Systems Software

INTRODUCTION	(02 Hours)
Introduction System software, Utility Software, systems programming .	
ASSEMBLER	**(04 Hours)**
Introduction, Cross Assembler, Micro Assembler, Meta Assembler, Single pass Assembler, Two Pass Assembler, Design of Operation code table, Symbol table, Literal table.	
MACRO PROCESSOR	**(04 Hours)**
Introduction of Macros, Macro processor design, Forward reference, Backward reference, positional parameters, keyword parameters, conditional assembly, Macro calls within Macros, Implementation of macros within Assembler. Designing Macro name table, Macro Definition table, Kew word parameter table, Actual parameter table, Expansion time variable storage.	

COMPILER STRUCTURE	**(02 Hours)**
Analysis-synthesis model of compilation, various phases of a compiler, tool based approach to compiler construction.	
LEXICAL AND SYNTAX ANALYSIS	**(14 Hours)**
Interface with input, parser and symbol table, token, lexeme and patterns, difficulties in lexical analysis, error reporting, and implementation. Regular definition, Transition diagrams. Context free grammars, ambiguity, associativity, precedence, top down parsing, recursive descent parsing, transformation on the grammars, predictive parsing, Bottom up parsing, operator precedence parsing, LR parsers.	
INTERMEDIATE CODE GENERATION	**(06 Hours)**
Intermediate representations, Code generation & instruction selection issues, basic blocks & flow graphs, register allocation, optimization of basic blocks, loops, global dataflow analysis.	
RUN TIME ENVIRONMENT	**(06 Hours)**
Absolute loader, Relocation - Relocating loader, Dynamic loader, Bootstrap loader, Linking-loader, Program relocatibility, Design of Absolute Loader, Design of direct-linking editor, other Loader scheme e.g. (Binders, Linking Loaders, Overlays, Dynamic Binders.	
Advanced Topics	**(04 Hours)**
Practicals will be based on the coverage of the above topics separately.	**(28 Hours)**
(Total Contact Hours: 42 + 28 = 70)	

BOOKS RECOMMENDED

1. A. V. Aho, R. Sethi and J D.Ullman, “Compilers-Principles, Techniques and Tools”, 2nd Edition, Pearson, 2006.
2. Leland L. Beck, ”System Software -An Introduction to System Programming”, 3rd Edition, Addision Wesley, reprint 2003.
3. Kenneth C. Louden, ”Compiler Construction-Principles and Practice”, 1st Edition, Thomson, 1997.
4. D. M. Dhamdhere, “System Programming and Operating System”, 2nd Edition, TMH, 1993.
5. Houlb, “Compiler Design in C”, PHI, EEE, 1995.

Systems Software

- System:
- 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.

Systems Software

- the hardware devices need user instructions to function.
- A set of instructions that achieve a single outcome are called program or procedure.
- Many programs functioning together to do a task make a **software**.

Software classification

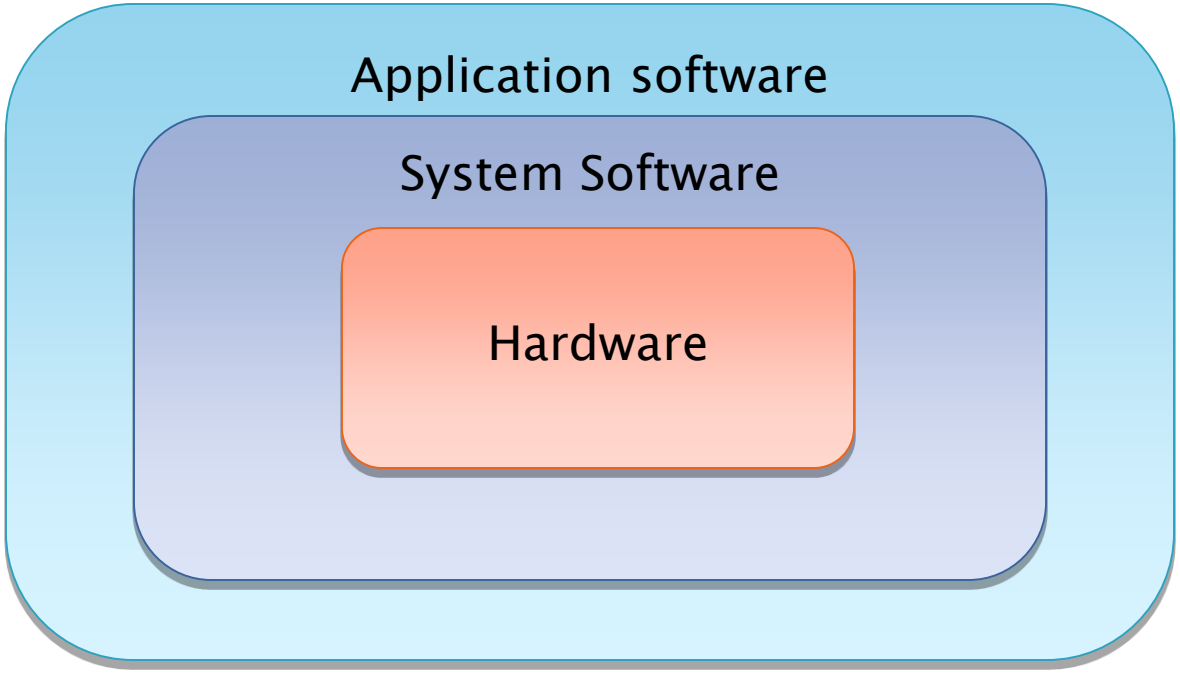
- Software can be classified into
  - System software:
  - **System software** (or **systems software**) is computer software designed to operate and control the computer hardware and to provide a platform for running application software.

    - 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 softwares)
      - Device Drivers

  - Application software:
    - Application software is kind of software which is designed for fulfillment specialized user requirement.
      - MS Office
      - Adobe Photoshop

Cont.

