MET's Institute of Engineering

Bhujbal Knowledge City, Adgaon, Nashik.

Department of Computer Engineering

"LOADER AND LINKER"

Prepared By

Prof. Anand N. Gharu

(Assistant Professor)

Computer Dept.

CLASS : TE COMPUTER 2019

SUBJECT : SPOS (SEM-I)

UNIT : III

27 OCT 2023

CONTENTS :-

- Introduction, Loader schemes:

1. Compile and Go
2. General Loader Scheme
3. Absolute Loaders
4. Subroutine Linkages
5. Relocating Loaders
6. Direct linking Loaders

- Overlay structure,
- Design of an Absolute Loader,
- Design of Direct linking Loader,
- Self-relocating programs,
- Static and Dynamic linking

CONTENTS :-

Sample videos :

1. Preprocessor :

https://youtu.be/JZkPEl8JjZo?list=PLhb7SOmGNUc6Fg7zmBOOS3yN2AG0JiREp

2. Compiler execution stages :

https://youtu.be/cJDRShqtTbk?list=PLhb7SOmGNUc6Fg7zmBOOS3yN2AG0JiREp

Introduction

Computer : A programmable device that can store, retrieve, and process data.(Combination of H/w & S/w)

Hardware : things which we can touch.

Software : things which we cannt touch.(Can only see)

Programming: A programming language is a set of commands, instructions, and other syntax use to create a software program.

Data : Information in a form a computer can use

Information : Any knowledge that can be communicated

Introduction

Machine language : The language, made up of binary coded instructions, that is used directly by the computer

Assembly language : A low-level programming language in which a mnemonic is used to represent each of the machine language instructions for a particular computer

Introduction

Source code : Program or set of instructions written in a high-level programming language

Object code : A machine language version of source code.

Target code : program output in Machine code (binary form)

,

Introduction

Preprocessor :

A preprocessor is a program that processes its input data to produce output that is used as input to another program.

Editor : A text editor is a type of computer program that edits plain text. Such programs are sometimes known as "notepad" software

Compiler : A program that translates a program written in a high-level language into machine code

Assembler : A program that translates an assembly language program into machine code

Introduction

Loader:-

• A loader is a program used by an operating system to load programs from a secondary to main memory so as to be executed.

Linker :

A linker is a computer program that takes one or more object files generated by a compiler and combines them into one, executable program

Debugger :

Debugger is program which is used to test program or execute program in single step execution.

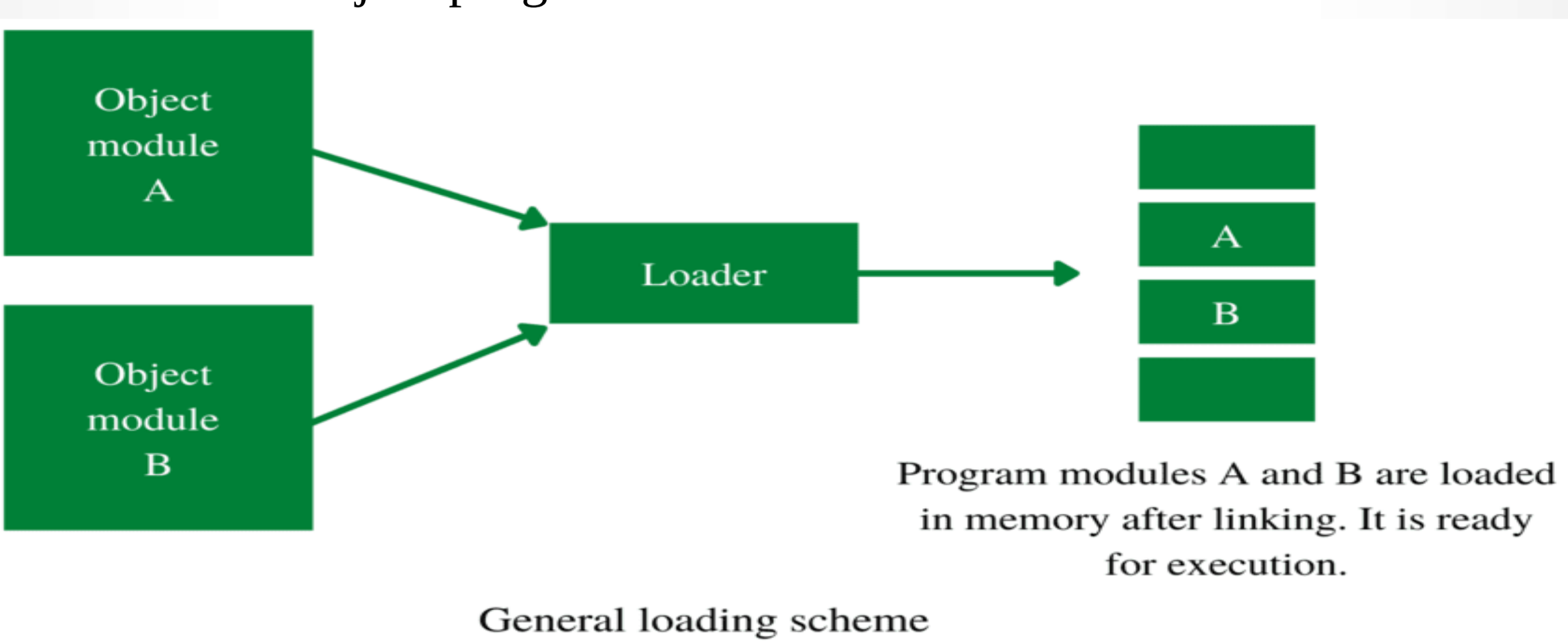
INTRODUCTION TO LOADER

- **Object code, or an object file,** is the representation of code that a compiler or assembler generates by processing a source code file.
- **A linker** is typically used to generate an executable file by linking object files together.
- **Object files** often also contain data for use by the code at runtime, relocation information, program symbols (names of variables and functions) for linking and/or debugging purposes, and other debugging information.

