CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science
Banaras Hindu University
Varanasi
Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

January 13, 2021

What is Programming Language I

- **What is a Programming Language?**

What is Programming Language II

✓ A programming language is a notation designed to connect instructions to a machine or a computer.

✓ Programming languages are mainly used to control the performance of a machine or to express algorithms.

✓ At present, thousand programming languages have been implemented.

✓ In the computer field, many languages need to be stated in an imperative form, while other programming languages utilize declarative form.

✓ The program can be divided into two forms such as syntax and semantics. Some languages are defined by an

What is Programming Language III

International Organization for Standardization (ISO) standard like C language.

Types of Programming Languages I

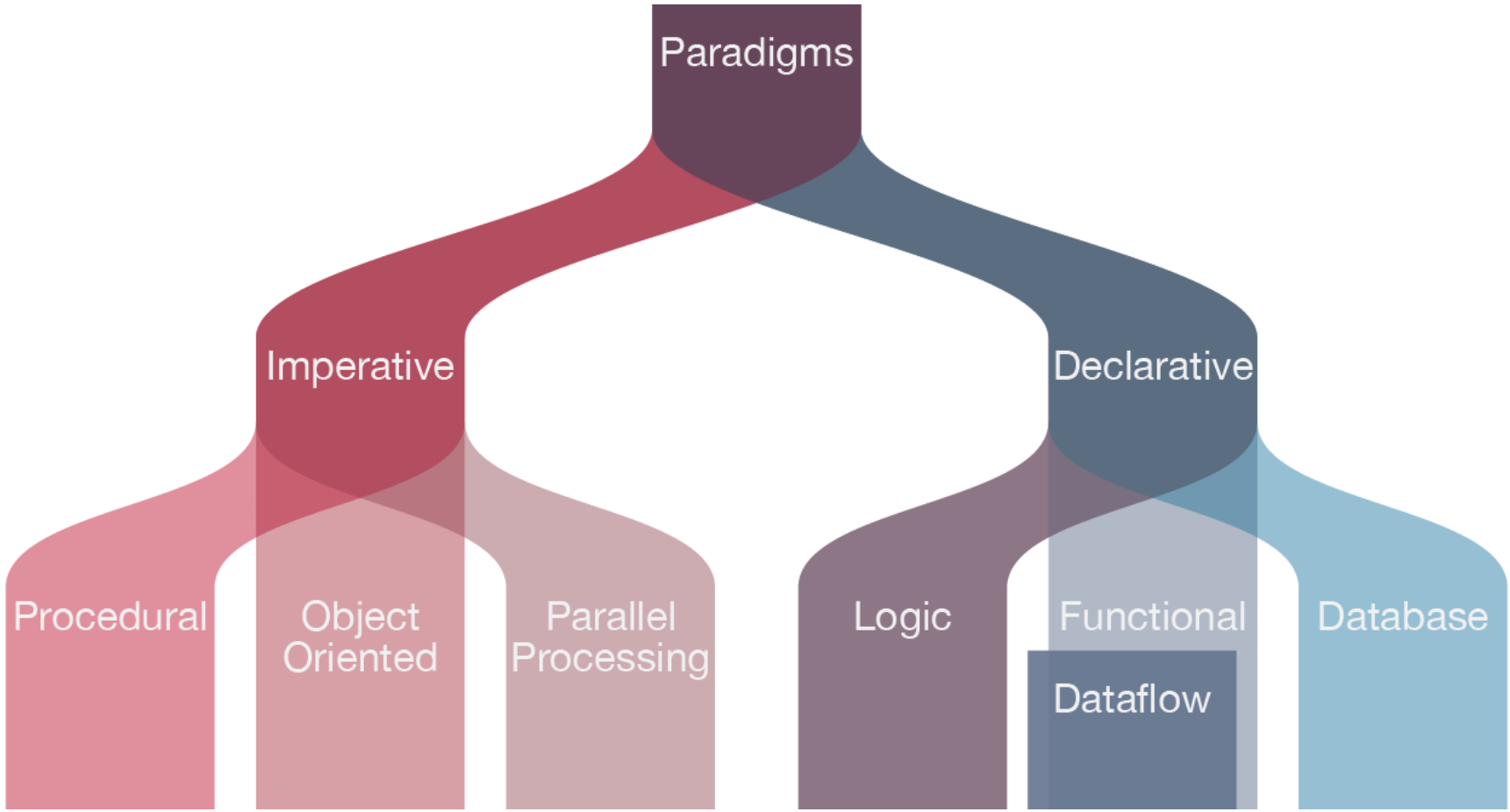


Figure 1: Programming Paradigms (Read it like pa·ruh·daim)

Types of Programming Languages II

▶ Declarative programming

✓ Is a programming paradigm—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.

✓ Declarative programming is when you write your code in such a way that it describes what you want to do, and not how you want to do it.

✓ It is left up to the compiler to figure out the how. You simply give a command to do the task.

▶Examples of declarative programming languages are SQL and Prolog.

Types of Programming Languages III

▶ Imperative programming

✓ Languages differ from declarative languages on one fundamental point: imperative programming focuses on the "how", declarative programming on the "what".

✓ Imperative programming languages are composed of step-by-step instructions (how) for the computer.

✓ They describe explicitly which steps are to be performed in what order to obtain the desired solution at the end.

✓ Imperative programming: a programming paradigm that uses statements that change a program's state.

Types of Programming Languages IV

✓ An imperative program consists of commands for the computer to perform.

✓ Imperative programming focuses on describing how a program operates.