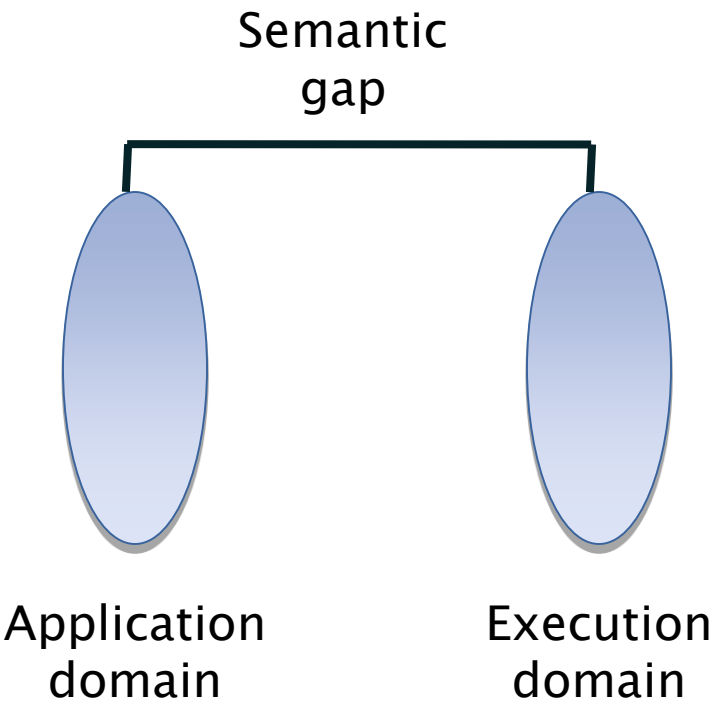
- The system software work as middleware between application software and hardware.



Systems Software

- Language processors (Why?)
  - Language processing activities arise due to the differences between the manner in which a software designer describes the ideas concerning the behavior of software and the manner in which these ideas are implemented in computer system.

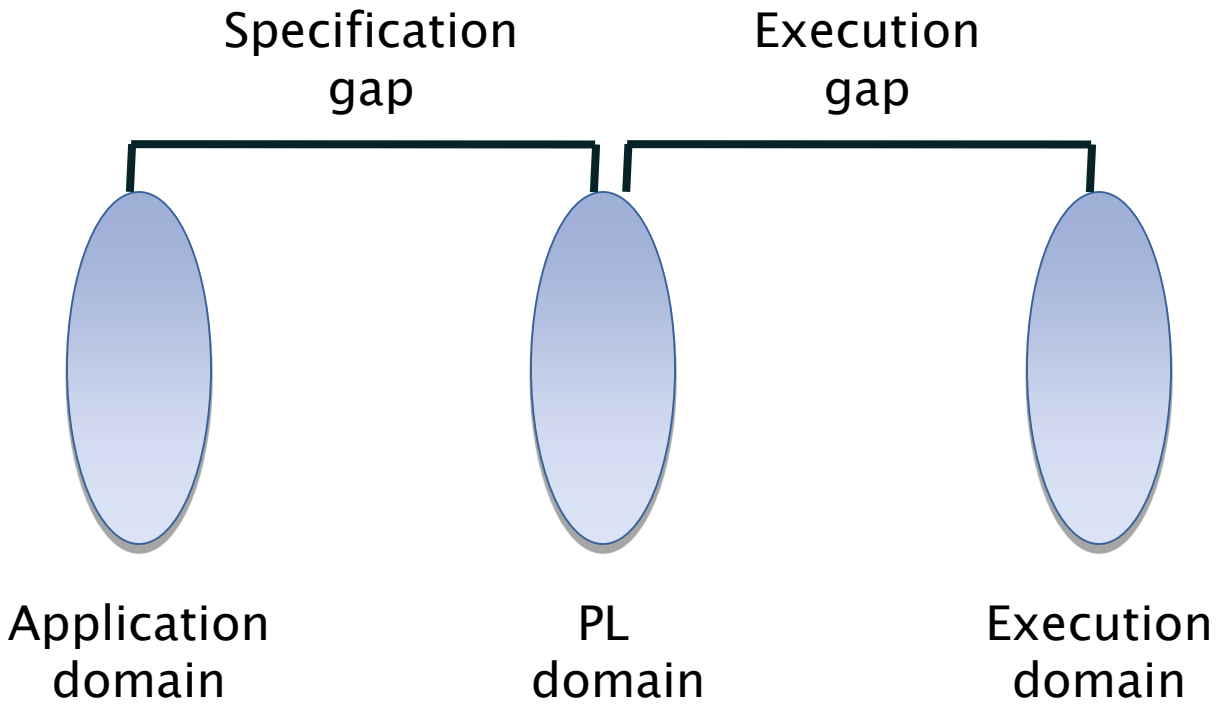
  - The designer expresses the ideas in terms related to the application domain of the software.
  - To implement these ideas, their description has to be interpreted in terms related to the execution domain.



- The term semantics to represent the rules of meaning of a domain, and the term semantic gap to represent difference between the semantics of two domains.

- The semantic gap has many consequences, some of the important are
  - Large development times
  - Large development effort
  - Poor quality software.

- these issues are tackled by software engneering thru' use of methodologies and programming languages.

