System Programming and Compiler Construction

CSC 602

Subject Incharge

Varsha Shrivastava

Assistant Professor

email: varshashrivastava@sfit.ac.in

Room No: 407

CSC 602 System Programming and Compiler Construction

Module 1

Introduction of System Software

Course Outcomes

CO1: Identify the relevance of different system programs and also distinguish different loaders and linkers, their contribution in developing efficient user applications.

CO2: Analyze the various data structures and passes of assembler design

CO3: Identify the need for different features and designing of macros.

CO4: Design Lexical Analyzer of a grammar.

CO5: Construct different parsers for given context free grammars.

CO6: Justify the need synthesis phase to produce object code optimized in terms of high execution speed and less memory usage

Text books to refer-

1. J J Donovan: Systems Programming Tata McGraw Hill Publishing Company

(System Programming).

2. A. V. Aho, R. Shethi and J.D. Ulman; Compilers - Principles, Techniques and Tools, Pearson Education

(Compiler Construction)

Term work Distribution

• Lab work (experiments):	15 Marks
• Assignment:	05 Marks
• Attendance :	05 Marks
TOTAL	**25 Marks**

Final University Marks Distribution

• University final exam	80 Marks
• Internal Assessment	20 Marks
• Term work	25 Marks
• Practical exam	25 Marks
TOTAL	**150 Marks**

Contents as per syllabus

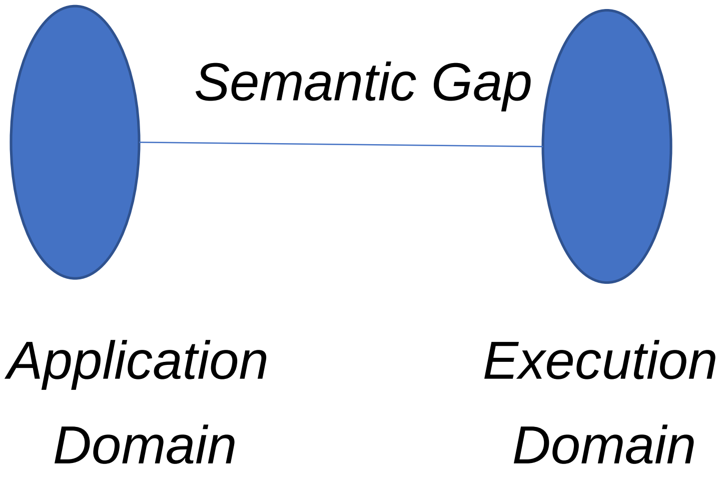
- Introduction
- Concept of System Software
- Goals of System Software
- System Program And System Programming
- Introduction to various system programs such as Assembler, Macro processor, Loader, Linker, Compiler, Interpreter, Device Drivers, Operating system, Editors, Debuggers.

Introduction to Language Processor

- A language processor is a software which bridges a specification and execution gap.
- Language processing activities arise due to:
  - The difference between software designer's idea related to behavior of software and the manner in which these ideas are implemented.

Introduction to Language Processor

- The designer expresses the ideas in terms related to the *application domain*.
- To implement these ideas in terms related to *execution domain.*
- The difference between the two domain termed as *semantic gap*.

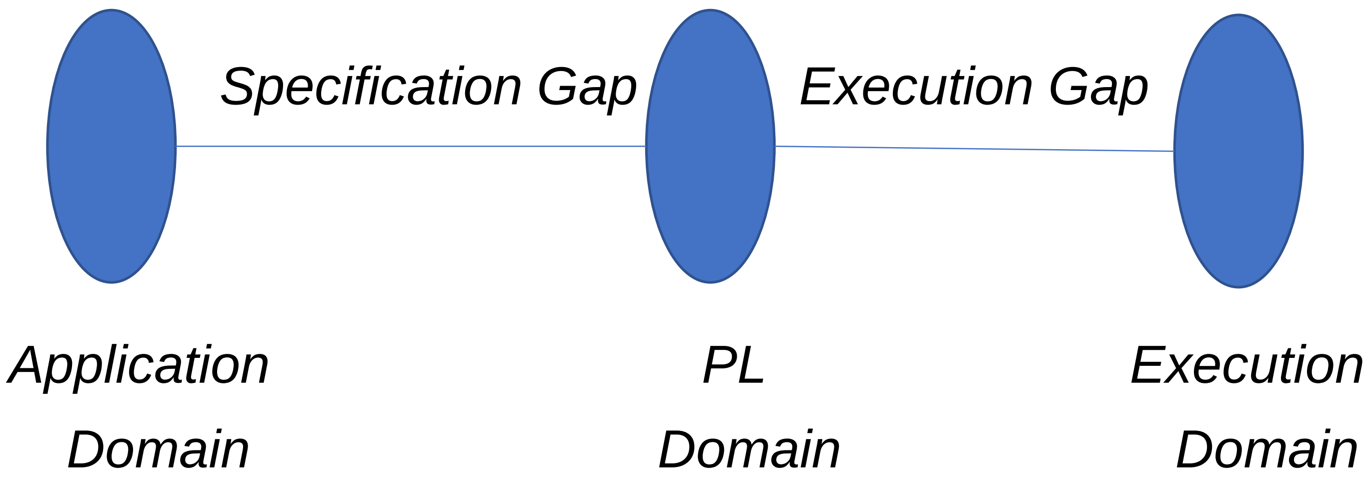


Introduction to Language Processor

- The semantic gap has many difficulties, some of the important ones being large development time and efforts, and poor quality of software.
- These issues are tackled through the use of programming language (PL).
- Software implementation using PL introduces a new domain as **PL domain**.

Introduction to Language Processor

- Now the semantic gap is bridged by the software engineering steps.
- The first step bridges gap between application domain and PL domain known as **specification gap**.
- While the second step bridges the gap between PL domain and execution domain as **execution gap**.



Introduction to Language Processor

- The specification gap is bridge by the software development team.
- While the execution gap is bridged by the designer of the programming *language processor.* Like compiler or interpreter.
- PL domain reduces the difficulties of semantic gap mentioned earlier.
- The language processor also provides a diagnostics capability which detects and indicates errors in its input. This helps in improving the quality of the software.

Introduction to Language Processor

A range of LP is defined to meet practical requirements.