1. **Structured Programming Language** : is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Types of Programming Languages V

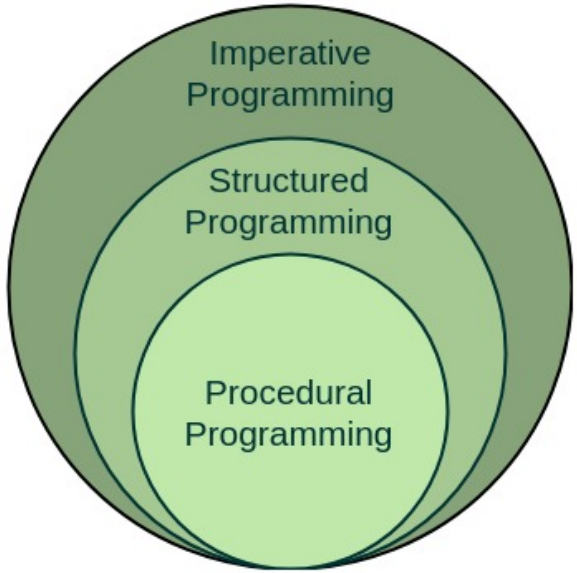


Figure 2: Structured Programming Language

Types of Programming Languages VI

✓ The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements.

✓ Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Types of Programming Languages VII

- The structured program mainly consists of three types of elements:
  - Sequence Statements
  - Iteration Statements
  - Selection Statements
- These structural blocks are –
  - Decision making blocks: if-else-elseif, switch-cases,
  - Repetitive blocks: For-loop, While-loop, Do-while loop etc
  - Subroutines/procedures: functions

Types of Programming Languages VIII

```
if (<condition>)
{
  <statements>;
}
else
{
  <statements>;
}
```

```
if (<condition>)
{
  <statements>;
}
else if(<condition>)
{
  <statements>;
}
else
{
  <statements>;
}
```

```
goto label;
<statements>;
<statements>;
label:
<statements>;
```

```
switch (x)
{
 case <1>:
   break;
 default:
}
```

Figure 3: Statements and conditions

Types of Programming Languages IX

```
for (i = 0; i < max; i++)
{
    <statements>;
}
```

```
while(<condition>)
{
    <statements>;
}
```

```
do
{
    <statements>;
}
while(<condition>);
```

Figure 4: Loops

```
int function ()
{
    return 0;
}
```

```
int procedure (void)
{
    return 0;
}
```

```
int main()
{
    return 0;
}
```

Figure 5: Functions and procedures

Types of Programming Languages X

```
Structured:
IF x<=y THEN
  BEGIN
      z := y-x;
      q :=SQRT(z);
  END
ELSE
  BEGIN
      z := x-y;
      q := -SQRT(z)
  END;
WRITELN(z,q);
```

```
Unstructured:
IF x>y THEN GOTO 2;
z := y-x;
q := SQRT(z);
GOTO 1;
2: z:= x-y;
q:=-SQRT(z);
1: writeln(z,q);
```

Figure 6: Loops

Types of Programming Languages XI

1. Sequence Execute a list of statements in order.

Example: Baking Bread

```
Add flour.
Add salt.
Add yeast.
Mix.
Add water.
Knead.
Let rise.
Bake.
```

Types of Programming Languages XII

2. Repetition Repeat a block of statements while a condition is true.

Example: Washing Dishes

```
Stack dishes by sink.
Fill sink with hot soapy water.
While moreDishes
    Get dish from counter,
    Wash dish,
    Put dish in drain rack.
End While
Wipe off counter.
Rinse out sink.
```

Types of Programming Languages XIII

3. Selection Choose at most one action from several alternative conditions.

Example: Sorting Mail

```
Get mail from mailbox.
Put mail on table.
While moreMailToSort
    Get piece of mail from table.
    If pieceIsPersonal Then
       Read it.
    ElseIf pieceIsMagazine Then
       Put in magazine rack.
    ElseIf pieceIsBill Then
       Pay it,
    ElseIf pieceIsJunkMail Then
       Throw in wastebasket.
    End If
End While
```

Figure 7: Sequence, Repetition and Selection

Types of Programming Languages XIV

- **Disadvantages of Structured Programming Approach:**
  - Since it is Machine-Independent, So it takes time to convert into machine code.
  - The converted machine code is not the same as for assembly language.
  - The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.

Types of Programming Languages XV

- Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Compiler I

✓ A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits).

✓ The computer processes the machine code to perform the corresponding tasks.

✓ A compiler should comply with the syntax rule of that programming language in which it is written.

Compiler II

✓ However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

✓ A compiler reads program in one language.