Introduction to Loader

INTRODUCTION TO LOADER

The loader loads the program into the main memory for execution of that program. It loads machine instruction and data of related programs and subroutines into the main memory**, this process is known as loading**. The loader performs loading; hence, the assembler must provide the loader with the object program



General loading scheme

FUNCTION OF LOADER

1. **Allocation:** It allocates memory for the program in the main memory.
2. **Linking:** It combines two or more separate object programs or modules and supplies necessary information.
3. **Relocation:** It modifies the object program so that it can be loaded at an address different from the location.
4. **Loading:** It brings the object program into the main memory for execution.

Loader

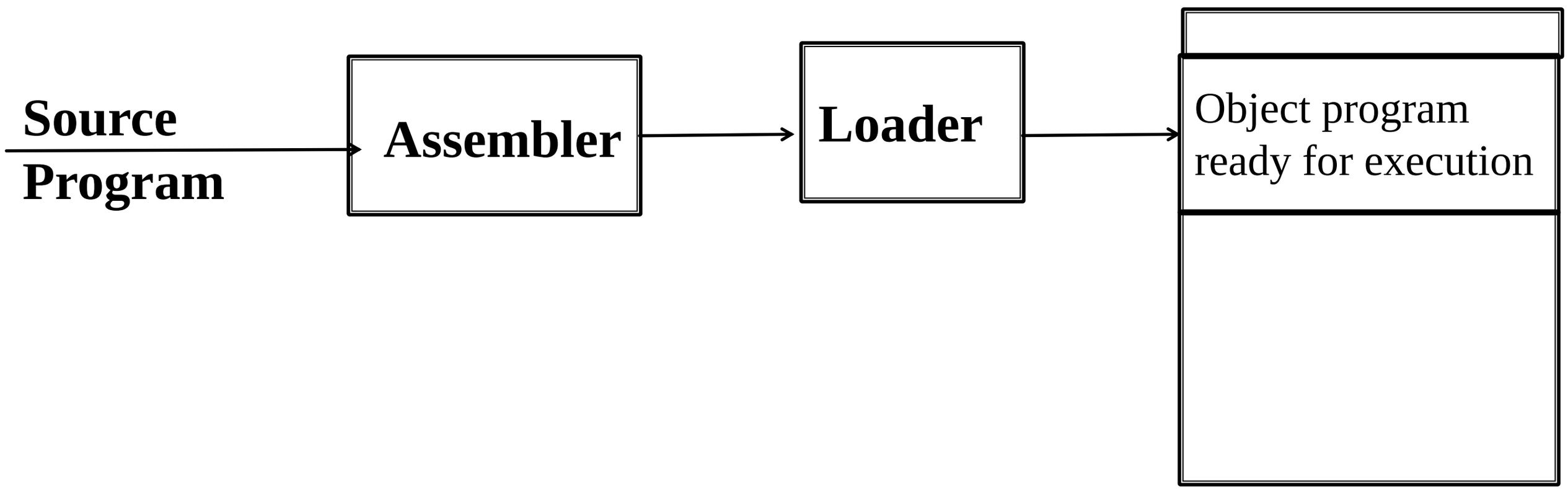


Fig.: Role of loader

Loader

- Loader is **utility program** which takes object code as input **prepares** it **for execution** and **loads** the executable code into the memory.

- Thus loader is actually **responsible** for initiating the executions process.

- Relocation and linking concepts

Linking concepts

- User defined function and library function
- **Example -** printf() ,scanf()

- The linking process **makes** address of modules known to each other so that transfer of control takes place during execution.

- Passing of parameter is handled by the linker.

- **Public variable-**same address

- **External variable** –defined in one module and used in another module.

- **Resolving of addresses of symbolic reference is handled by the linker**.

Linking concepts…

- **Linker-**

1.Handled the passing of parameter

2.Resolving of address of symbolic reference

Relocation concepts

- It is the process of modifying the addresses used in the address sensitive instructions of a program
- So, program Can execute correctly from any designated area of memory.
- **Ex. MOVER AREG,X**
- **Ex.:-Program A, call function F1();**
- Program A and F1() must be linked with each other.

