. **Error Prone**
   - Since programmer has to remember all the opcode and the memory locations, it is bound to be error prone
4. **Tedious**
   - Programming becomes too complex to modify

Second Generation (2GL) – Assembly Language

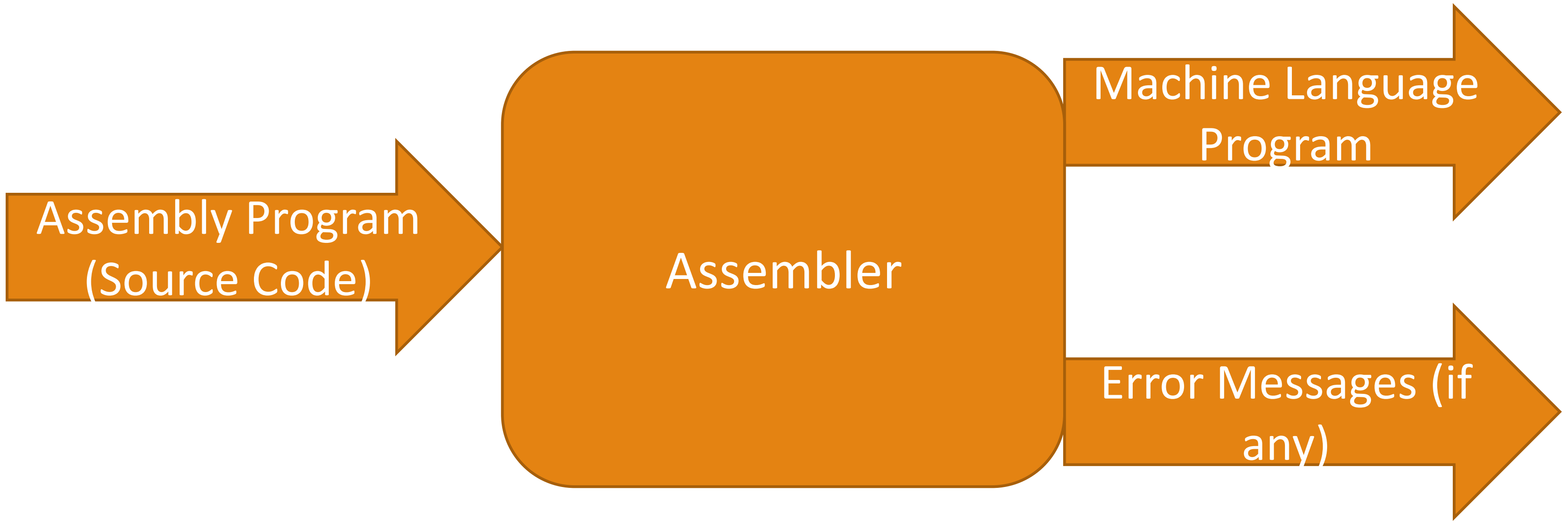
- By the late 1950s, this language had become popular.
- Known as Symbolic language
- Assembly language consists of letters of the alphabet.
  - This makes programming much easier than trying to program a series of zeros and ones.
- An assembler converts the assembler language statements into machine language

Second Generation (2GL) – Assembly Language

- The general format of Assembly Language:

Label	Opcode	Operands	Comment
BEGIN	ADD	A, B	Add B; to A;

Second Generation (2GL) – Assembly Language



Second Generation (2GL) – Assembly Language

- **Advantages:**

- Easy to understand and Use
- Less Error Prone
- Efficiency
  - Faster compare to high-level language programs
- More control on hardware

Second Generation (2GL) – Assembly Language

- **Disadvantages:**

- Machine Dependent
  - Different computer have their own assembly languages
- Harder to learn
- Slow development time
- Less efficient
- No standardization
- No support for modern software engineering technology

Third Generation (3GL)-High-level Language

- Closer to English but included simple mathematical notation.
- Programmer do not need to know how computer works in detail.
- Programmer can write program by learning syntax of language.

```
X := X + Y ;
```

- High level language must use interpreter, compiler or translator to convert human understandable program to computer readable code (machine code).