Compiler III

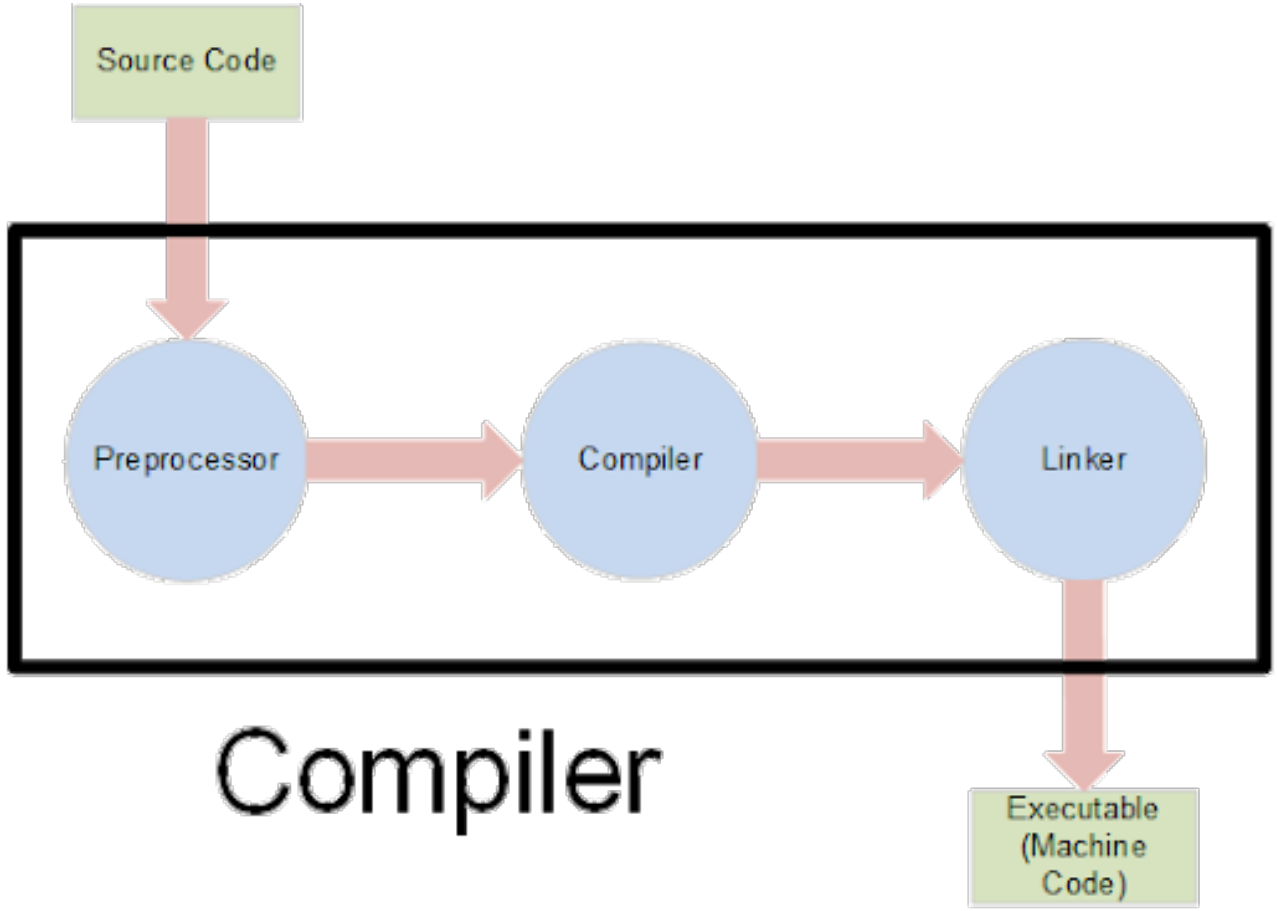


Figure 8: Compiler Operation

Compiler IV

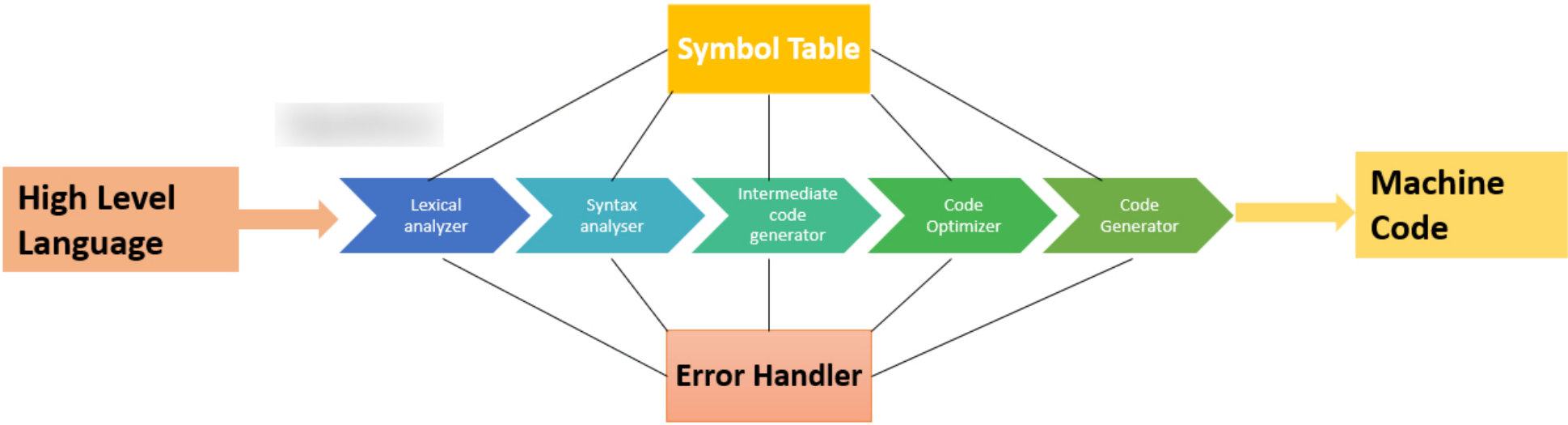


Figure 9: Phases of Compiler

Compiler V

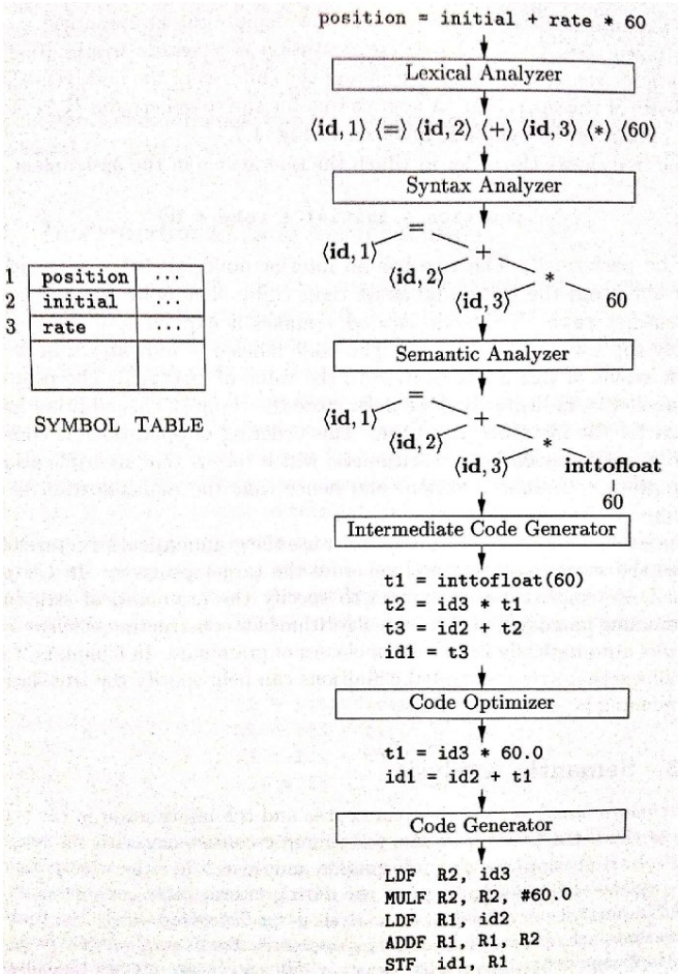


Figure 10: Compiler Operation

Interpreter I

►In computer science, an interpreter is a computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.