1. A *language translator* bridges an execution gap to the machine language like assembler and compiler.
2. A *detranslator* bridges the same as the language translator, but in the reverse manner.
3. A *preprocessor* is a language processor which bridges an execution gap but not translator.
4. A *language migrator* bridges specification gap between two PLs.

Language Processing Activities

The Language Processing activities can be divided into those that bridge the specification gap and those that bridge the execution gap.

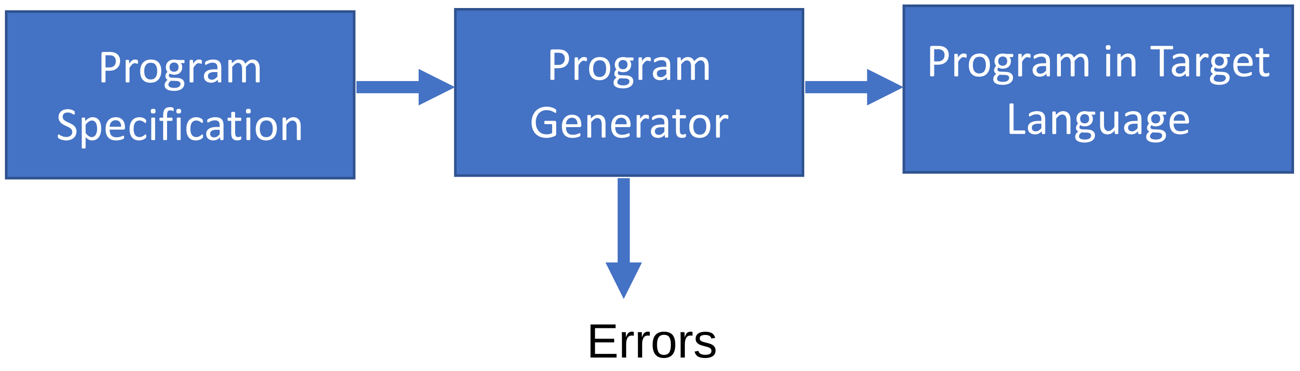
There are two types of such activities :

1. *Program Generation Activities*
2. *Program Execution Activities*

Language Processing Activities

1. *Program Generation Activities*

Generates the target program in machine language by accepting the specification of the source program



Language Processing Activities

2. Program Execution Activities

Program execution is divided into 2 parts:

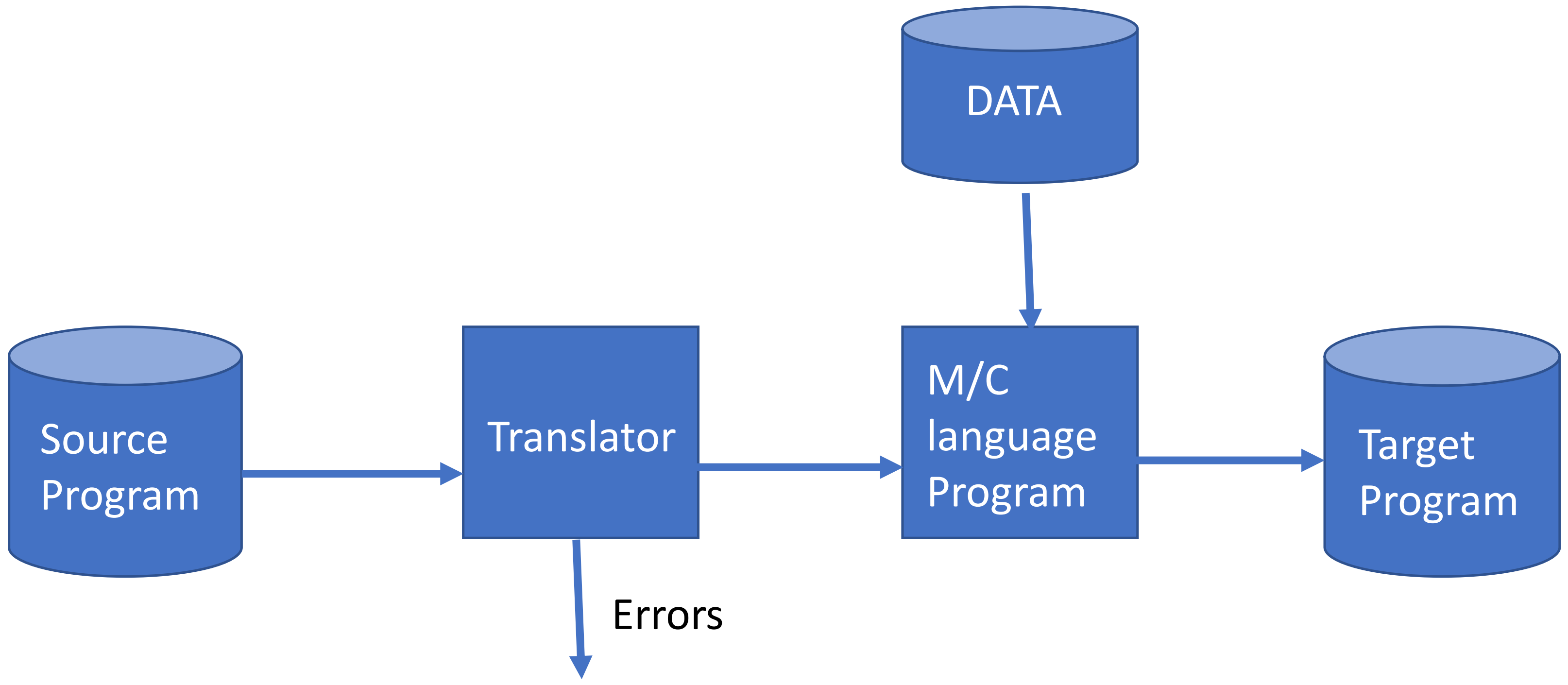
(a) Program Translation

(b) Program Interpretation

Language Processing Activities

2. *Program Execution Activities*

(a) Program Translation



Language Processing Activities

2. Program Execution Activities

(a) Program Translation :

This activity is used to convert source program into machine code.