Relocation concepts

Case I: Address assigned to Prog. A and F1 () when they translated to memory

Drawback- a lot of storage area is wasted

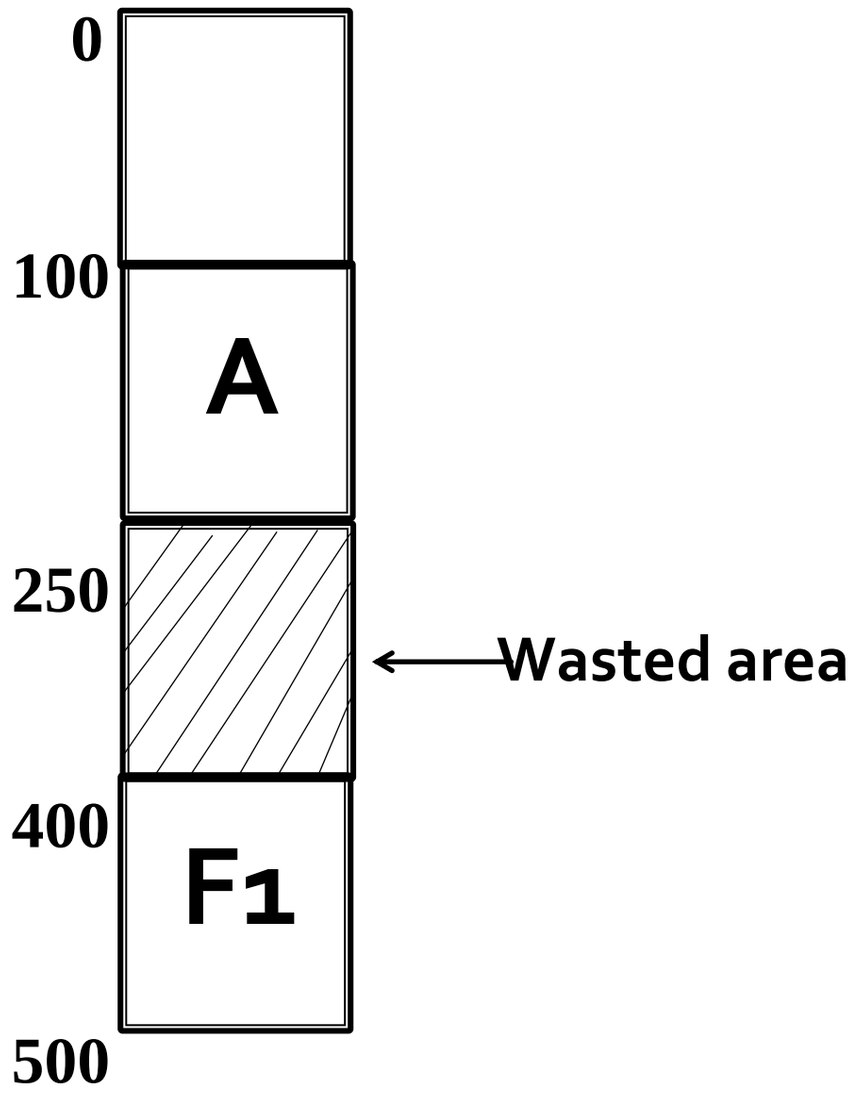


Fig.: General Loading scheme

Relocation concepts…

Case II: These two module cannot co-exist at same storage locations.