► An interpreter generally uses one of the following strategies for program execution:

1. Parse the source code and perform its behavior directly;
2. Translate source code into some efficient intermediate representation and immediately execute this;

Interpreter II

3. Explicitly execute stored pre-compiled code made by a compiler which is part of the interpreter system.

Compiler Vs. Interpreter I

Basis of Difference	Compiler	Interpreter
steps	Compiler transforms code written in a high-level programming language into the machine code, at once, before program runs	Interpreter coverts each high-level program statement, one by one, into the machine code, during program run
steps	Intermediate object code is generate	No intermediate code is generated
steps	Conditional control statements are executed faster	Conditional control statements are executed slower
steps	Memory requirement : More	Memory requirement: Less
steps	Program need not be compiled every time	Every time high level program is converted into lower level program

Compiler Vs. Interpreter II

Basis of Difference	Compiler	Interpreter
steps	Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted
Programming Steps	► Create the program. ►Compiler will parse or analyses all of the language statements for its correctness. If incorrect, throws an error. ►If no error, the compiler will convert source code to machine code. ►It links different code files into a runnable program(know as exe) ►Run the Program	Create the Program No linking of files or machine code generation Source statements executed line by line DURING Execution
Advantage	The program code is already translated into machine code. Thus, it code execution time is less.	Interpreters are easier to use, especially for beginners.

Compiler Vs. Interpreter III

Basis of Difference	Compiler	Interpreter
Disadvantage	You can't change the program without going back to the source code.	Interpreted programs can run on computers that have the corresponding interpreter.
Machine code	Store machine language as machine code on the disk	Not saving machine code at all.
Running time	Compiled code run faster	Interpreted code run slower
Model	It is based on language translationlinking-loading model.	It is based on Interpretation Method.
Program generation	Generates output program (in the form of exe) which can be run independently from the original program.	Do not generate output program. So they evaluate the source program at every time during execution.

Compiler Vs. Interpreter IV

Basis of Difference	**Compiler**	**Interpreter**
Execution	Program execution is separate from the compilation. It performed only after the entire output program is compiled.	Program Execution is a part of Interpretation process, so it is performed line by line.
Memory requirement	Target program execute independently and do not require the compiler in the memory.	The interpreter exists in the memory during interpretation.
Best suited for	Bounded to the specific target machine and cannot be ported. C and C++ are a most popular a programming language which uses compilation model.	For web environments, where load times are important. Due to all the exhaustive analysis is done, compiles take relatively larger time to compile even small code that may not be run multiple times. In such cases, interpreters are better.

Compiler Vs. Interpreter V

Basis of Difference	**Compiler**	**Interpreter**
Code Optimization	The compiler sees the entire code upfront. Hence, they perform lots of optimizations that make code run faster	Interpreters see code line by line, and thus optimizations are not as robust as compilers
Dynamic Typing	Difficult to implement as compilers cannot predict what happens at turn time.	Interpreted languages support Dynamic Typing
Usage	It is best suited for the Production Environment	It is best suited for the program and development environment.