Features of the program translation:-

- Translation of program should be done before it is executed.
- Translation of program should be saved for repetitive execution.
- Retranslation of the program should be done if there are any changes

Language Processing Activities

2. Program Execution Activities

(b) Program Interpretation :

Interpretation of program consists of following steps:

- The code is fetched.
- Analyze the code.
- Execute the meaning of the code.

Introduction

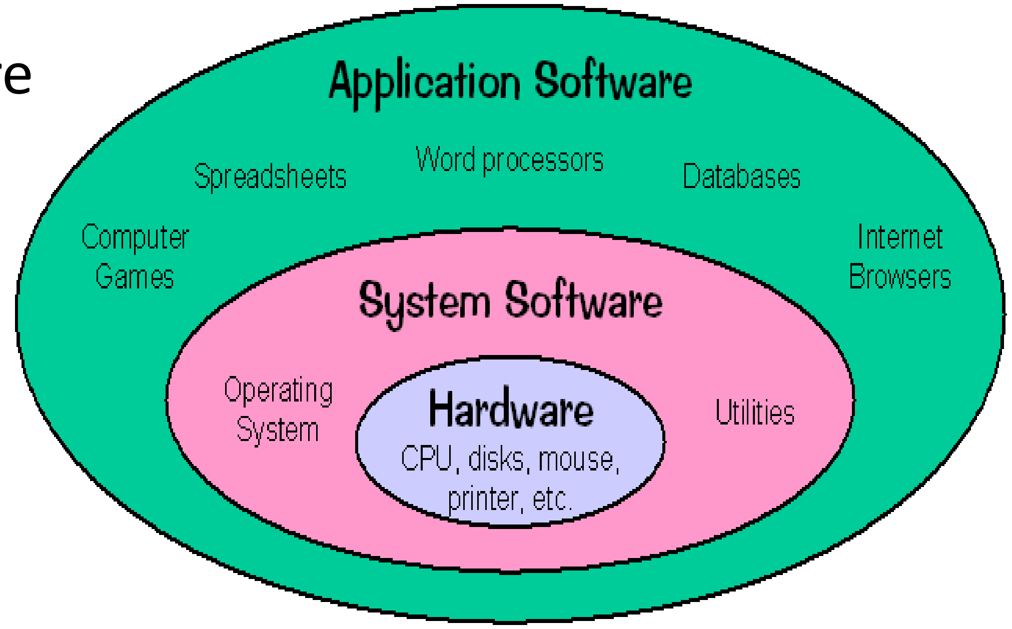
What is System Program?

A program which directly interacts with the hardware.

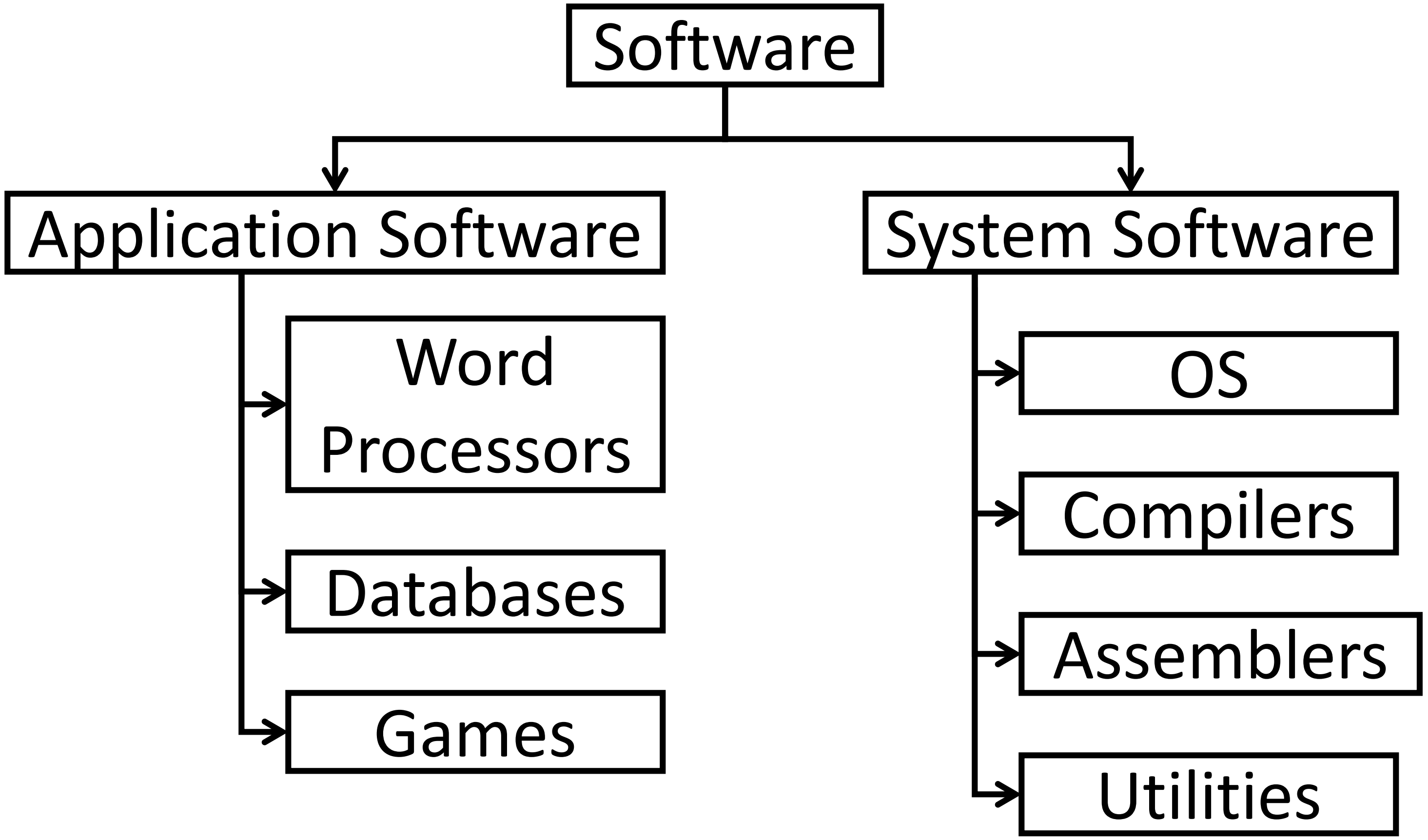
Introduction to System Software

- Software - the programs and other operating information used by a computer.