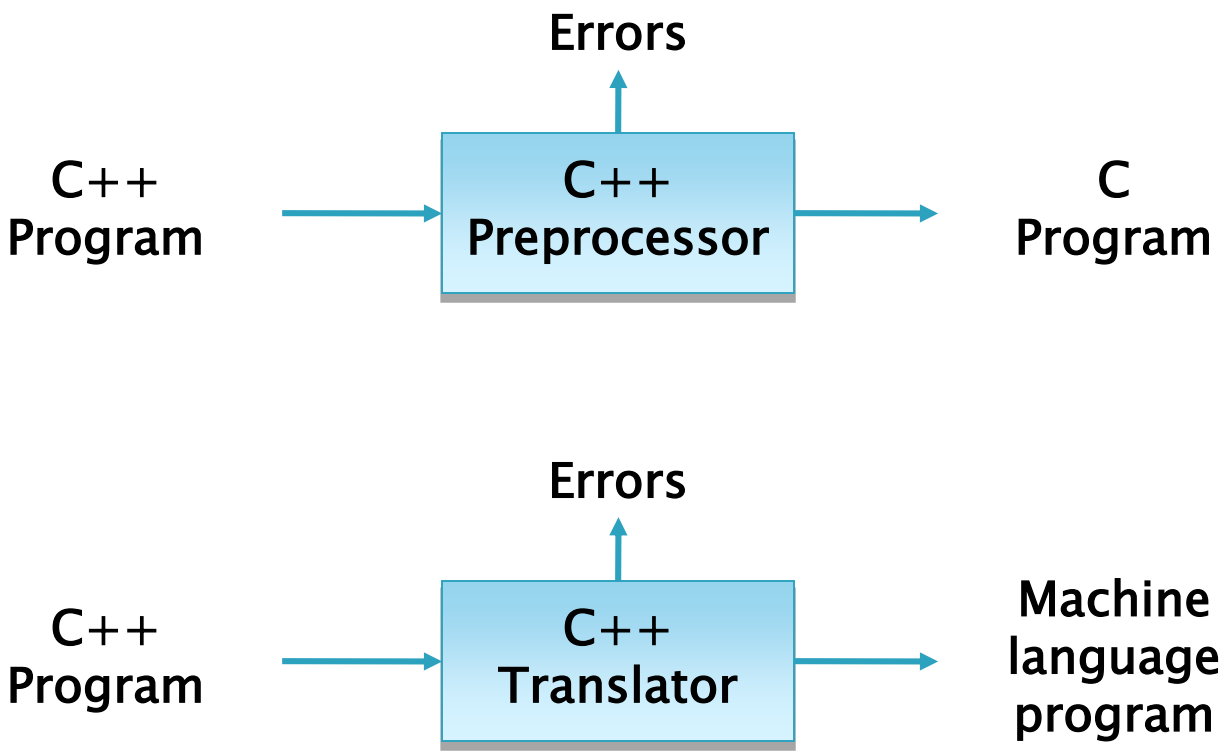


- s/w development team
- Programing language processor

Cont.

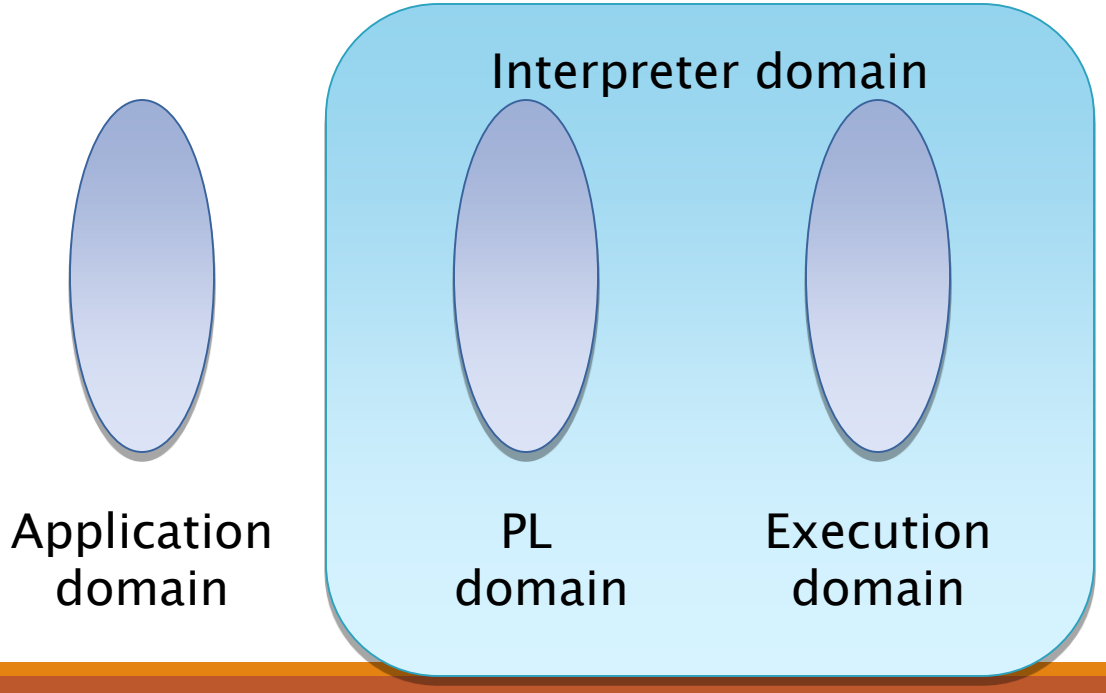
- Language processor: A language processor is software which bridge a specification or execution gap.
  - A Language Translator
  - De-translator
  - Preprocessor
  - Language migrator

Example


Errors
C++
Program
C++
Preprocessor
C
Program
Errors
C++
Program
C++
Translator
Machine
language
program

Interpreter

- An interpreter is language processor which bridges an execution gap without generating a machine language program that means the execution gap vanishes totally.