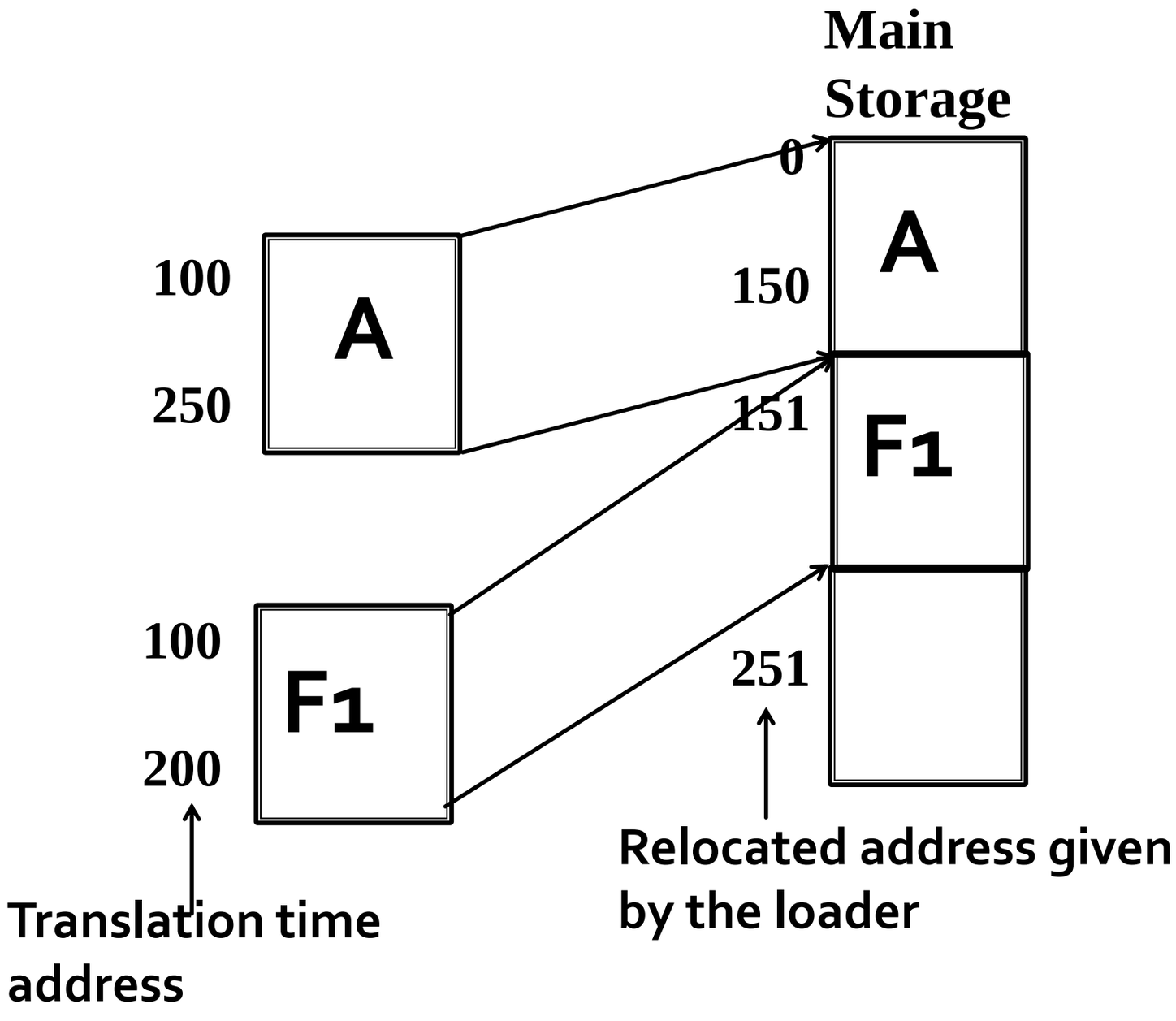
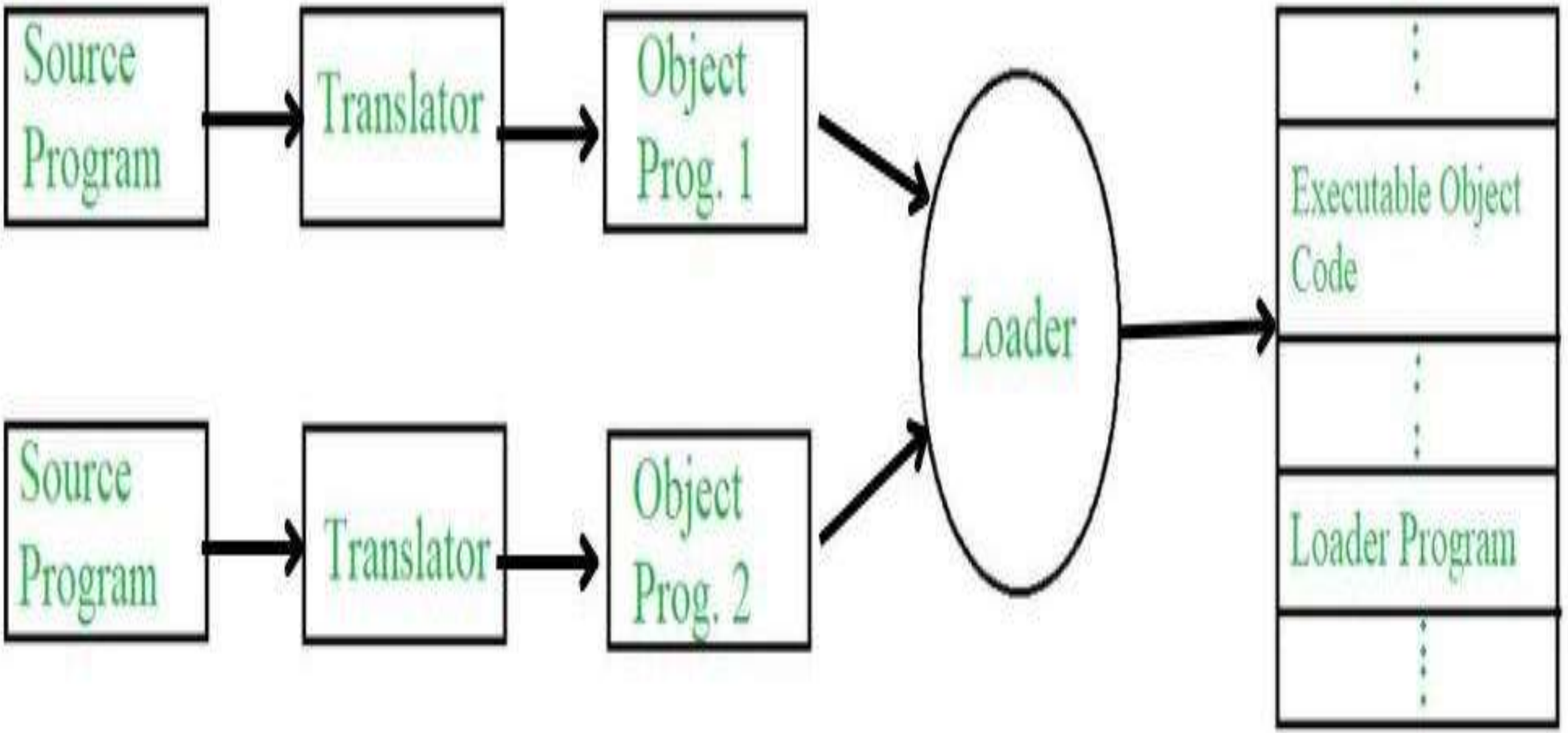


Fig.: Relocation to avoid address conflict or storage waste

Relocation concepts…

- A loader must relocate A and F1 to avoid address conflict or storage waste.
- Relocation refers **to adjustment of address field** not to movement of a program.