- 2 types :
  - Application Software
  - System Software



Software's


Software
Application Software
System Software
Word Processors
Databases
Games
OS
Compilers
Assemblers
Utilities

Application Software

- Application software is a set of one or more programs designed to carry out operations for a specific application.

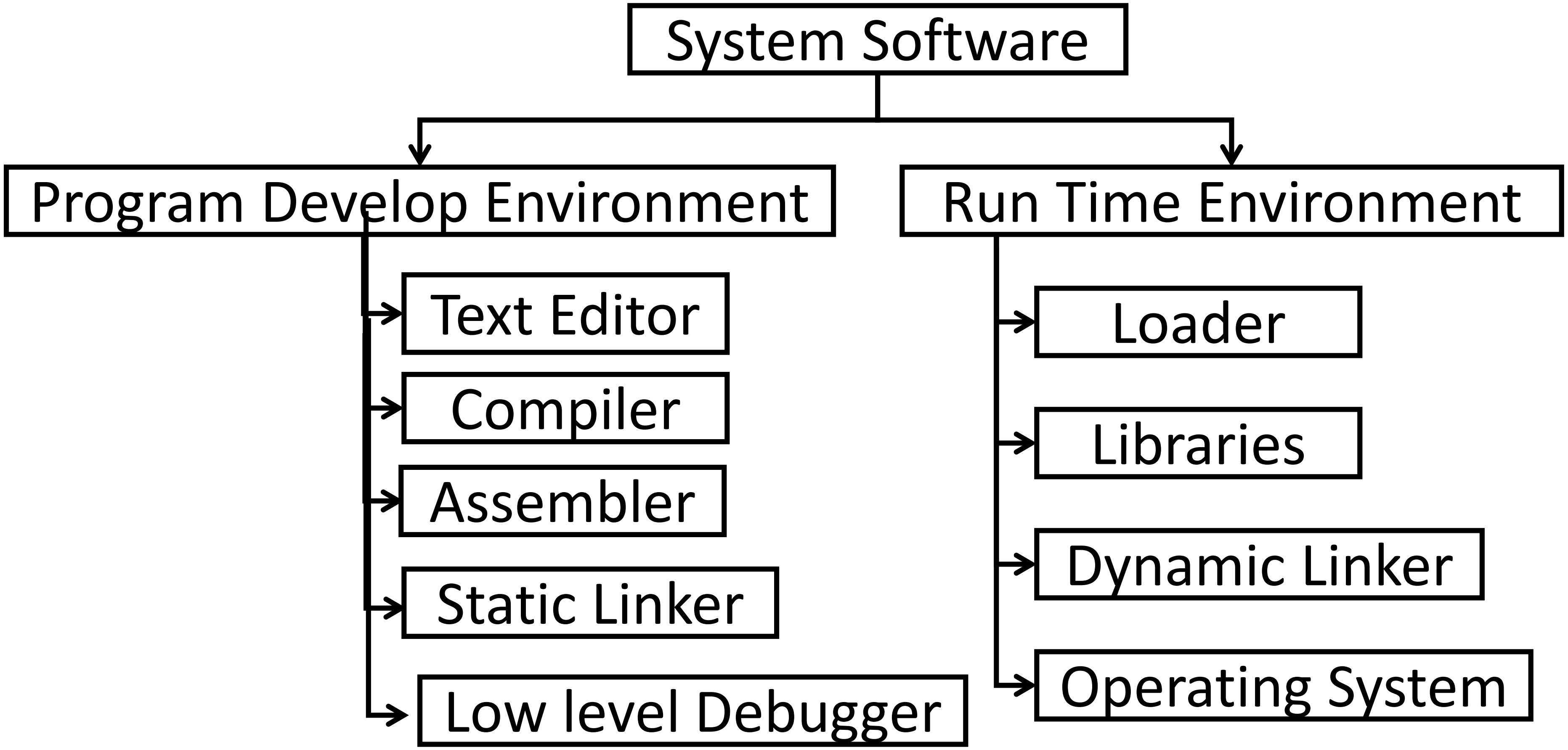
- Application software cannot run on itself but is dependent on system software to execute.

- Example : games, word processors, image processors, etc.