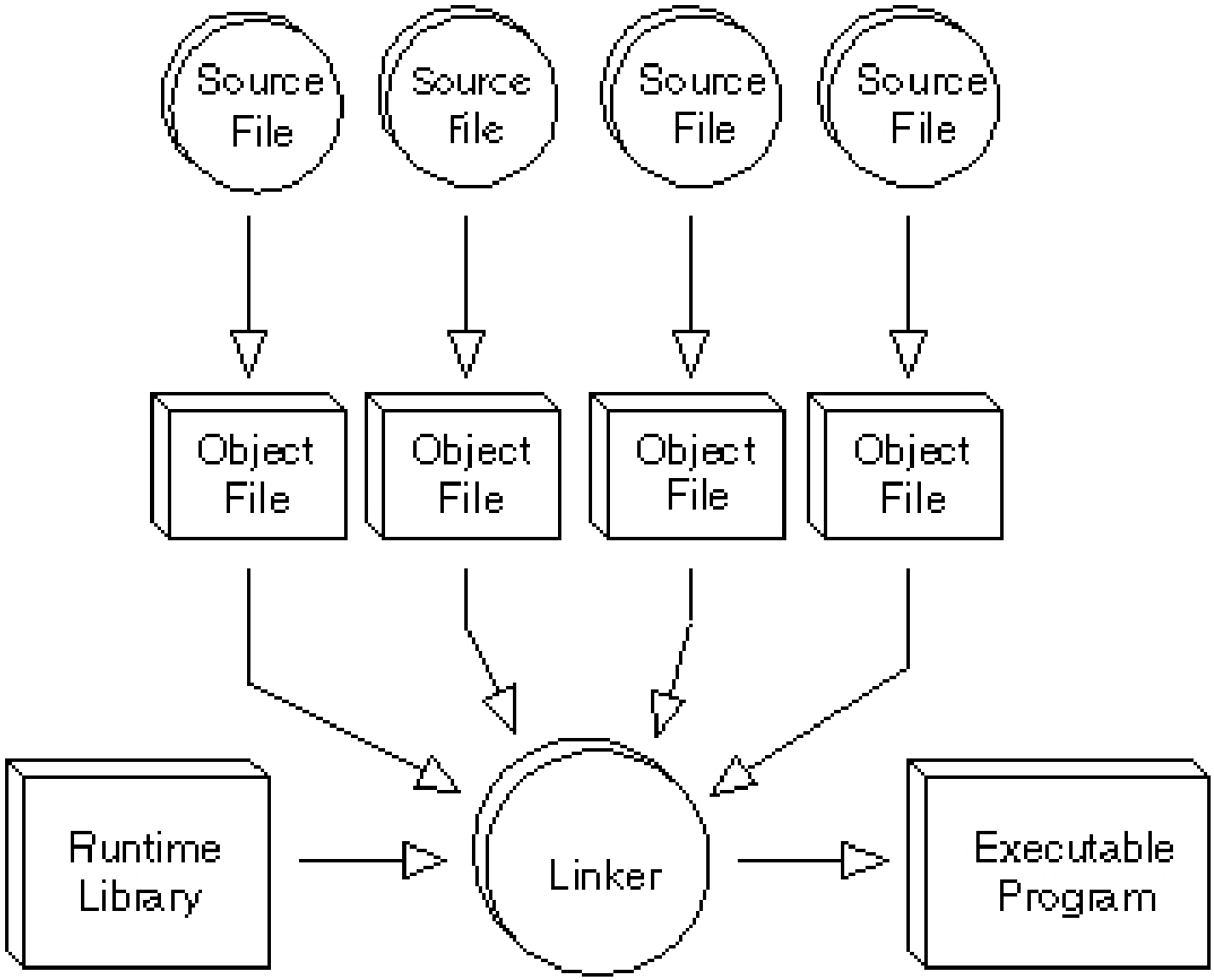
Compiler Vs. Interpreter VI

Basis of Difference	Compiler	Interpreter
Error execution	Compiler displays all errors and warning at the compilation time. Therefore, you can't run the program without fixing errors	The interpreter reads a single statement and shows the error if any. You must correct the error to interpret next line.
Input	It takes an entire program	It takes a single line of code.
Output	Compliers generates intermediate machine code.	Interpreter never generate any intermediate machine code.
Errors	Display all errors after, compilation, all at the same time.	Displays all errors of each line one by one.
Pertaining Programming languages	C,C++,C#, Scala, Java all use complier.	PHP, Perl, Ruby uses an interpreter.

Linker I

In computing, a linker or link editor is a computer system program that takes one or more object files (generated by a compiler or an assembler) and combines them into a single executable file, library file, or another "object" file.

