Introduction To System Software

MODULE 1

CONCEPT

- The software designer describes the ideas concerning the behavior of the software – in terms related to the application domain.
- This description has to be interpreted in the terms related to the execution domain.
- Semantic gap – difference in the rules of meaning of two domains

CONCEPT

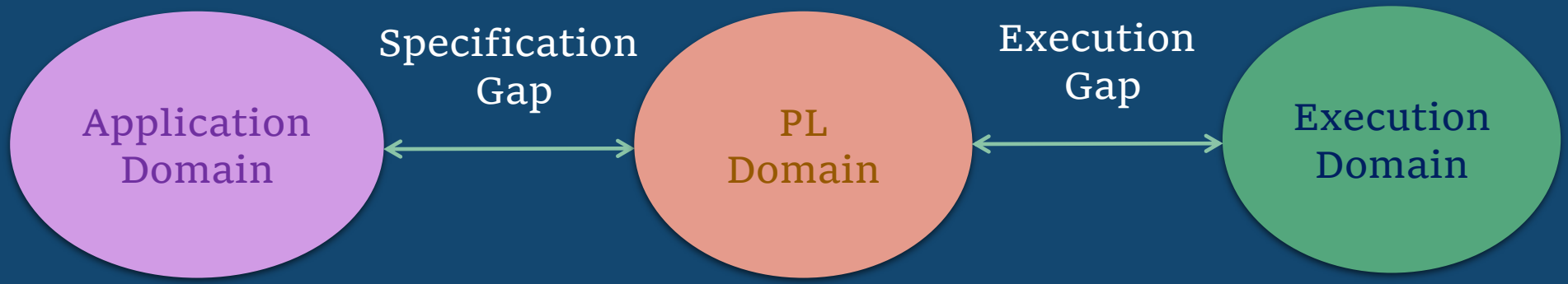
Consequences due to semantic gap

- Large development times
- Large development efforts
- Poor quality software

To bridge the gap between AD and ED, the Software development steps

- Specification, design, coding
- PL implementation

CONCEPT


Application Domain
Specification Gap
PL Domain
Execution Gap
Execution Domain

PROGRAMMING LANGUAGE DOMAIN

- Languages are used for communication
- Natural Mode of Communication: **Human Languages**
- To complete task from Machine: Use program or Request to machine
- Programs are written in: **Programming Languages**
- Generation of Languages:
    - First Generation Language (Machine Language)
    - Second Generation Language (Assembly Language)
    - Third Generation Language (High Level Language)

GENERATION OF LANGUAGES

1ST GENERATION

- Also known as Machine Language
- Binary Language – Uses '0' and '1'
- Easy Understandable instructions to computers
- Used for internal structure of program
- Advantage
  - Increase speed of Processing
  - No need of translation
- Disadvantage
  - Difficult to Learn
  - Error Prone

2ND GENERATION

- Also known as Assembly Language
- Specific Meaningful words Mnemonics
- Not easily understood by machine
- Program need to be translated into machine language
- The translator is known as 'Assembler'

3RD GENERATION

- Also known as High Level Language
- Syntax uses English keywords which are easy to understand
- Languages are not machine oriented
- Program need to be translated into machine language
- Slow processing in comparison with 1st and 2nd Generation Languages
- The translator is known as 'Compiler'
- Example: C, C++, Java

LANGUAGE PROCESSOR

- A language processor is software which bridges a specification or execution gap.
- A spectrum of language processors is designed to meet practical requirements
- Language Processing Activities
  - Program Generation: Bridges Specification Gap
  - Program Execution: Bridges Execution Gap

GOALS OF SYSTEM SOFTWARE

1. User Convenience

Facet	Example
Fulfilment of Necessity	Ability to execute the program Use the File System
Good Service	Speedy response to Computational Request
User friendly Interfaces	Easy-to-use Command Graphical User Interfaces (GUI)
New Programming Model	Concurrent Programming
Web – oriented Features	Means to set up web enabled servers
Evolution	Add New Features Use new Computer Technologies