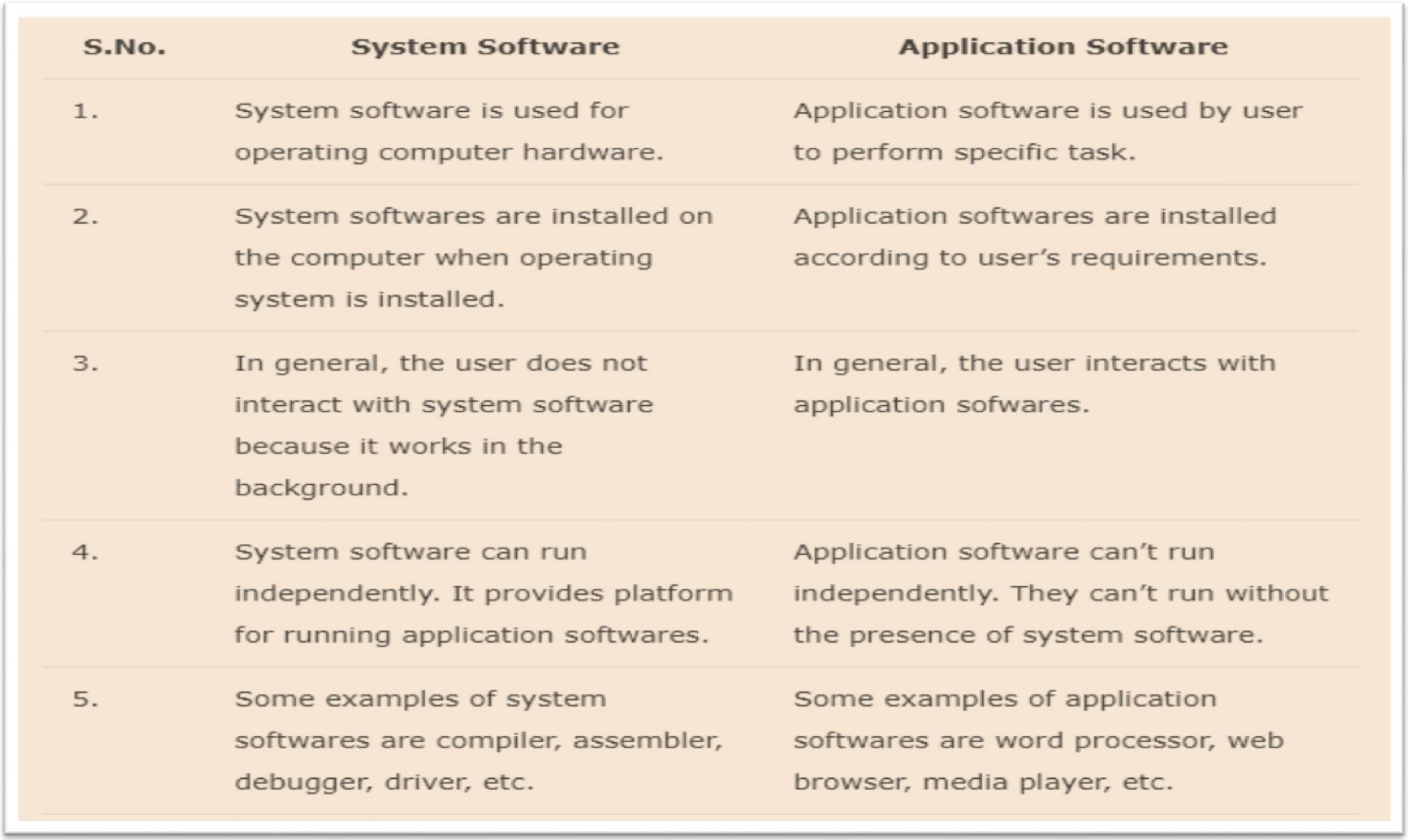
System Software

- System software is computer software designed to operate and control the computer hardware.
- It is used to provide a platform for running application software.
- Example : compilers, assemblers, utilities, etc.

System Software



System Software vs Application Software



S.No.	System Software	Application Software
1.	System software is used for operating computer hardware.	Application software is used by user to perform specific task.
2.	System softwares are installed on the computer when operating system is installed.	Application softwares are installed according to user's requirements.
3.	In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application sofwares.
4.	System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
5.	Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application softwares are word processor, web browser, media player, etc.

System Programs

- Operating Systems
- Device Drivers
- Compilers
  - Preprocessor
  - Interpreters
- Macro Processors
- Assemblers
- Linkers
- Loaders

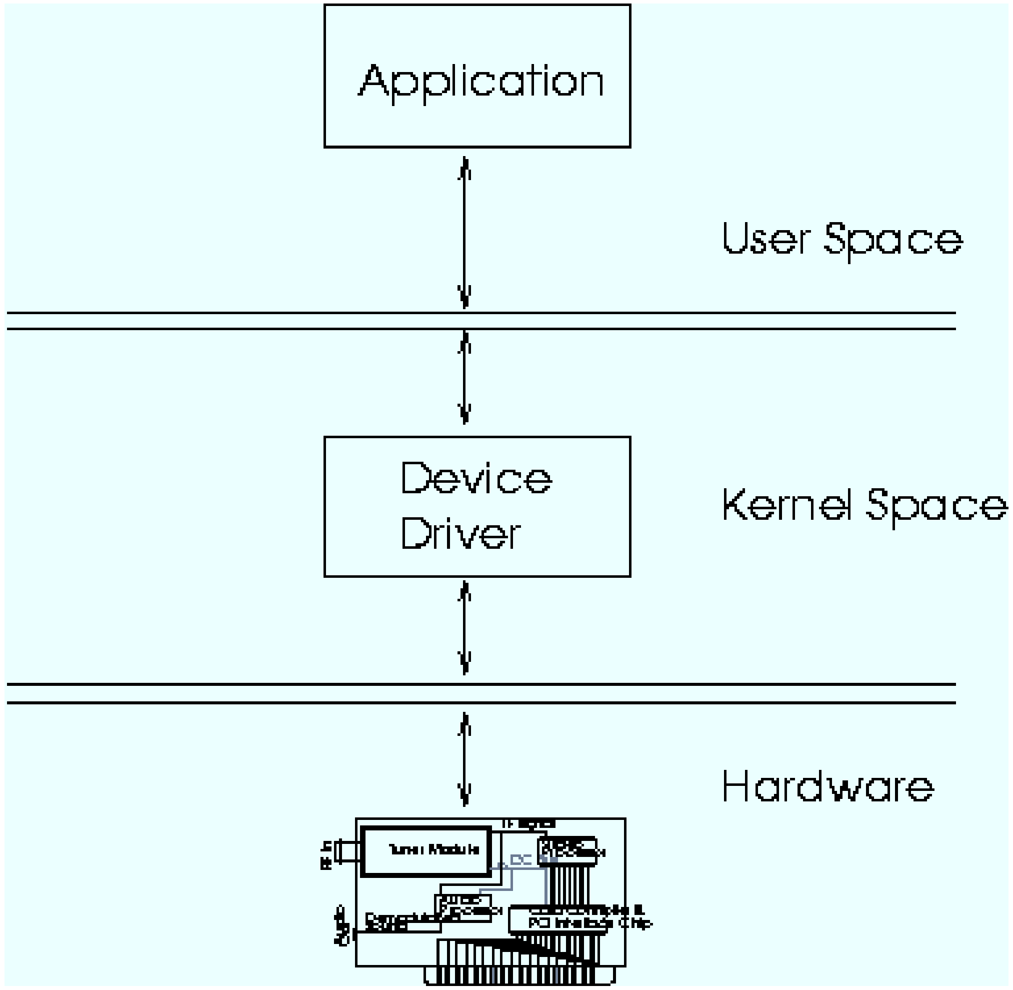
Operating System

- An operating system (OS) is software that manages computer hardware and software resources and provides common services for computer programs.
- The operating system is an essential component of the system software in a computer system.
- Application programs usually require an operating system to function.

Device Drivers

- A device driver is a program that controls a particular type of device that is attached to your computer.
- There are device drivers for printers, displays, CD-ROM readers, diskette drives, and so on.
- When you buy an operating system, many device drivers are built into the product.