Linker II


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

Linker III

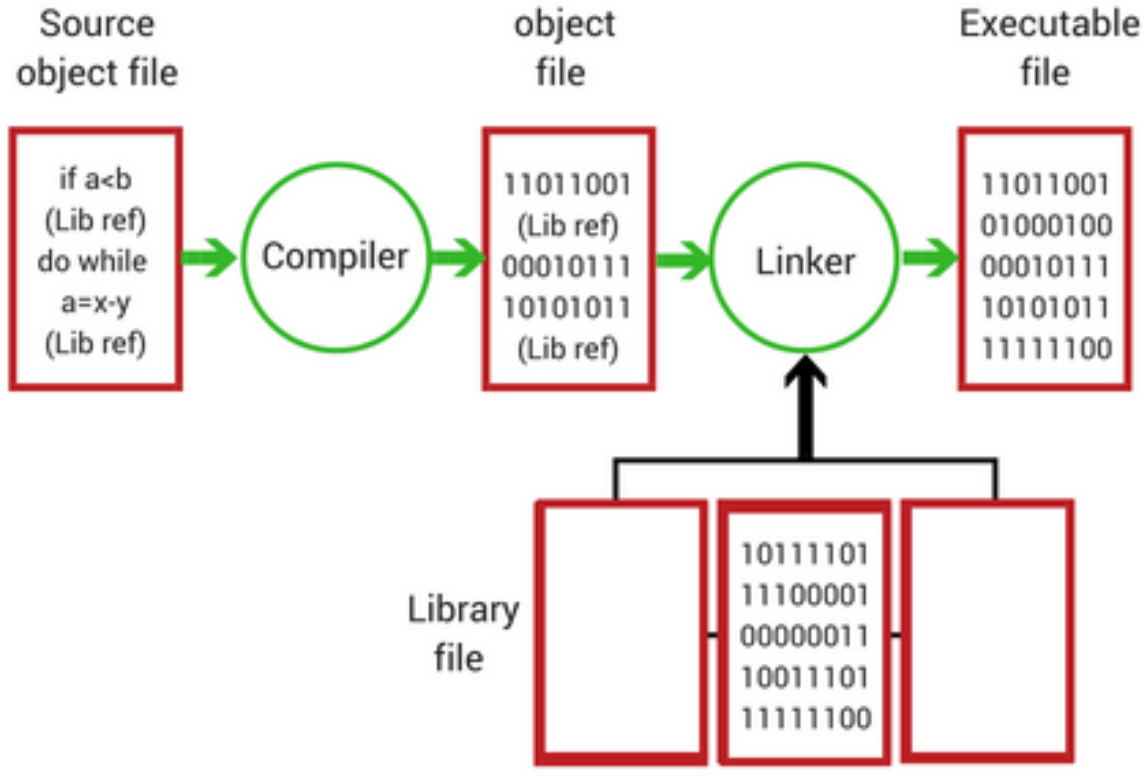


Figure 11: Linker

Linker IV

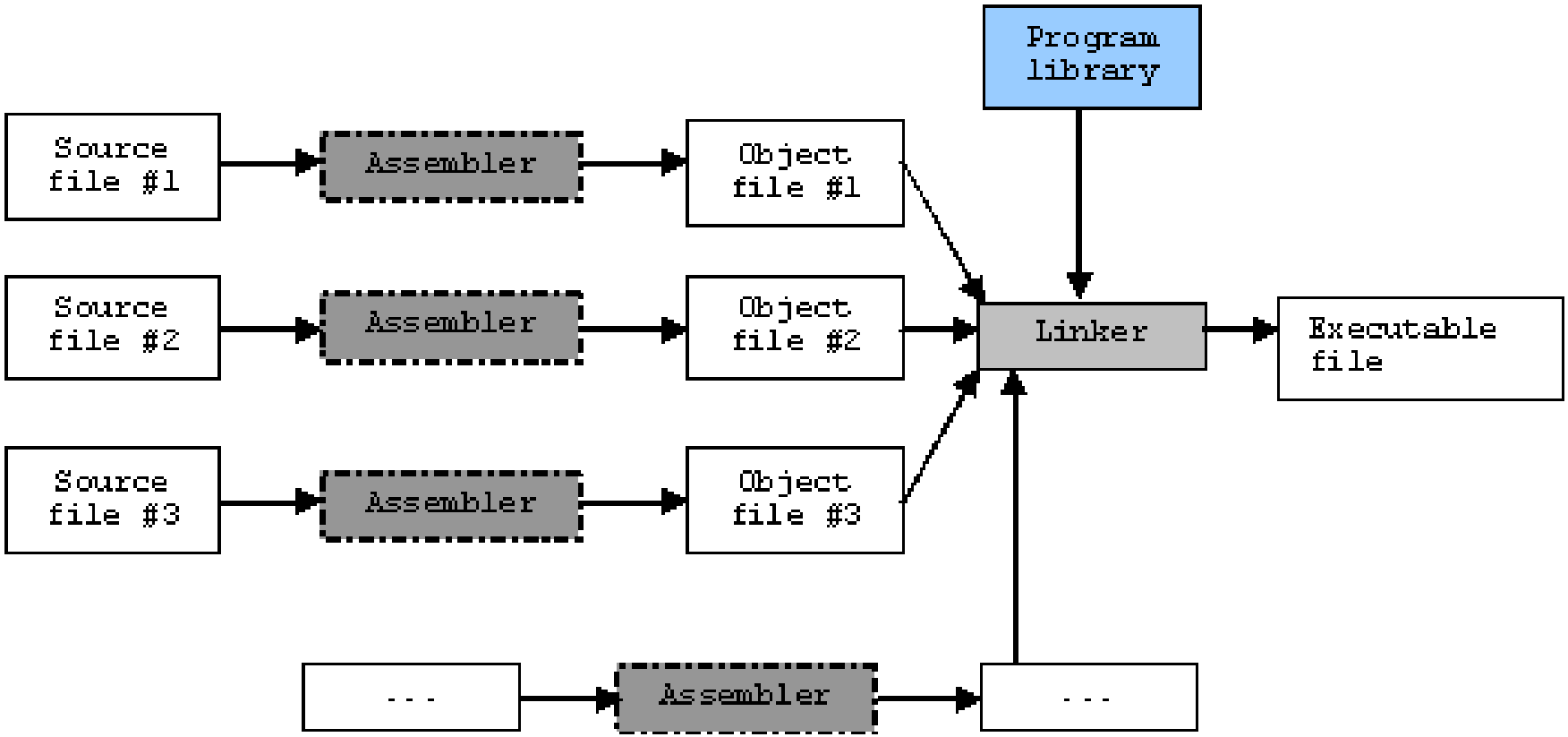


Figure 12: The object files linking process

Linker V

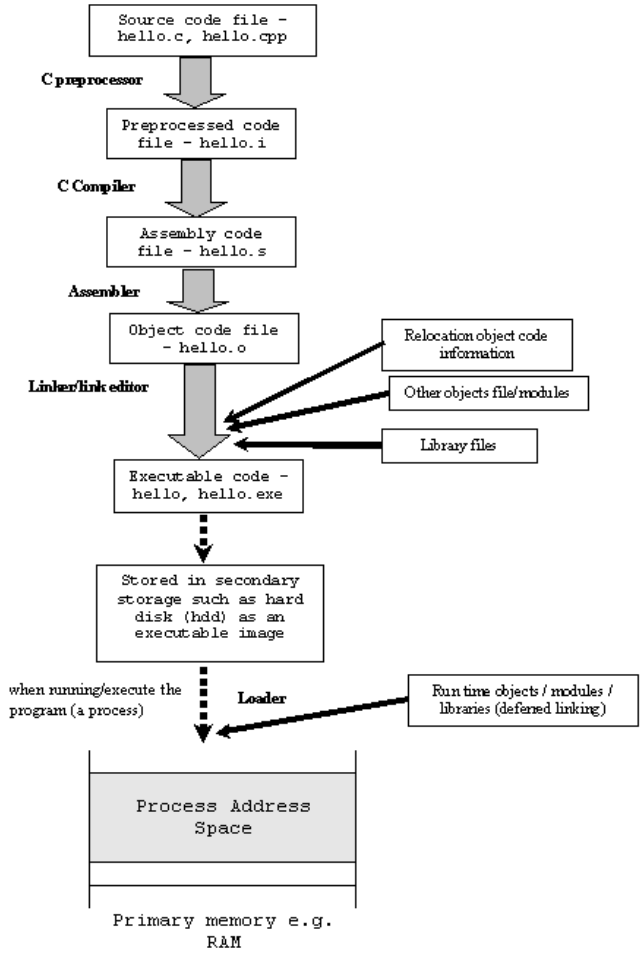


Figure 13: Compile, link and execute stages for running program

Type of Linkers:

1. Dynamic linking and Dynamic Libraries : Dynamic Linking doesn't require the code to be copied, it is done by just placing name of the library in the binary file.
   - ✓ The actual linking happens when the program is run, when both the binary file and the library are in memory.
   - ✓ Loading the program will load these objects/libraries as well, and perform a final linking.
   - ► **This approach offers two advantages:**

Linker VII

1. Often-used libraries (for example the standard system libraries) need to be stored in only one location, not duplicated in every single executable file, thus saving limited memory and disk space.
   - ✓ If a bug in a library function is corrected by replacing the library, all programs using it dynamically will benefit from the correction after restarting them.
   - ✓ Programs that included this function by static linking would have to be re-linked first. There are also disadvantages:

Linker VIII

2. Known on the Windows platform as "DLL hell", an incompatible updated library will break executables that depended on the behavior of the previous version of the library if the newer version is incorrectly not backward compatible.
   ✓ A program, together with the libraries it uses, might be certified (e.g. as to correctness, documentation requirements, or performance) as a package, but not if components can be replaced (this also argues against automatic OS updates in critical systems; in both cases, the OS and libraries form part of a qualified environment).