GOALS OF SYSTEM SOFTWARE

2. Efficient Use

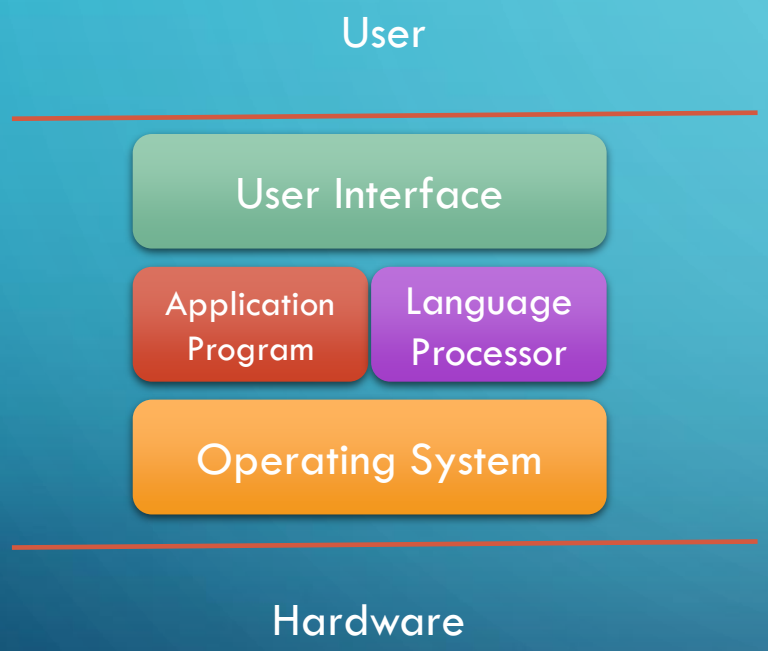
- System software must efficiently use fundamental computer resources like CPU, Memory, Disks and other I/O Devices
- Poor efficiency can occur if a program does not use the resource allocated to it which further results in snowballing effect.
- *Snowballing Effect:* If the resource is allocated to a user, it is denied to other programs that need it. These programs can't execute and hence the resources allocated to them also remains idle
- To achieve efficiency, the system software must minimize the waste of resources by programs and its own overhead.

GOALS OF SYSTEM SOFTWARE

3. Non-Interference

The system software must ensure that no person can illegally use programs and resources in the system or interfere with them