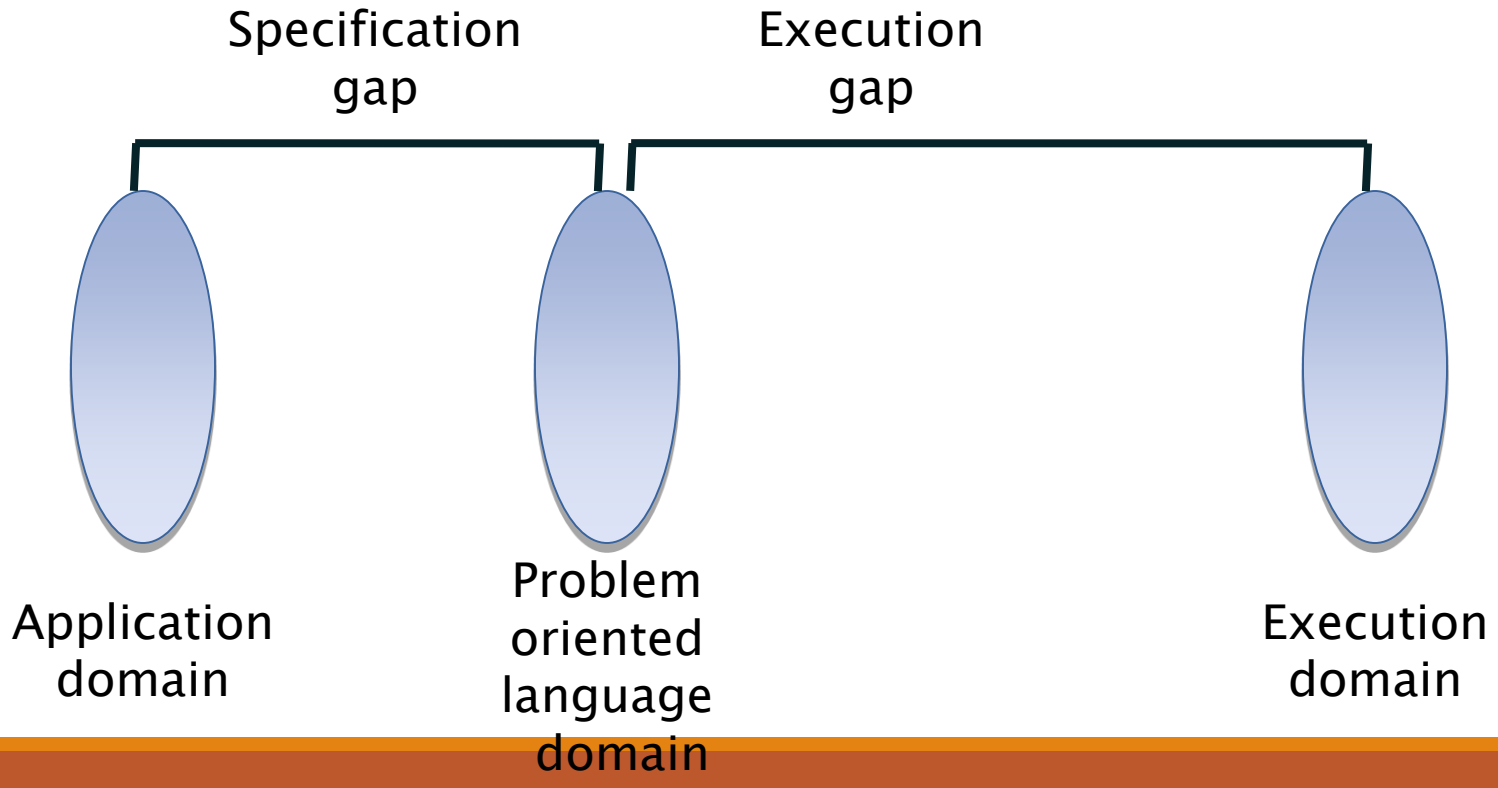


Problem oriented language

- Three consequences of the semantic gap are in fact the consequences of specification gap.
- A classical solution is to develop a PL such that the PL domain is very close or identical to the application domain.
- Such PLs can only used for specific applications, they are problem oriented languages.

Procedure oriented language

- A procedure oriented language provides general purpose facilities required in most application domains.

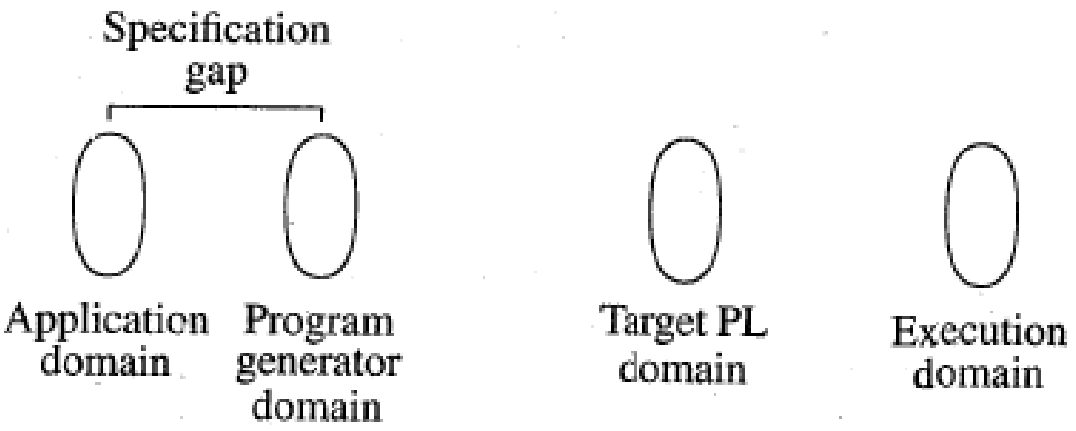


Language Processing Activities

- Fundamental activities divided into those that bridge the specification gap and execution gap.
  - Program generation activities
  - Program execution activities

Cont.

- Program generation activities
  - A program generation activity aims at automatic generation of a program.
  - A source language is a specification language of an application domain and the target language is procedure oriented PL.
  - Program generator introduces a new domain between the application and PL domain , call this the program generator domain.
  - Specification gap now between Application domain and program generation domain, reduction in the specification gap increases the reliability of the generated program.