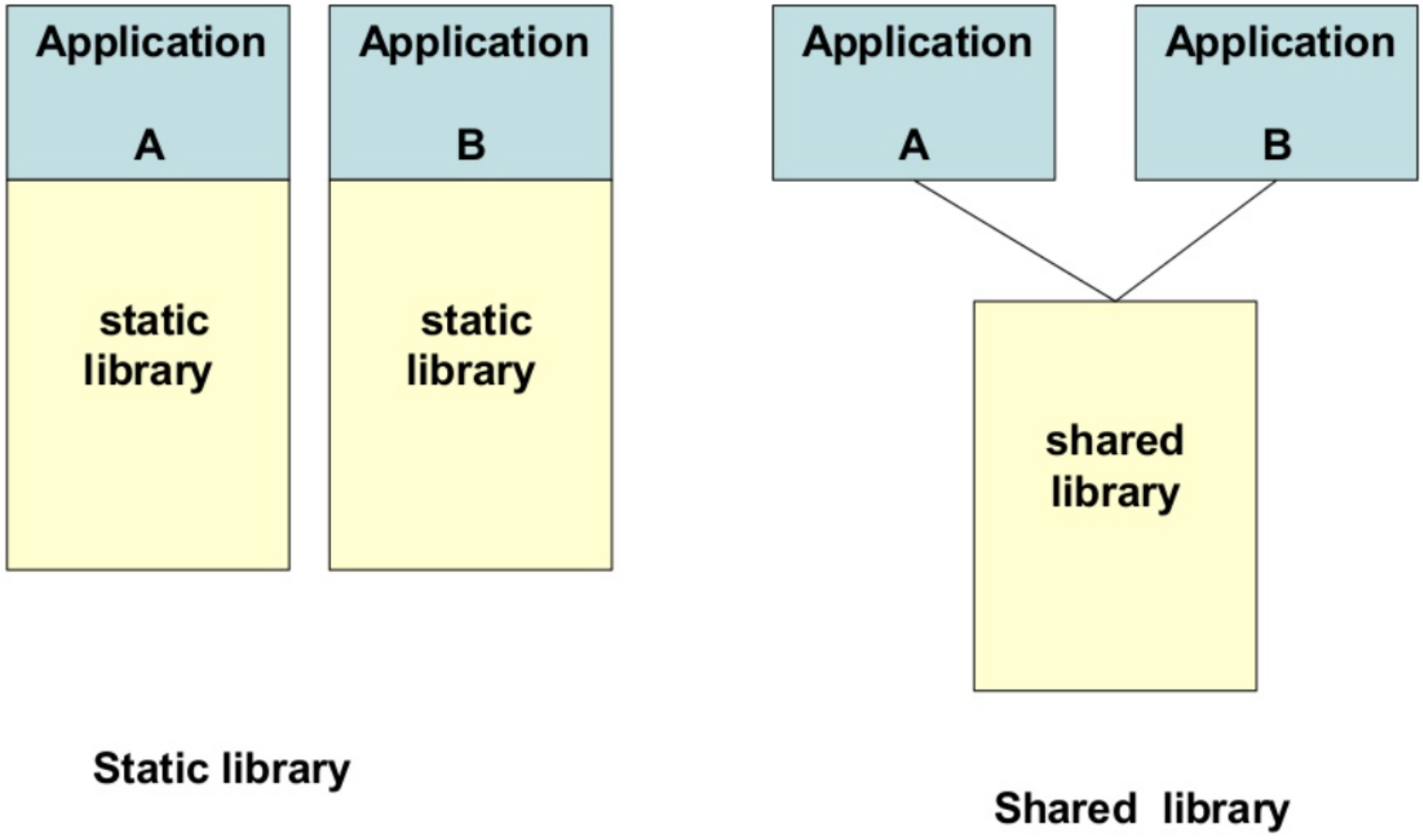
Linker IX

2. Static Linking and Static Libraries : Static linking is the result of the linker copying all library routines used in the program into the executable image.
   - ✓ This may require more disk space and memory than dynamic linking, but is more portable, since it does not require the presence of the library on the system where it runs.
   - ✓ Static linking also prevents "DLL hell", since each program includes exactly the versions of library routines that it requires, with no conflict with other programs.