ARCHITECTURE OF LOADER



Architecture of Loader

ARCHITECTURE OF LOADER

- **Source program:** This is a program written in a high-level programming language that needs to be executed.
- **Translator:** This component, such as a compiler or interpreter, converts the source program into an object program.
- **Object program:** This is the program in a machine-readable form, usually in binary, that contains both the instructions and data of the program.
- **Executable object code:** This is the object program that has been processed by the loader and is ready to be executed.

ADVANTAGES OF LOADER

- **Memory management** – allow separate/protected area
- **Dynamic Linking** – handle external references
- **Relocation** – to avoid conflict with other program
- **Error handling** – missing libraries, incompatible instruction
- **Modularity –** easier to maintain and update
- **Reusability** – same module/libraries can be used by other program also

DISADVANTAGES OF LOADER

- **Complexity** – implementation difficult due to memory management, relocation etc.
- **Overhead** – take more time to process
- **Limited flexibility** – may not be easily portable
- **Security** – poor design may introduce vulnerability
- **Dependency issues** – may depend on external libraries

Loading Schemes

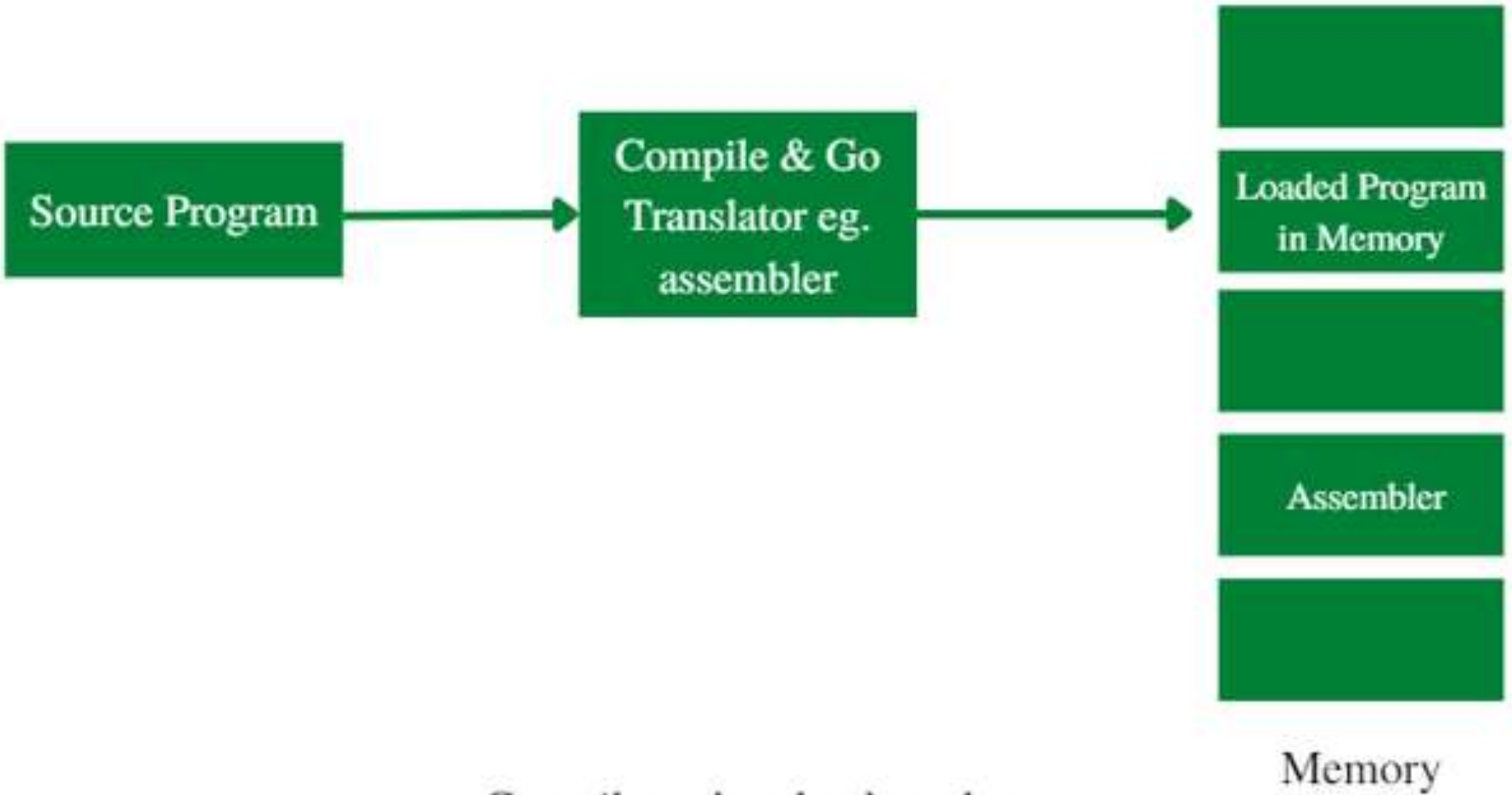
Loading Schemes…

Various types of loader ,based on various functionalities

1. Compile and go loader
2. General loader scheme
3. Absolute loader
4. Subroutine linkage
5. Relocating loader
6. Direct linking loader

COMPILE AND GO LOADER

Compile and Go Loader :



Compile and go loader scheme

COMPILE AND GO LOADER

Performing the loader function is to have the assembler run in one part of memory and place the assembled machine instructions and data, as they are assembled, directly into their assigned memory location. It is also called "assemble and go loader"

In this scheme, the source code goes into the translator line by line, and then that single line of code loads into memory.