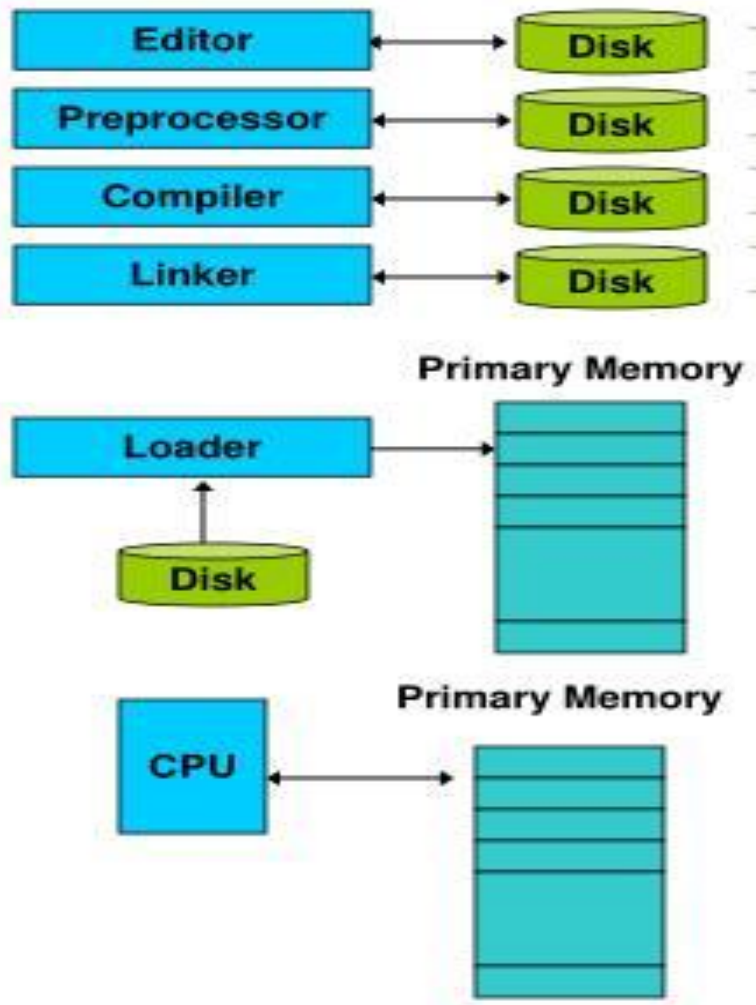
Third Generation (3GL)-High-level Language

- Many high level languages have appeared since Fortran II the most widely used have been:

COBOL	**Business applications**
FORTRAN	**Engg & Scientific Applications**
PASCAL	**General use and as a teaching tool**
C & C++	**General Purpose – currently most popular.**
PROLOG	**Artificial Intelligence**
JAVA	**General all purpose programming**

A Typical C Program Development Environment

- **Phases of C Programs:**



1. Program is created in the editor and stored on disk
2. Preprocessor program processes the code
3. Compiler creates object code and stores it on disk.
4. Linker links the object code with the libraries
5. Loader puts program in memory.
6. CPU takes each instruction and executes it, possibly storing new data values as the program executes