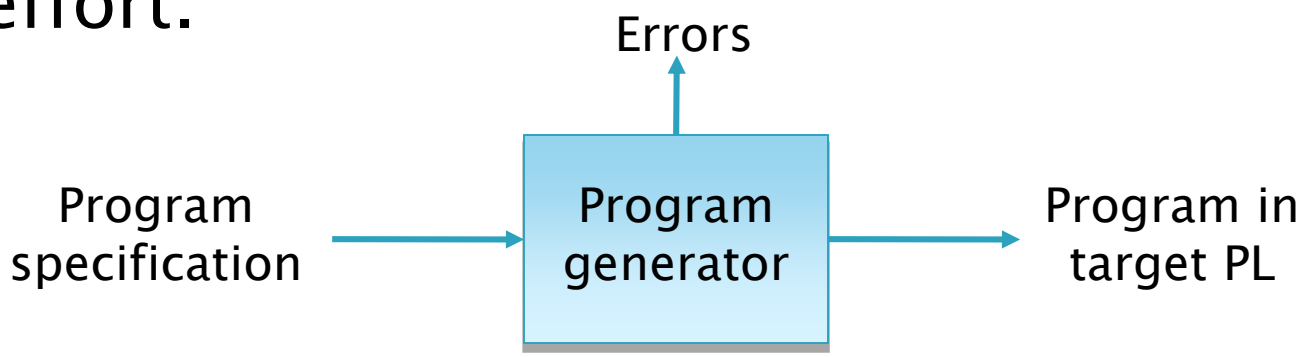


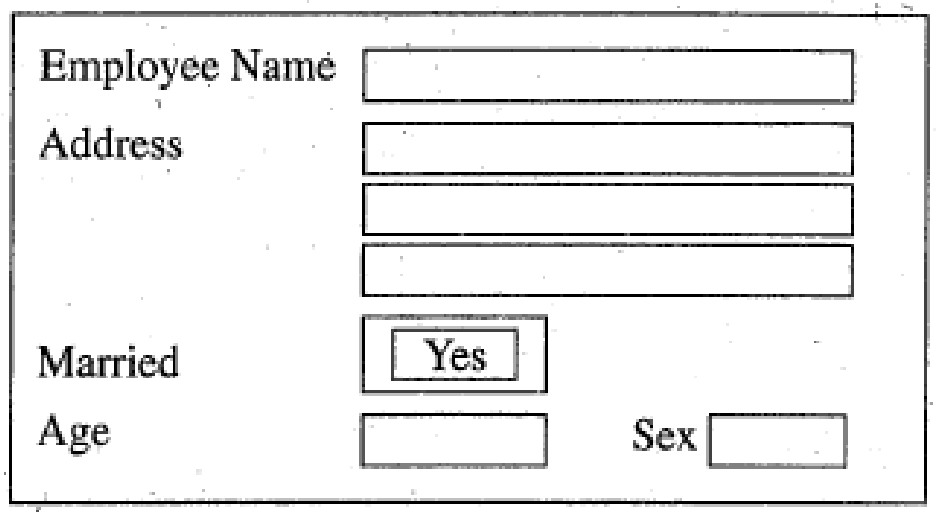
Program Generator Domain

Examples: Adventure maker ,YOYO Games, Alice

Cont.

- This arrangement also reduces the testing effort.





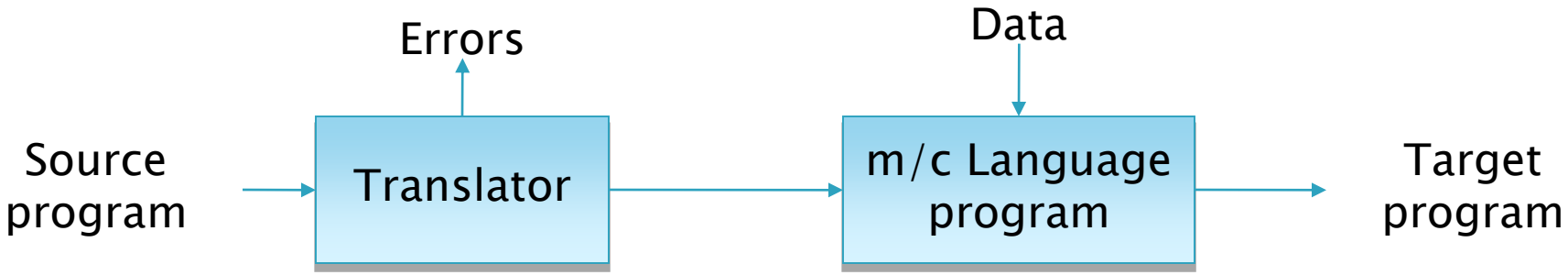
Screen Handling Program

Cont.

- Program Execution
  - Two popular model
    - Program Translation
    - Program Interpretation

Cont.

- The program translation model bridges the execution gap by translating a sources program into program in the machine or assembly language of the computer system, called target program.



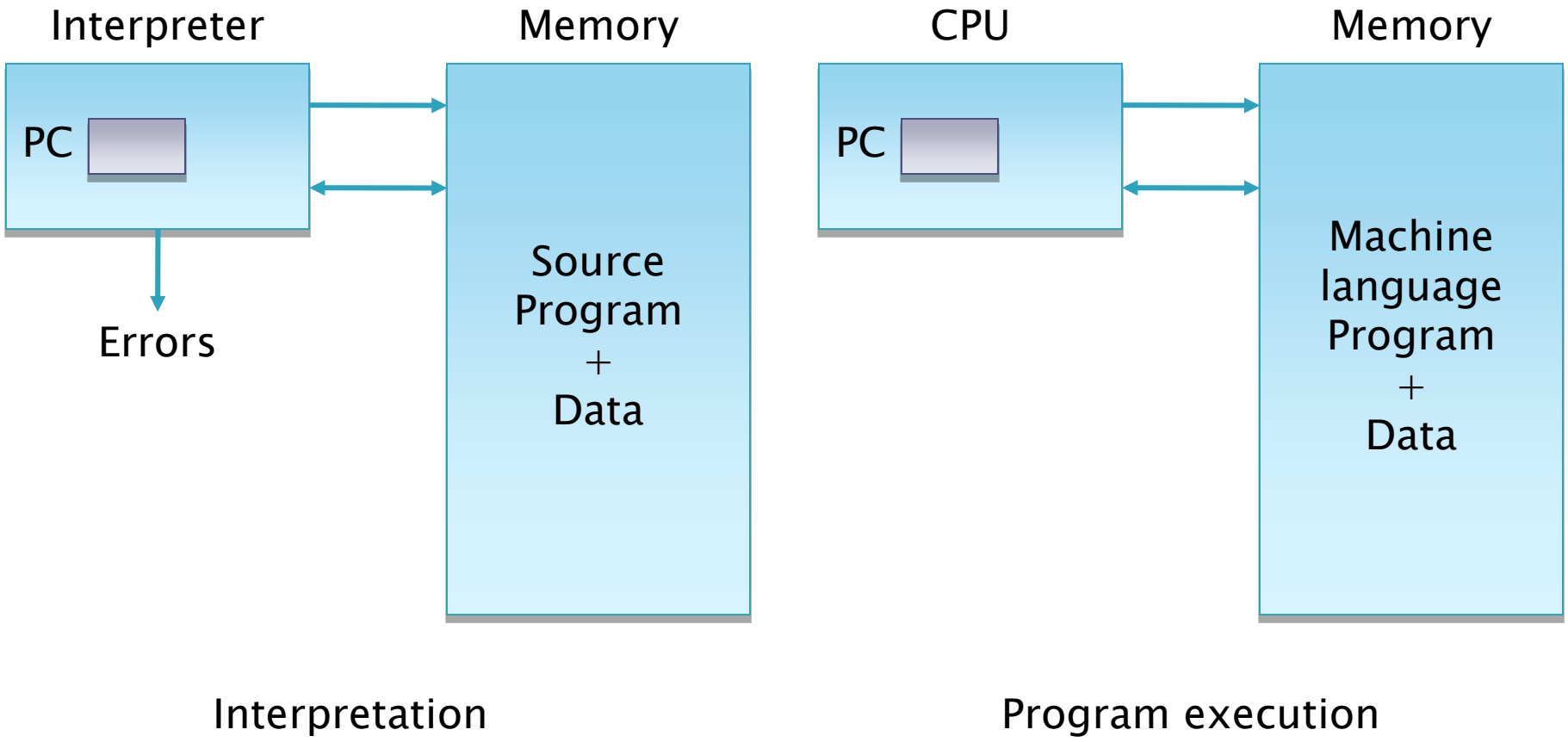
Cont.