System Software



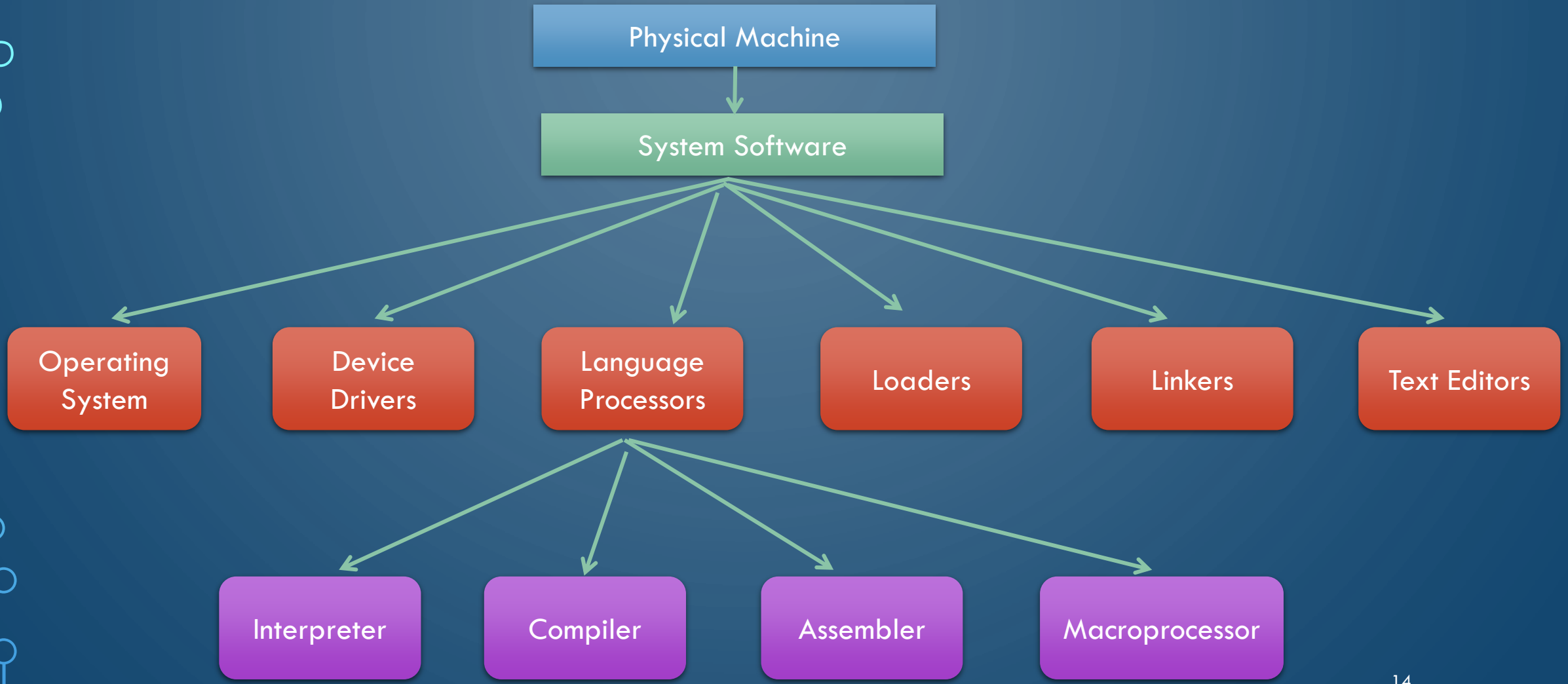
- System software is computer software designed to operate and control the computer hardware.
- It is used to provide a platform for running application software.
- Example : compilers, assemblers, utilities, etc.

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 PROGRAM AND SYSTEM PROGRAMMING

- System Sofware: It is a collection of programs
- System Program: Each Program in the collection of system software
- Design Goals of System Programs:
  - ✓The program should function correctly under all conditions
  - ✓The program should be effective in its computing environment
  - ✓The program should be portable
  - ✓The program should be able to evolve to provide new functionalities and adapt to new technologies
- System Programming: It is the set of techniques used to realize the design goals of system program

SYSTEM SOFTWARE



OPERATING SYSTEM

- Mediator between User Programs and Hardware
- Allocation of Resources and Services
- It includes
  - Process Management
  - Memory Management
  - File System Management
  - Secondary Storage Management
- Program to manage these resources: Scheduler, Traffic Controller

UTILITIES AND DEVICE DRIVERS

Utilities

- small program that provides additional capabilities to operating system
- Special but non-essential part of operating system
- Performs functions related to computer system management and maintenance
- Example: Antivirus SW, Data Compression SW, Disk Optimization SW