In another language, chunks of source code go into execution. Line-by-line code goes to the translator so there is no proper object code. Because of that, if the user runs the same source program, every line of code will again be translated by a translator. So here re-translation happens.

ADVANTAGES COMPILE AND GO LOADER

1. It is very simple to implement.
2. The translator is enough to do the task, no subroutines are needed.
3. It is the most simple scheme of the functions of the loader
4. It provides security

Disadvantages:

1. There is no use of the assembler but it is still there so a wastage of memory takes place.
2. When source code runs multiple times the translation is also done every time. so re-translation is happening.
3. Difficult to produce an orderly modular program
4. Difficult to handle multiple segments like if the source program is in a different language. eg. one subroutine is assembly language & another subroutine is FORTRAN

GENERAL LOADER SCHEME

General Loader Scheme :

In this loader scheme, the source program is converted to object program by some translator (assembler). The loader accepts these object modules and puts the machine instruction and data in an executable form at their assigned memory. The loader occupies the same portion of main memory.

2. General Loading Scheme

Program modules A and B are loaded in memory after linking. It is ready for execution

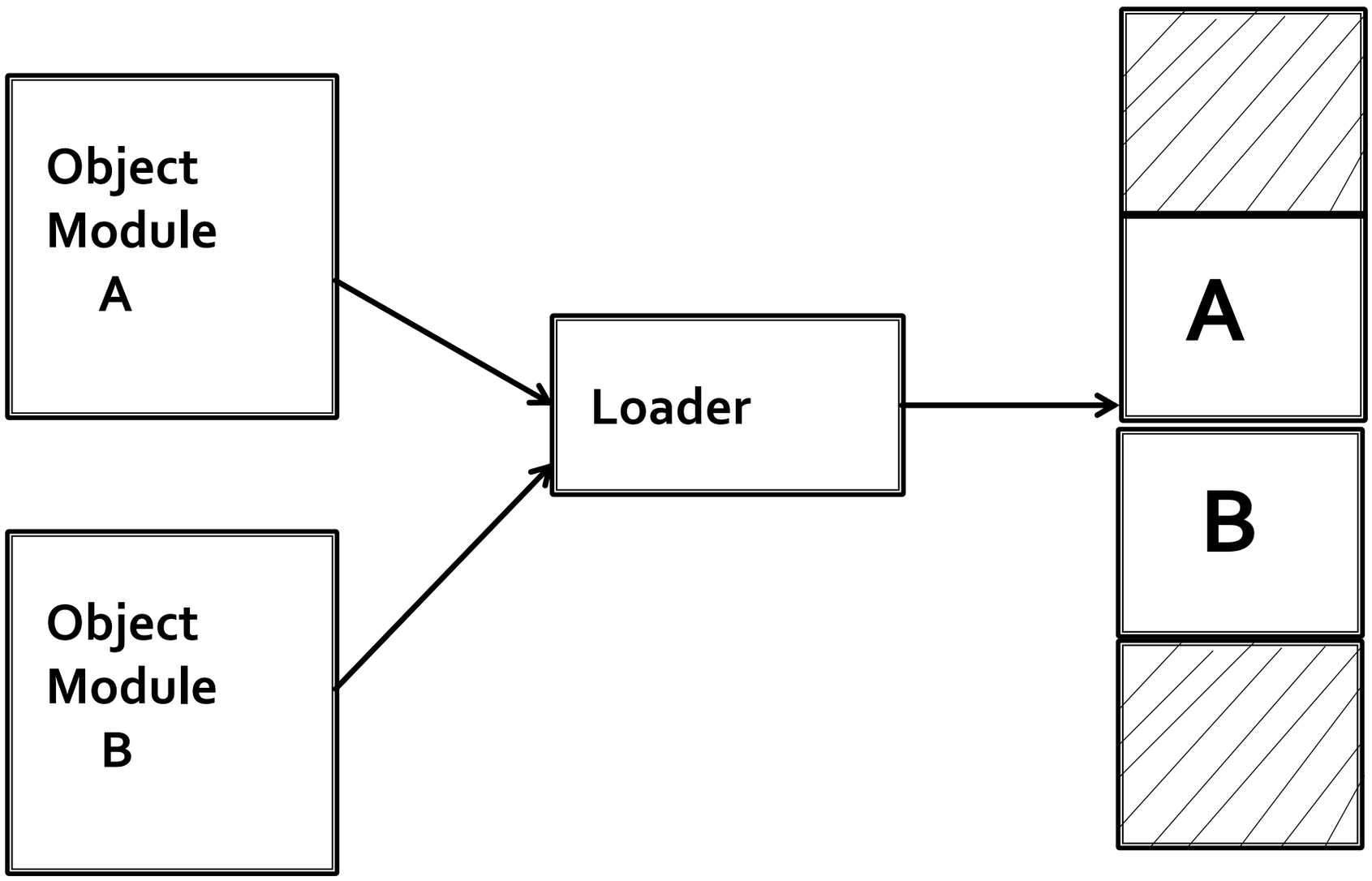
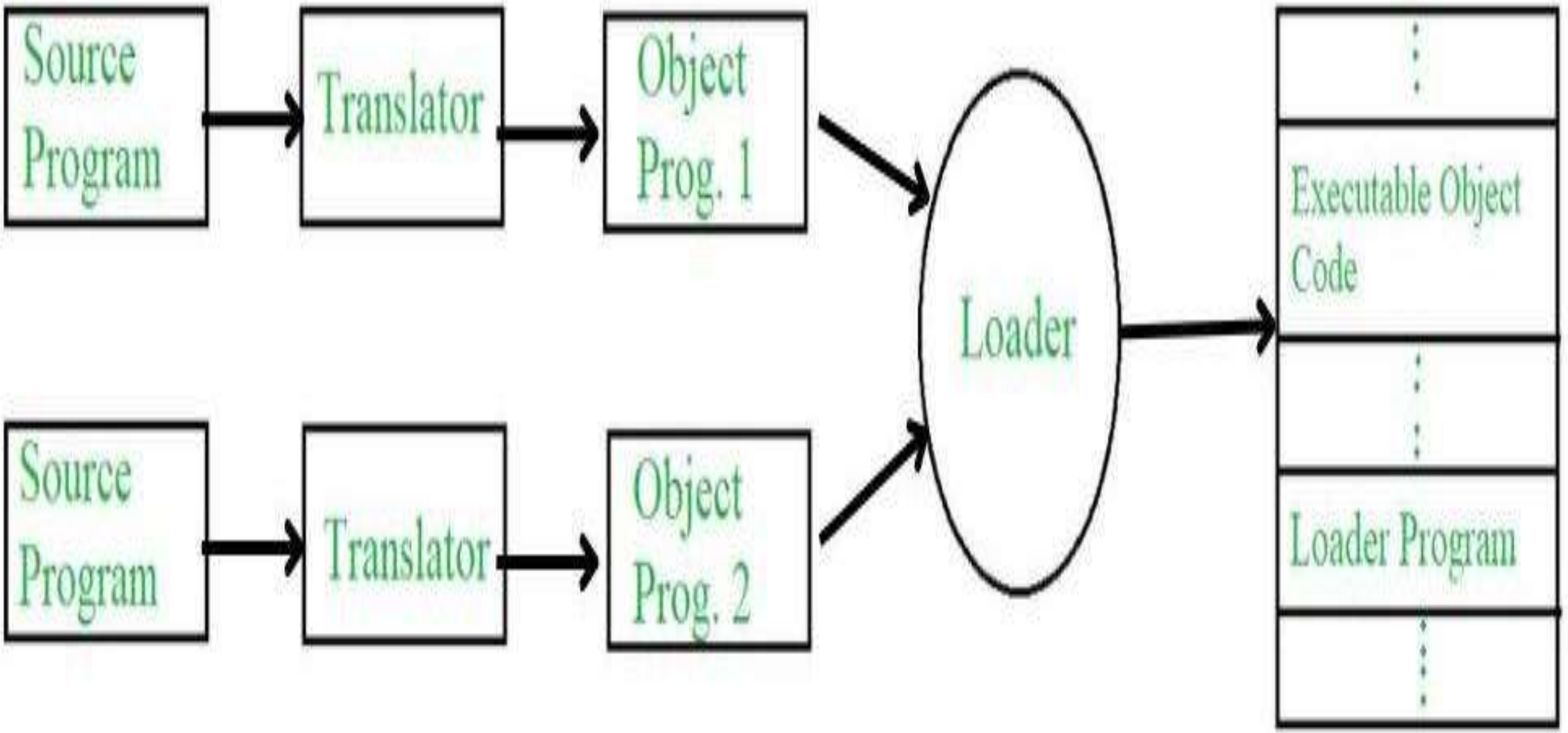