1. Edit

2. Preprocess

3. Compile

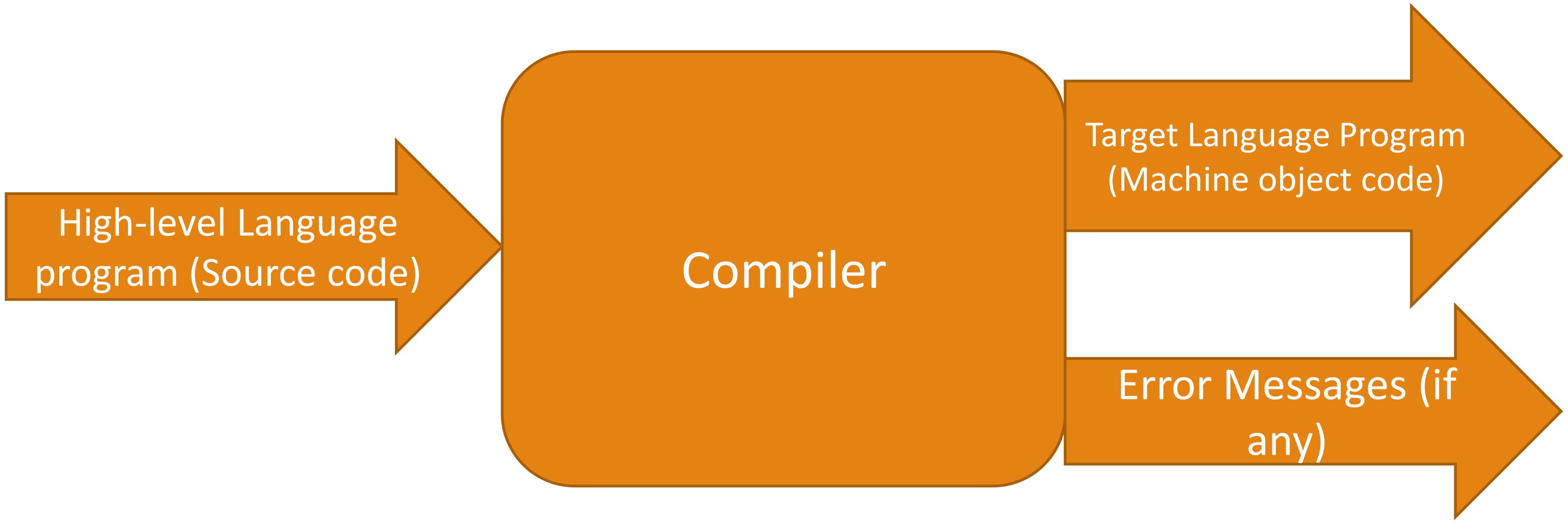
4. Link

5. Load

6. Execute

Compiler

- Language translator: convert high-level language into machine language
- Compiler replaces single high-level statement with a series of machine language instructions

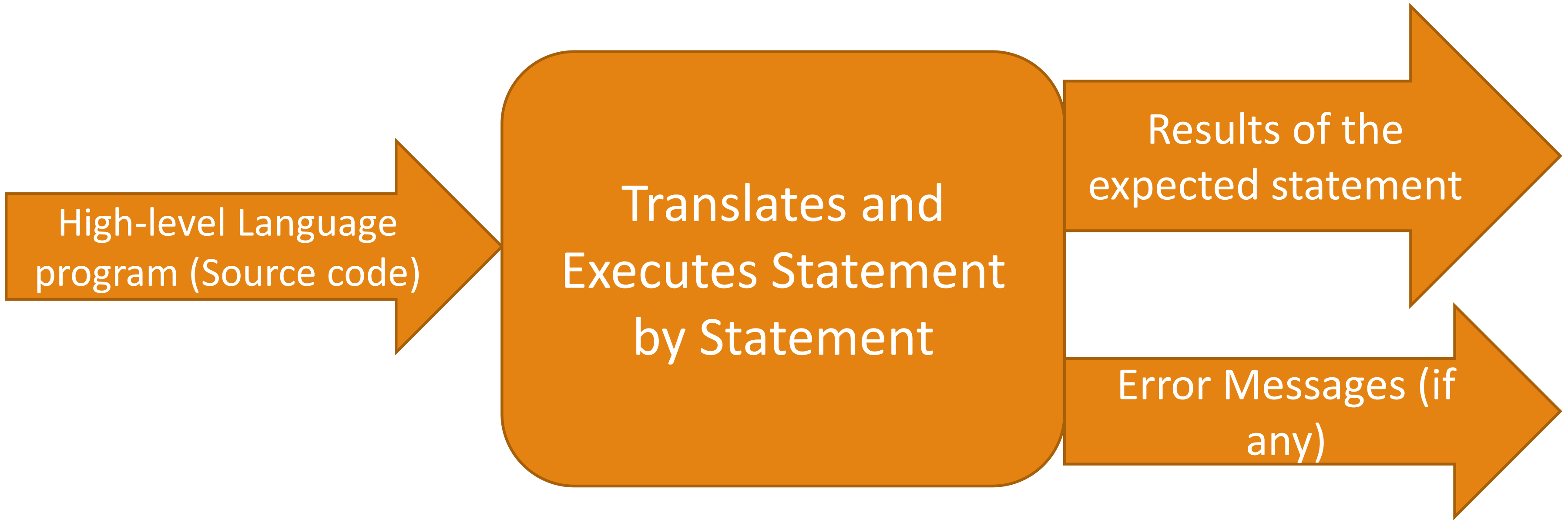

High-level Language program (Source code)
Compiler
Target Language Program (Machine object code)
Error Messages (if any)

Compiler

- Program compilation: Compiler translates whole program into an equivalent machine language program
- Once the program has been compiled, the resulting machine code (object code) saved separately, which can be run on its own at any time
- Once the object code is generated, there is no need of actual source code
- If source code is modified – necessary to recompile the program
- For each high-level language, a separate compiler is required

Interpreter

- Language translator: convert high-level language into machine language
- Unlike compiler: it translates a statement of program and executes immediately, before translating next statement