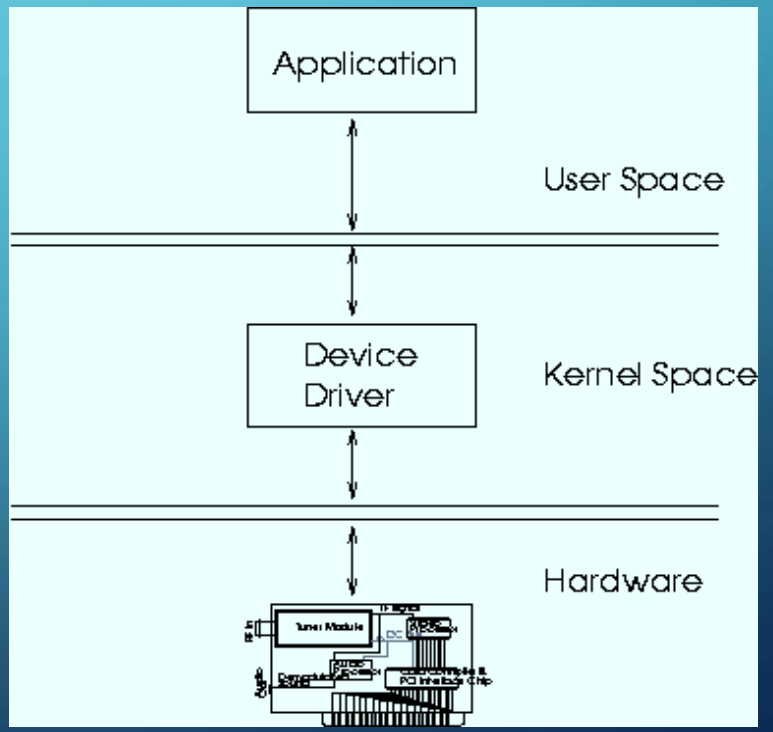
Device Drivers:

- A computer program that controls a particular device connected to computer.
- Devices includes Input Output and Storage Devices

UTILITIES AND DEVICE DRIVERS

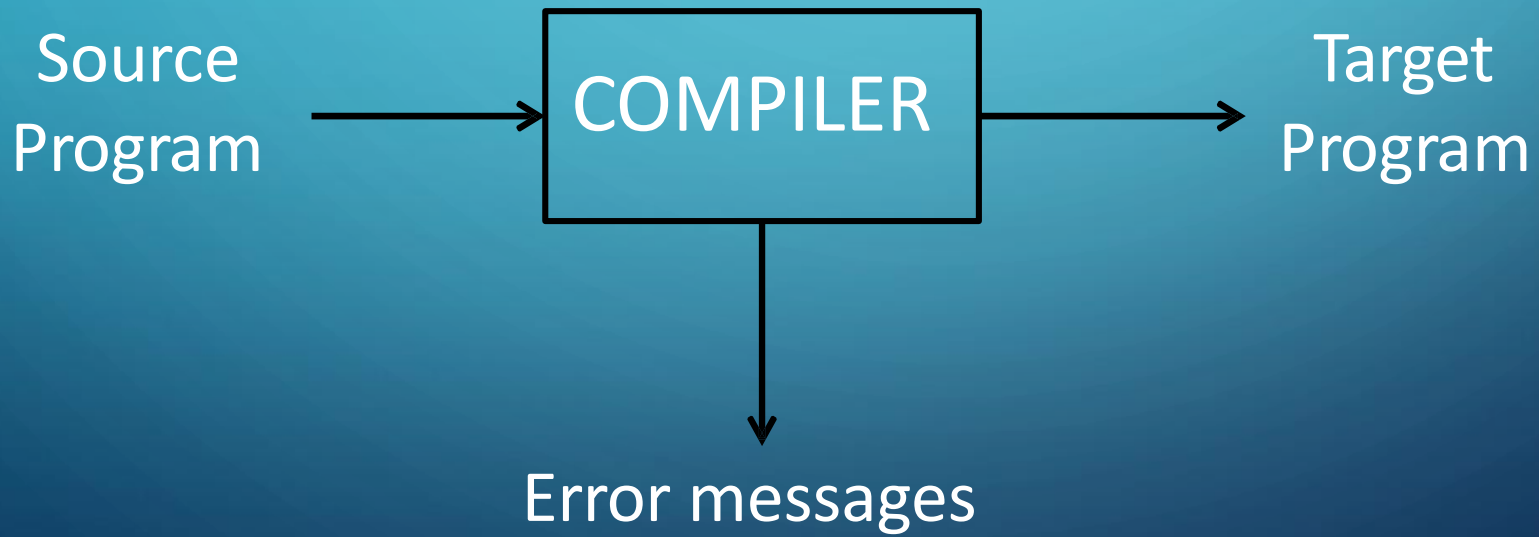
Device Drivers:

- 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.