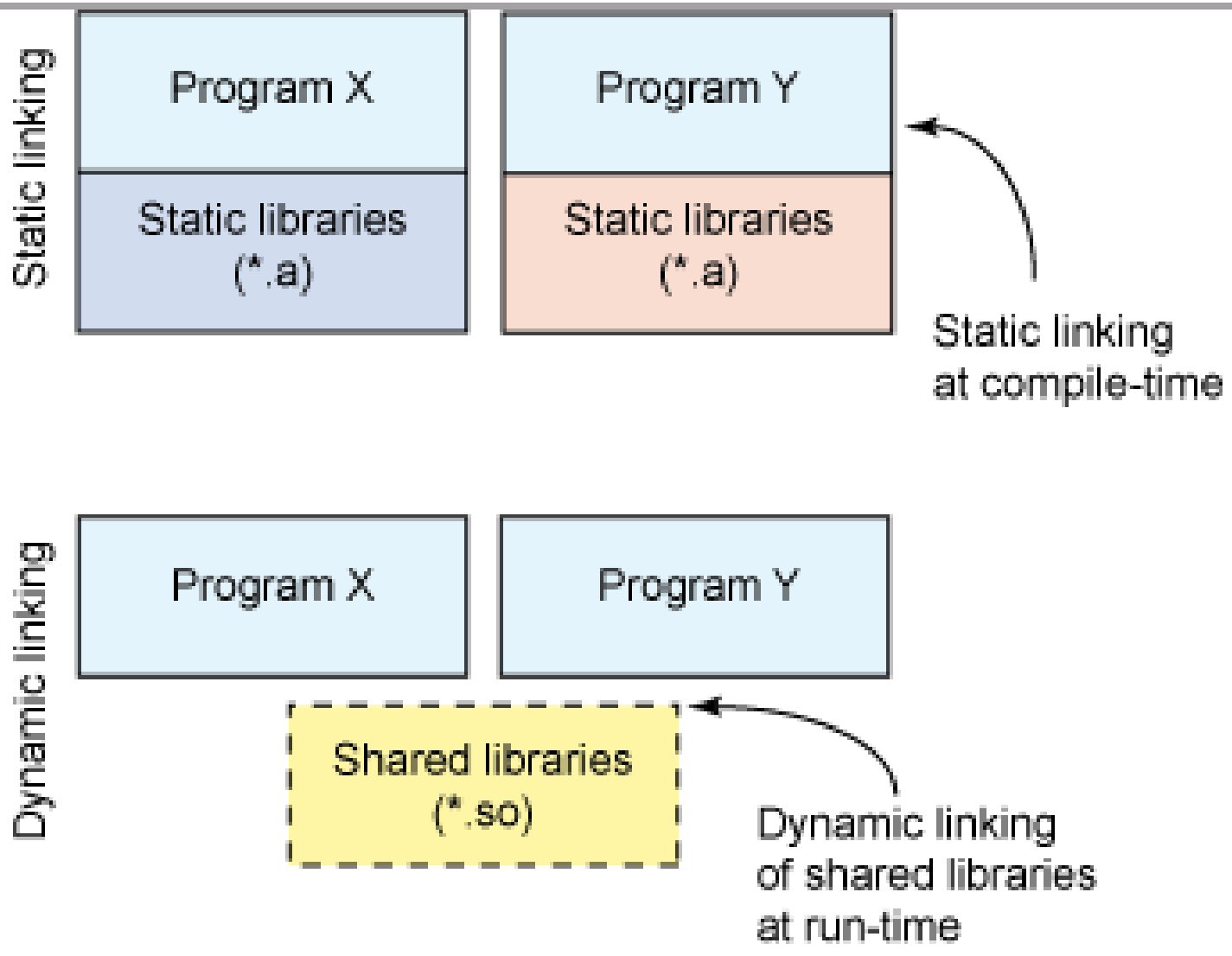
Linker X

✓ A program using just a few routines from a library does not require the entire library to be installed.

Static Library vs. Shared Library



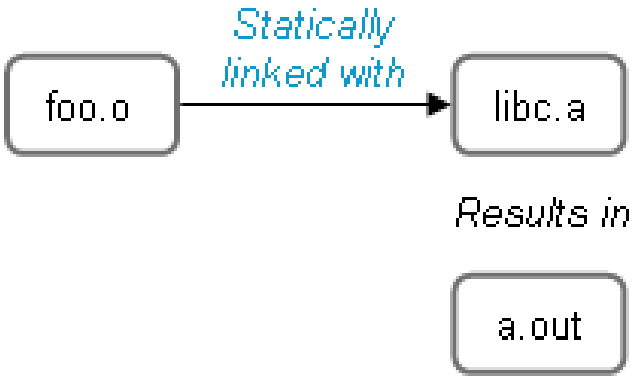
Linker XII


Static linking
Program X
Static libraries
(*.a)
Program Y
Static libraries
(*.a)
Static linking
at compile-time
Dynamic linking
Program X
Program Y
Shared libraries
(*.so)
Dynamic linking
of shared libraries
at run-time

Linker XIII

Static Linking

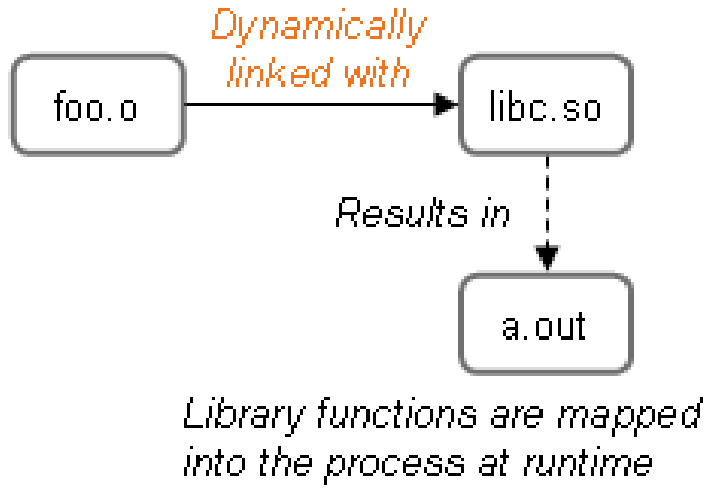
Static linking combines your work with the library into one binary.



The executable is statically linked because a copy of the library is physically part of the executable.

Dynamic Linking

Dynamic linking creates a combined work at runtime.



Library functions are mapped into the process at runtime

The executable is dynamically linked because it contains filenames that enable the loader to find the program's library references at runtime.

Linker XIV

Sl No.	Static linking	Dynamic linking
1.	In static linking, all the library modules are copied to the final executable image. When the program is loaded, OS places only a single file to the memory which contain both the source code and the referencing libraries.	Whereas in dynamic linking only the names of external or shared libraries is placed into the memory. Dynamic linking lets many programs use single copy of executable module.
2.	Static linking is done by the linkers in the final step of the compilation.	Whereas the dynamic linking is done at run time by the OS.
3.	Statically linked files consume more disk and memory as all the modules are already linked.	But in Dynamic linking, only one copy of the reference module is stored which is used by many programs thereby saving memory and disk space.

Linker XV

Sl No.	Static linking	Dynamic linking
4.	In Static linking, if external source program is changed then they have to be recompiled and relinked.	But in case of dynamic linking only a single module needs to be updated and recompiled. Statically linked programs are faster. Statically dynamic slower.
5.	Since the statically linked file contains every package and module, no compatibility issues occur.	Whereas in dynamic linking, since the library files are separately stored there may be compatibility issues (say one library file is compiled by new version of compiler).
6.	Statically linked programs always take constant load time.	Whereas the time is variable in dynamically linked programs.