OS and Device drivers



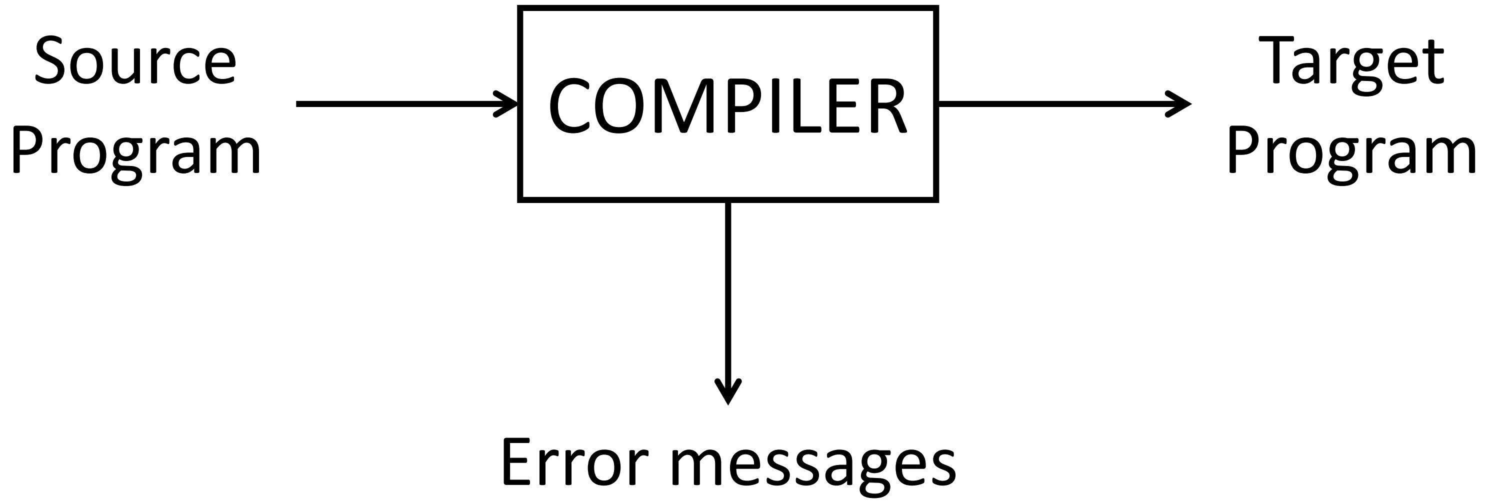
Compilers

- Preprocessors
  - A tool that produces input for compilers.
  - Deals with macro-processing, augmentation, file inclusion, language extension, etc.

- Interpreters
  - Similar to compilers (translates high-level language into low-level machine language)
  - An interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence.
  - If an error occurs, an interpreter stops execution and reports it.

Compilers

- Compiler is a program (or set of programs) that **translates** a source code (high level language) into machine understandable code (low level language).



Compilers

- A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code and executes the whole program
- May involve many passes.
- A compiler reads the whole program even if it encounters several errors.

Interpreter vs Compiler

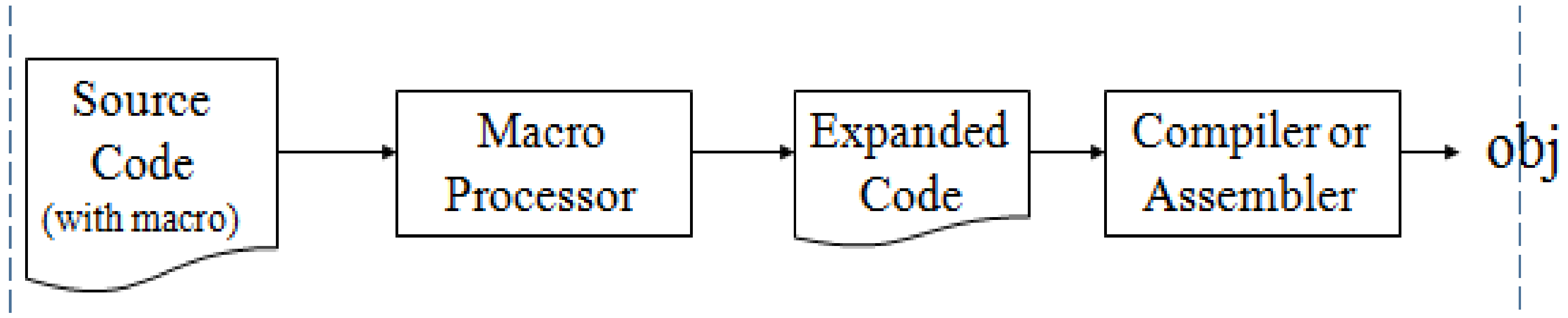
Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Macro Processors

- A macro instruction is a notational convenience for the programmer.
- It allows the programmer to write shorthand version of a program (module programming).
- The macro processor replaces each macro invocation with the corresponding sequence of statements (expanding).

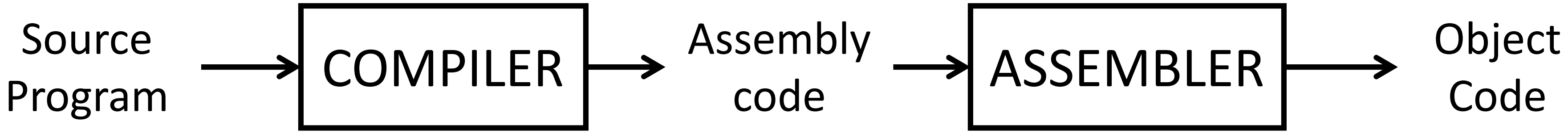
Macro Processors

- A macro processor can –
  - Recognize macro definitions
  - Save the macro definition
  - Recognize macro calls
  - Expand macro calls



Assemblers

- An assembler translates assembly language programs into machine code.
- The output of an assembler is called an object file.
  - The object file contains the data required to place these instructions in memory and information to enable loader to prepare program for execution.