COMPILER

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



COMPILER

- 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.

PREPROCESSORS AND INTERPRETERS

- 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.

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.

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.

LANGUAGE PROCESSORS

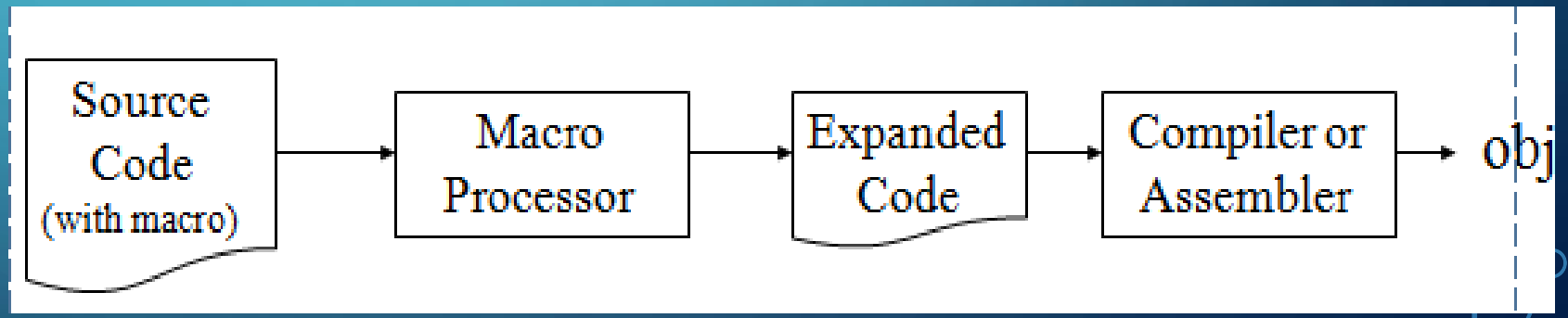
Parameter	Assembler	Compiler	Interpreter
Conversion	Assembly Language into Machine Language	Source Program to Target Program	Line by Line Conversion from HLL to Machine Language and executes line by line
Speed of Execution	Fast	Fast	Slow
Translation of	Entire Program	Entire Program	Line by Line
Mechanism for Execution	Program need to be assembled	Once compiled can be executed multiple times	For every execution, each line must be interpreted
Creation of Object File	Yes	Yes	No
Example	TASM, MASM	C Compiler, Javac	Basic Interpreter, Python Interpreter

MACRO PROCESSOR

- **Macro:** Abbreviation for small code
- **Macro Definition:** Sequence of code that has name
- **Macro Processor:** Program that substitutes and specializes macro definitions and Macro calls
- The macro processor replaces each macro invocation with the corresponding sequence of statements (expanding).