- Characteristics of the program translation model:
  - A program must be translated before it can be executed
  - The translated program may be saved in a file. The saved program may be executed repeatedly.
  - A program must be retranslated following modifications.

Cont.

- **Program interpretation**: during interpretation interpreter takes source program statement, determines its meaning and performs actions which implement it.
- The function of an interpreter is same as the execution of machine language program by CPU.

Cont.



Interpretation

Program execution

Cont.

- Characteristics
  - The source program is retained in the source form itself, no target program form exists,
  - A statement is analyzed during its interpretation.
- Comparison
  - In translator whole program is translated into target and if modified the source program, whole source program is translated irrespective to size of modification.
  - That not the in case of interpreter, interpretation is slower than execution of m/c language program.

At the end ...

You will understand:

- the role of system and application software,
- the tasks of a system programmer,
- Knowledge of device drivers
- the principles and methods of designing language compilers
- the approaches to invoking OS services.

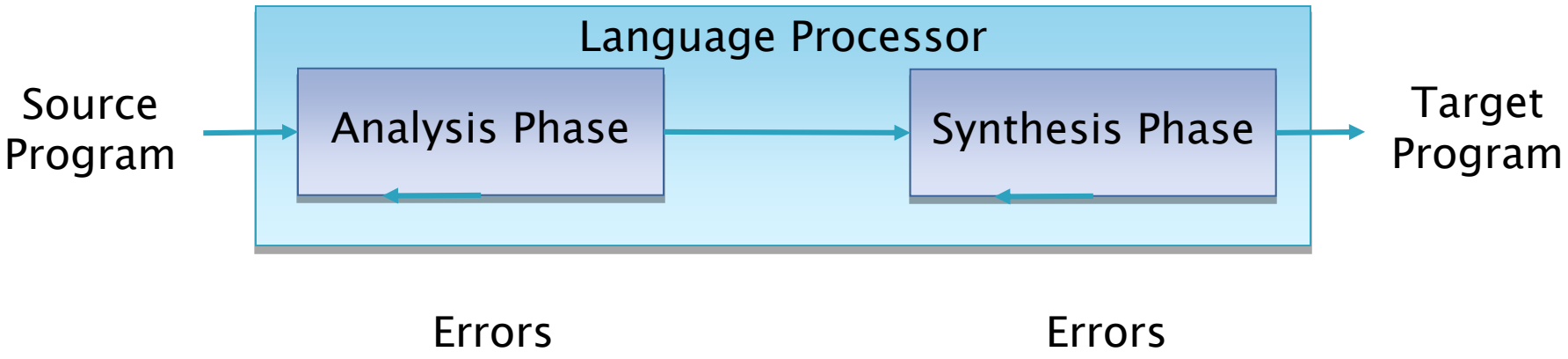
You will be able to

- apply the concepts and techniques of system-level programming,
- write simple compiler modules
- using tools to generate a compiler for a target language
- Unix device driver implementation.

Fundamental of Language Processoing

- Language Processing = Analysis of SP + Synthesis of TP.
- Analysis phase of Language processing
- **Lexical rules** which govern the formation of valid lexical units in the source language.
- **Syntax rules** which govern the formation of the valid statements in the source language.
- **Semantic rules** which associate meaning with the valid statements of the language.

- The synthesis phase is concerned with the construction of target language statement which have same meaning as a source statement.
  - Creation of data structures in the target program(**memory allocation**)
  - Generation of target code.(**Code generation**)



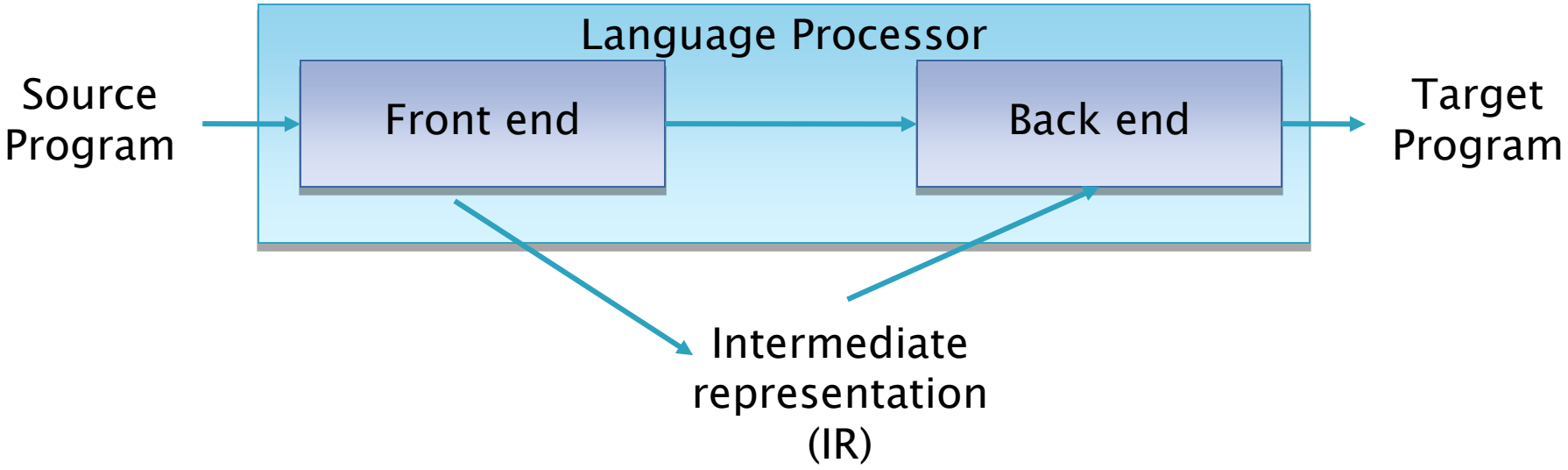
Cont.