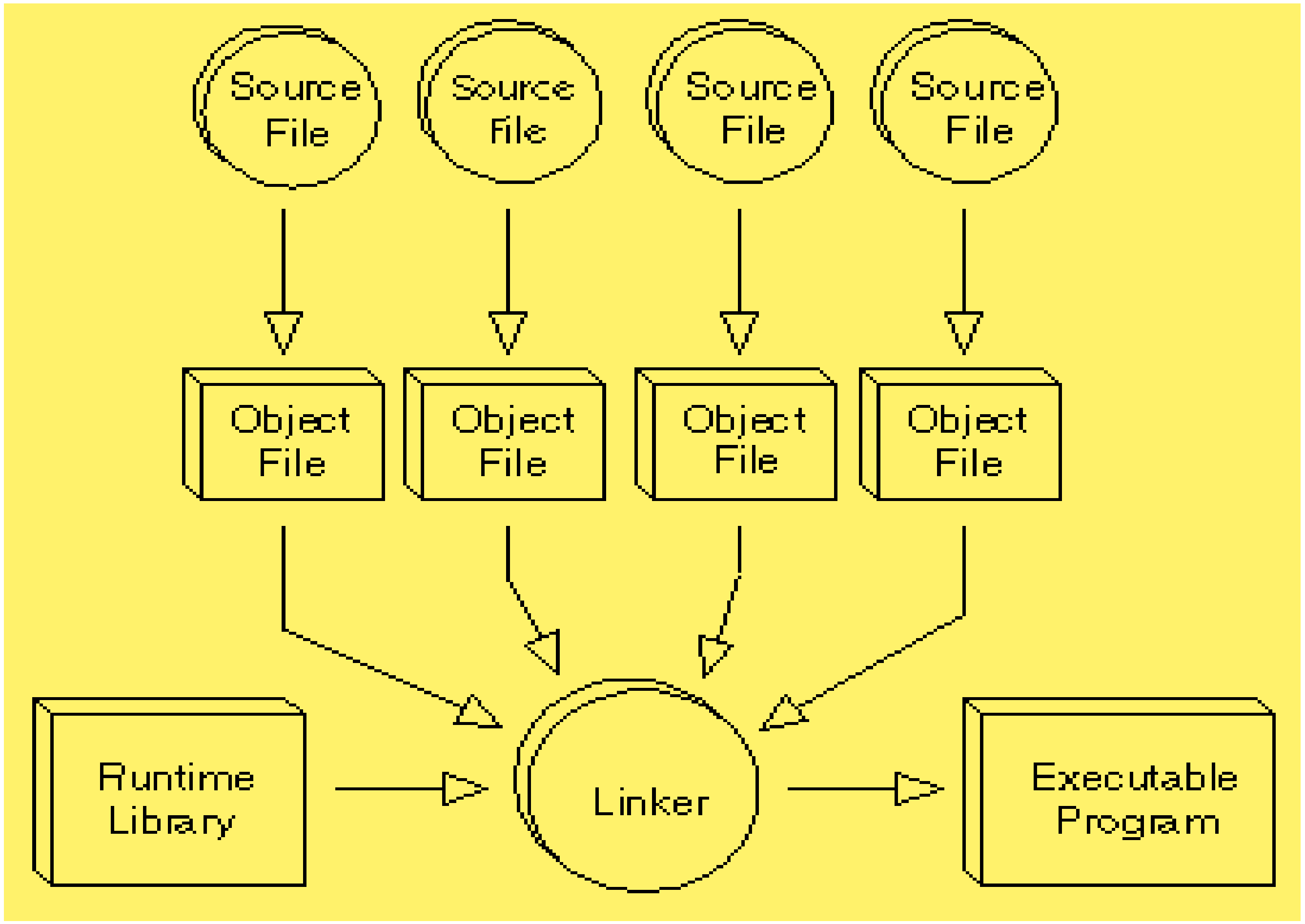


Linkers

- Linker is a computer program that links and merges various object files together in order to make an executable file.

- The major task of a linker is to search and locate referenced modules in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