MACRO PROCESSOR

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



LINKERS AND LOADERS

Linkers

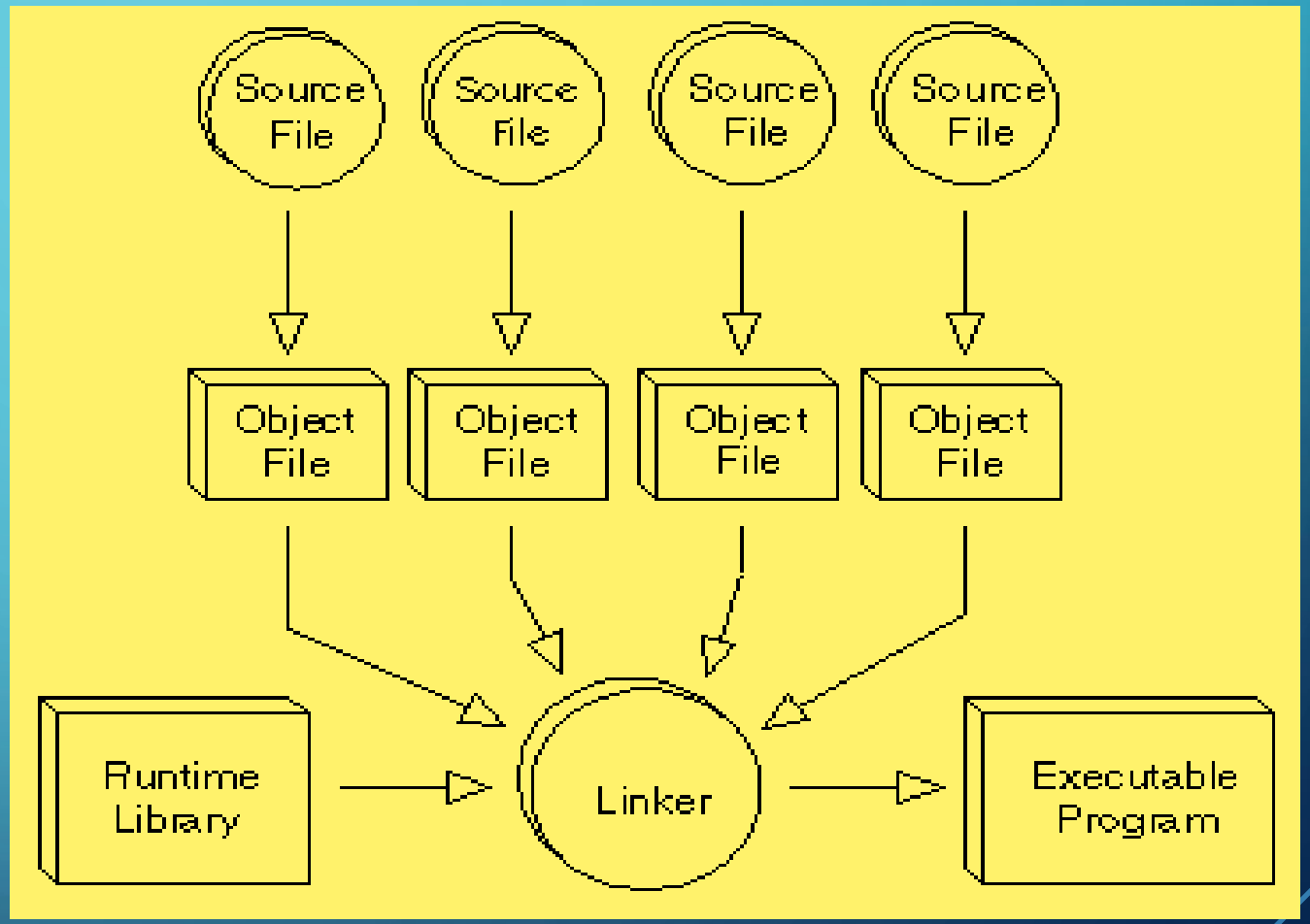
- Linker is a computer program that links and merges various object files together in order to make an executable file.
- The major task of a linker is to search and locate referenced modules in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

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.
- Schemes: Relocating, Absolute and direct linking