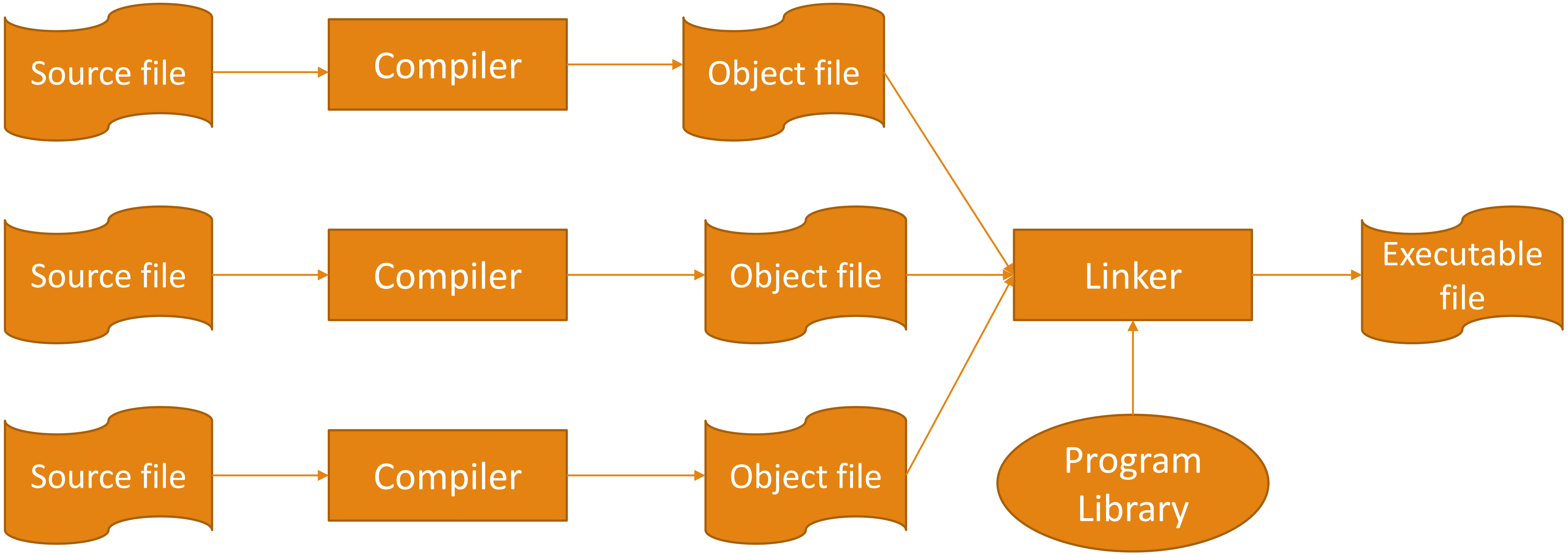

High-level Language program (Source code)
Translates and Executes Statement by Statement
Results of the expected statement
Error Messages (if any)

Linker

- An application usually consists hundreds, thousands or even million of lines of code
- Code divided into logical groups and stored into different modules
  - So debugging and maintenance becomes easy
  - Each module can be modified and compiled independently

  - Linker links several object modules and libraries to form a single program

Linker


Source file
Compiler
Object file
Source file
Compiler
Object file
Source file
Compiler
Object file
Linker
Executable file
Program Library

Loader

- Part of operating system that brings an executable file residing on the disk into the memory and starts running
- Four basic tasks of loader:

1. Allocation: allocates memory space for programs
2. Linking: combines two or more separate object programs and supplies the information needed to allow references between them
3. Relocation: prepares a program to execute properly from its storage area
4. Loading: place data and machine instructions into the memory

Loader

- Types of loader:
- Absolute Loader:
  - Loads the file into memory at the location specified by the beginning portion (header) of file and then passes control to program
  - If memory space specified by header is currently in use, execution cannot be processed
    - User must wait until requested memory becomes free
- Relocating loader:
  - Loads the program in memory, altering the various addresses as required to ensure correct referencing

Advantages of High-level Languages

- Readability
  - Easy to read, write and maintain
- Machine independent
- Easy debugging
- Easier to maintain
- Low development cost
- Easy documentation

Disadvantages of High-level Languages

- Poor control on hardware
- Less efficient
  - Process of translation increases the execution time of an application

Popular High-level Languages

- FORTRAN
- COBOL
- BASIC
- PASCAL
- C
- C++
- JAVA
- PROLOG
- LISP

Fourth Generation (4GL)

- 4GLs have simple, English-like rules, commonly used to access databases
- 4GLs are divided into three categories:

1. Query Languages: allow user to retrieve information from databases (ex. SQL)
2. Report Generators: produce customized reports using data stored into database
3. Application Generators: the user writes programs to allow data to be entered into the database

Fourth Generation (4GL)

- Advantage:
  - User can create an application in a much shorter time for development and debugging than with other programming languages

- Disadvantage:
  - Program need more disk space and large memory capacity compared to 3GL program

Fifth Generation Languages (5GL)

- Though no clear definition at present, natural language programs generally can be interpreted and executed by the computer with no other action by the user than stating their question.
- User will free from learning any programming language to communicate with computes
- Programmers may simply type the instruction or tell the computer via microphones what is needs to do
- Limited capabilities at present.