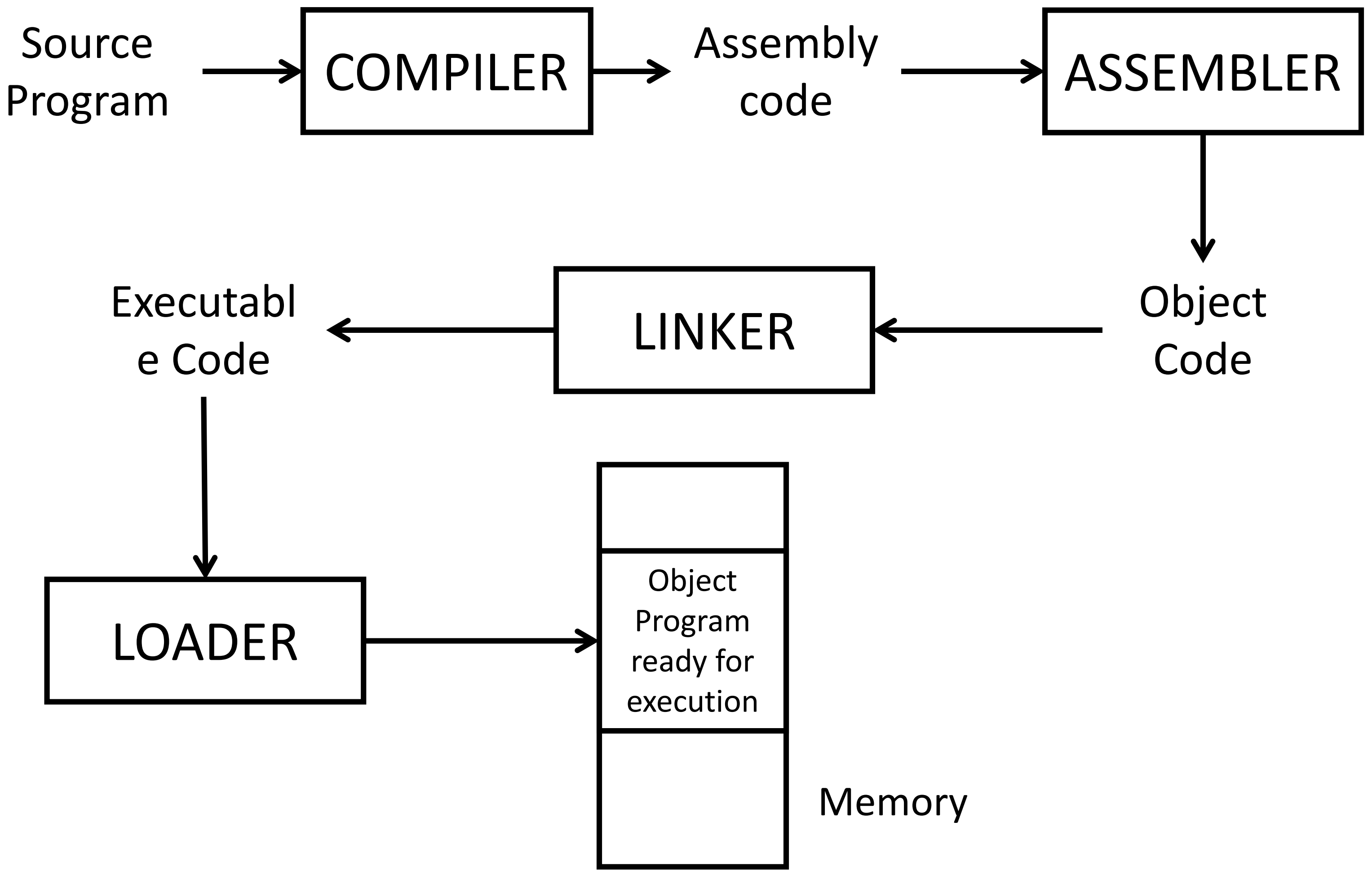
Linkers


Source File
Source File
Source File
Source File
Object File
Object File
Object File
Object File
Runtime Library
Linker
Executable Program

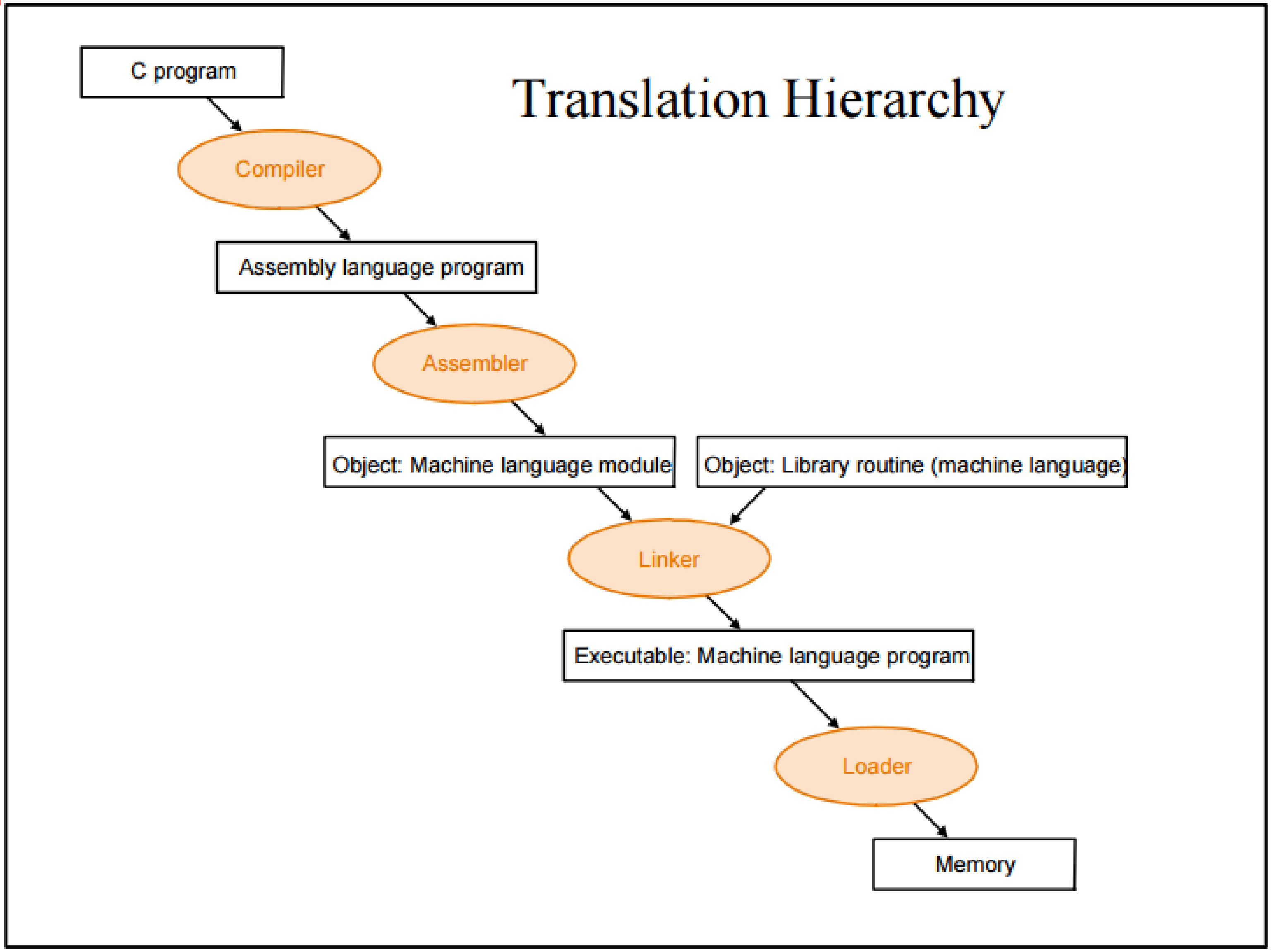
Loaders

- Loader is a part of operating system and is responsible for loading executable files into memory and execute them.
- It calculates the size of a program (instructions and data) and creates memory space for it.
- It initializes various registers to initiate execution.

Stages


Source Program
COMPILER
Assembly code
ASSEMBLER
Object Code
LINKER
Executabl e Code
LOADER
Object Program ready for execution
Memory

Compilation steps


Translation Hierarchy
C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

University Questions

- What is System Programming? List some system programs and write their functions.
- Difference between Application and system Software.
- Differentiate between application program and system program . Indicate the order in which following system programs are used, from developing programs up to its execution.

  Assembler, loader,linker,macroprocessor,compiler, editor
- What is Language Processing? Explain Application, PL & Execution domain.

THE END!

Have a nice day!