Linkers

LINKERS
AND
LOADERS

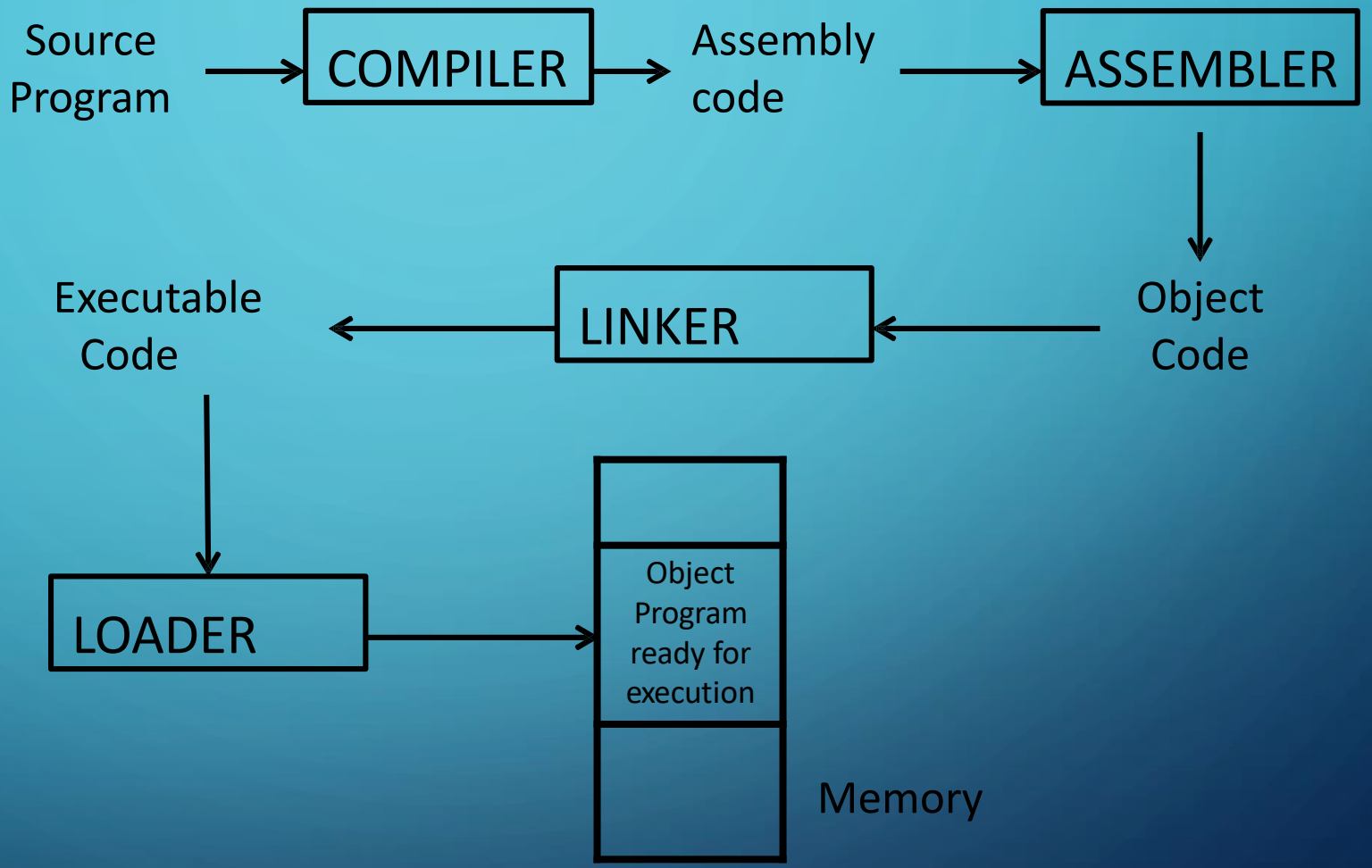


LINKERS AND LOADERS

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.
- Schemes: Relocating, Absolute and direct linking

STAGES


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

TEXT EDITORS AND DEBUGGER

Text Editor

- Compilers usually accept source programs written using an editor that will produce a standard file such as ASCII file.
- Compilers normally are bundled together with the editors and other programs into an Interactive Development Environment or IDE

.

TEXT EDITORS AND DEBUGGER

Debugger

- Debuggers are program that can be used to determine execution errors in a compiled program.
- It is also packaged with a compiler in an IDE.
- Running a program with debugger differs from straight execution.
- The debugger keeps track of most or all source code information, such as line numbers names of variable and procedures.
- It can also halt execution at pre-specified locations called breakpoints
- At breakpoint it provide information on what functions have been called and what the current values of the variables are.

SYSTEM SOFTWARE