- **Forward references**: for reducing execution gap the language processor can performed on a statement by statement basis.
- Analysis of source statement can be immediately followed by synthesis of equivalent target statements. But this may not feasible due to :**Forward reference**
- “A forward reference of a program entity is a reference to the entity which precedes its definition in the program.”

Cont.

- **Language processor pass**: “A language processor pass is the processing of every statement in a source program, or its equivalent representation, to perform a language processing function.”
- **Intermediate representation(IR)**: “An intermediate representation is a representation of a source program which reflects the effect of some, **but not all**, analysis and synthesis tasks performed during language processing.”

Cont.

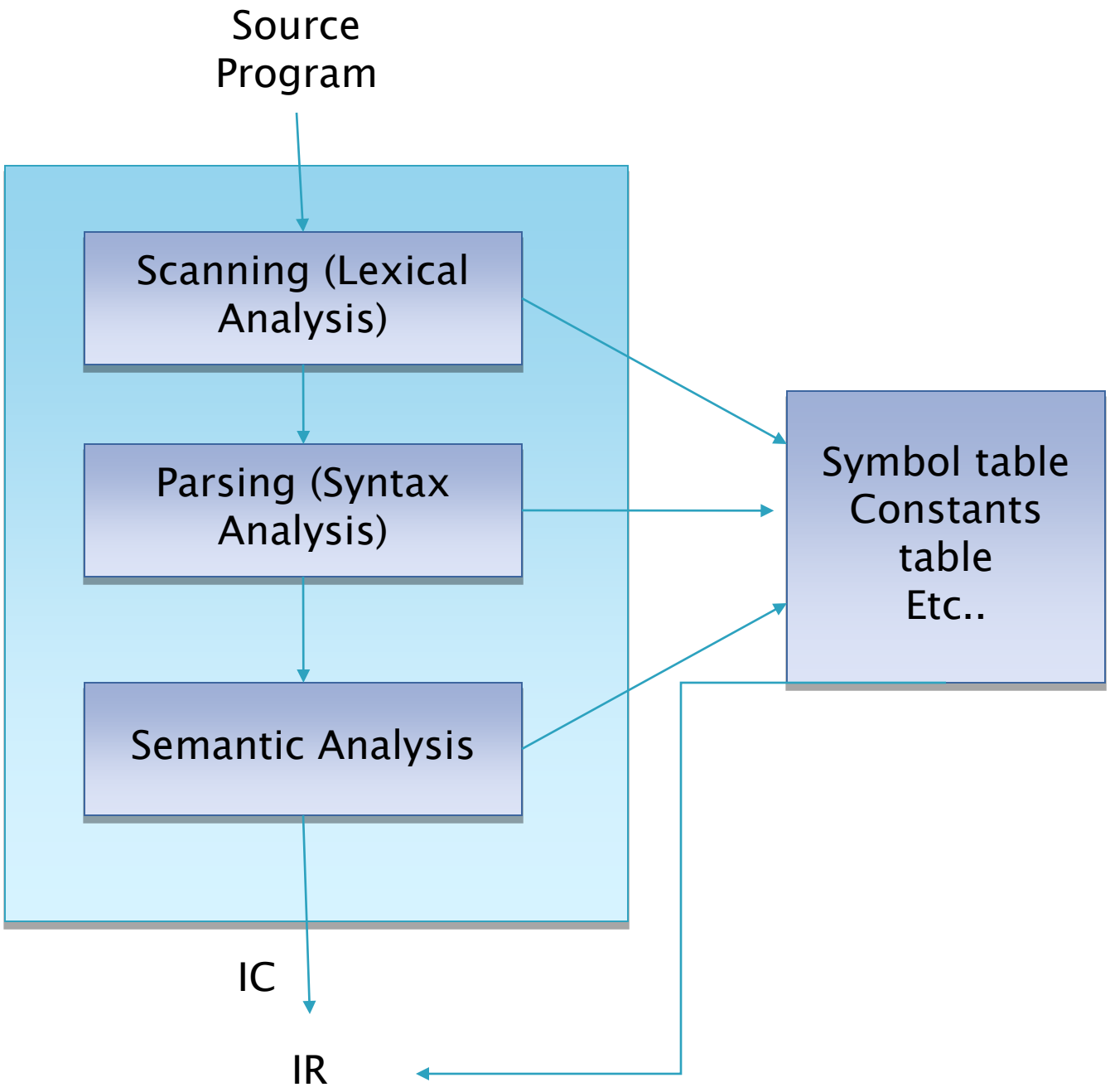

Language Processor
Source Program
Front end
Back end
Target Program
Intermediate representation (IR)

Cont.

- **Semantic Action**: “All the actions performed by the front end, except lexical and syntax analysis, are called semantic action.
  - Checking semantic validity of constructs in SP
  - Determining the meaning of SP
  - Constructing an IR

Toy Compiler

- The Front End
  - The front end performs lexical, syntax and semantic analysis of the source program, each kind of analysis involves the following functions:
    - Determine validity of source statement from the view point of the analysis.
    - Determine the ‘content’ of a source statement
      - For lexical, the lexical class to which each lexical unit belongs.
      - Syntax analysis it is syntactic structure of source program.
      - Semantic analysis the content is the meaning of a statement.
    - Construct a suitable representation of source statement for use by subsequent analysis function/synthesis phase.


Source Program
Scanning (Lexical Analysis)
Parsing (Syntax Analysis)
Semantic Analysis
Symbol table
Constants table
Etc..
IC
IR

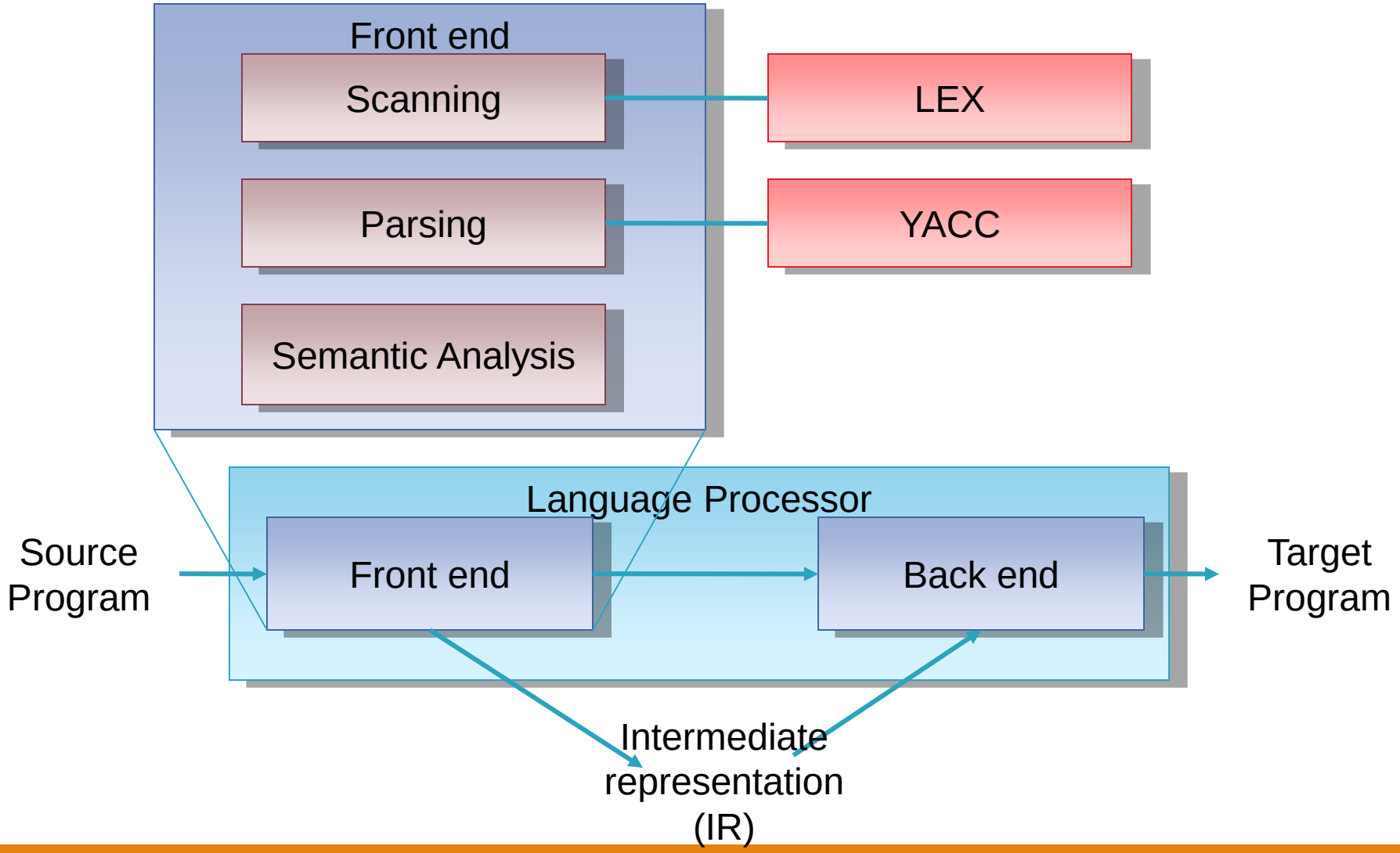
- Out put of front end produced two components: (IR)
  - Table of information
    - The symbol table which contain information concerning all identifier used in the source program.
  - An intermediate code (IC) which is a description of the source program.
    - The IC is a sequence of IC units, each IC unit representing the meaning of one action in SP. IC units may contain references to the information in various table.

Cont.

- Lexical Analysis (Scanning):
  - Lexical analysis identifies the lexical units in source statement
  - it then classifies the unit into different classes
  - Ex. Id's, Constant reserved id's etc. and enters them into different tables.
  - This classification may be based on the nature of a string or on the specification of the source language.
  - Lexical analysis build descriptor called token, for each lexical unit. It contain two fields class code and number in class
    - Class code: identifies the class to which a lexical unit belongs.
    - Number in class: entry number of lexical unit in the relevant table.

- Syntax Analysis(Parsing)
  - Syntax analysis process the string token built by lexical analysis to determine the statement class e.g. assignment statement, if statement, etc.
  - Syntax Analysis builds an IC which represents the structure of the statement.
  - IC is passed to semantic analysis to determine the meaning of the statement.

- Semantic analysis
  - Semantic analysis of declaration statements differs from the semantic analysis of imperative statements.
  - The former results in addition of information to the symbol table e.g. Type, length and dimensionality of variables.
  - The latter identifies the sequence of action necessary to implement the meaning of a source statement.
  - When semantic analysis determines the meaning of a sub tree in the IC, it adds information to a table or adds an action to sequence of the action.
  - The analysis ends when the tree has been completely processed. The update tables and the sequence of action constitute the IR produced by the analysis phase.


Front end
Scanning
Parsing
Semantic Analysis
LEX
YACC
Language Processor
Source Program
Front end
Back end
Target Program
Intermediate representation (IR)