Fig.: General Loading scheme

GENERAL LOADER SCEMES



Architecture of Loader

GENERAL LOADER SCEMES

- **Source program:** This is a program written in a high-level programming language that needs to be executed.
- **Translator:** This component, such as a compiler or interpreter, converts the source program into an object program.
- **Object program:** This is the program in a machine-readable form, usually in binary, that contains both the instructions and data of the program.
- **Executable object code:** This is the object program that has been processed by the loader and is ready to be executed.

ADVANTAGES OF GENERAL LOADER

1. Smaller than Assembler
2. No reassembly is needed
3. Possible to write subroutines in different languages
4. There is no wastage of memory because assembler is not placed in memory so more memory is available to user.
5. Avoid drawbacks of Compile and Go Loader.

DISADVANTAGES OF GENERAL LOADER

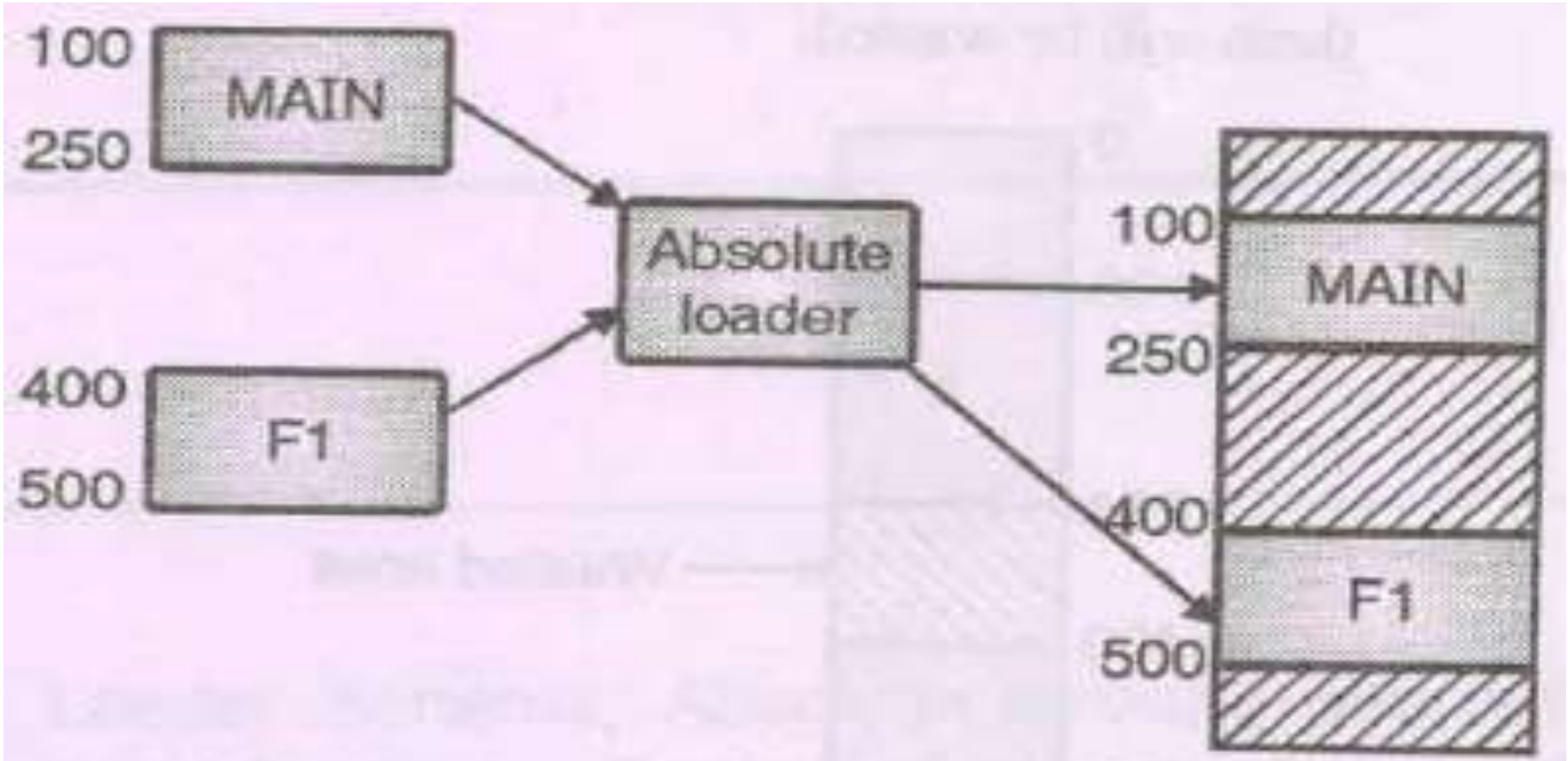
- Some portion of memory occupied by loader
- General loader cannot handle different object model from other computer.
- Dependenacy issues

ABSOLUTE LOADER

Absolute Loader: It loads a program at a specific memory location, specified in the program's object code. This location is usually absolute and does not change when the program is loaded into memory.

- An Absolute loader is the simplest of all other loaders.
- It takes the output of Assembler and load into memory without relocation.
- The output of the assembler can be stored on any machine readable form of storage, But most commonly it is stored on punched cards or magnetic tape, disk, or drum.
- It loads a binary program in memory for Execution .

3.Absolute Loader



(S5.4)**Fig. 3.2.2 : Absolute loader example**

3.Absolute Loader....

- Binary program is stored in a file that contains:
- Header records: Contains load Origin ,Length of code, load time execution starting address of program.
- Transfer record: contains entry point of execution

ADVANTAGES OF ABSOLUTE LOADER

1. It is simple to implement
2. No relocation is required.
3. The process of execution is efficient.
4. The task of loader becomes simpler as it simply obeys the instruction regarding where to place the object code to the main memory.
5. This scheme allows multiple programs or the source programs written in different languages.

DISADVANTAGES OF ABSOLUTE LOADER

1. Programmer must specify the starting address to the assembler for the program where it should be loaded.
2. so programmer must know memory management as well as memory status at any time.
3. It is very difficult to relocate in case of multiple subroutine.

3.Absolute Loader

- In absolute loader 4 loader function are performed by,
- Allocation: by programmer
- Linking: by programmer
- Relocation: by assembler
- Loading-by loader

SUBROUTINE LINKAGE LOADER

Subroutine: In given program ,it is often needed to perform a particular subtask many times on different data values. such a subtask is usually called a subroutine.

Subroutine linkage method: The way in which a machine makes it possible to call and return from subroutine is referred to as its Subroutine linkage method

4. SUBROUTINE LINKAGE

A program consisting of main program and a set of functions (subroutines) could reside in several files. These program units are assembled separately.

The problem of subroutine linkage is this : a main program A wishes to call the subroutine B and if the subroutine B resides in another file then the assembler will not know the address of B and declare it as an undefined symbol.

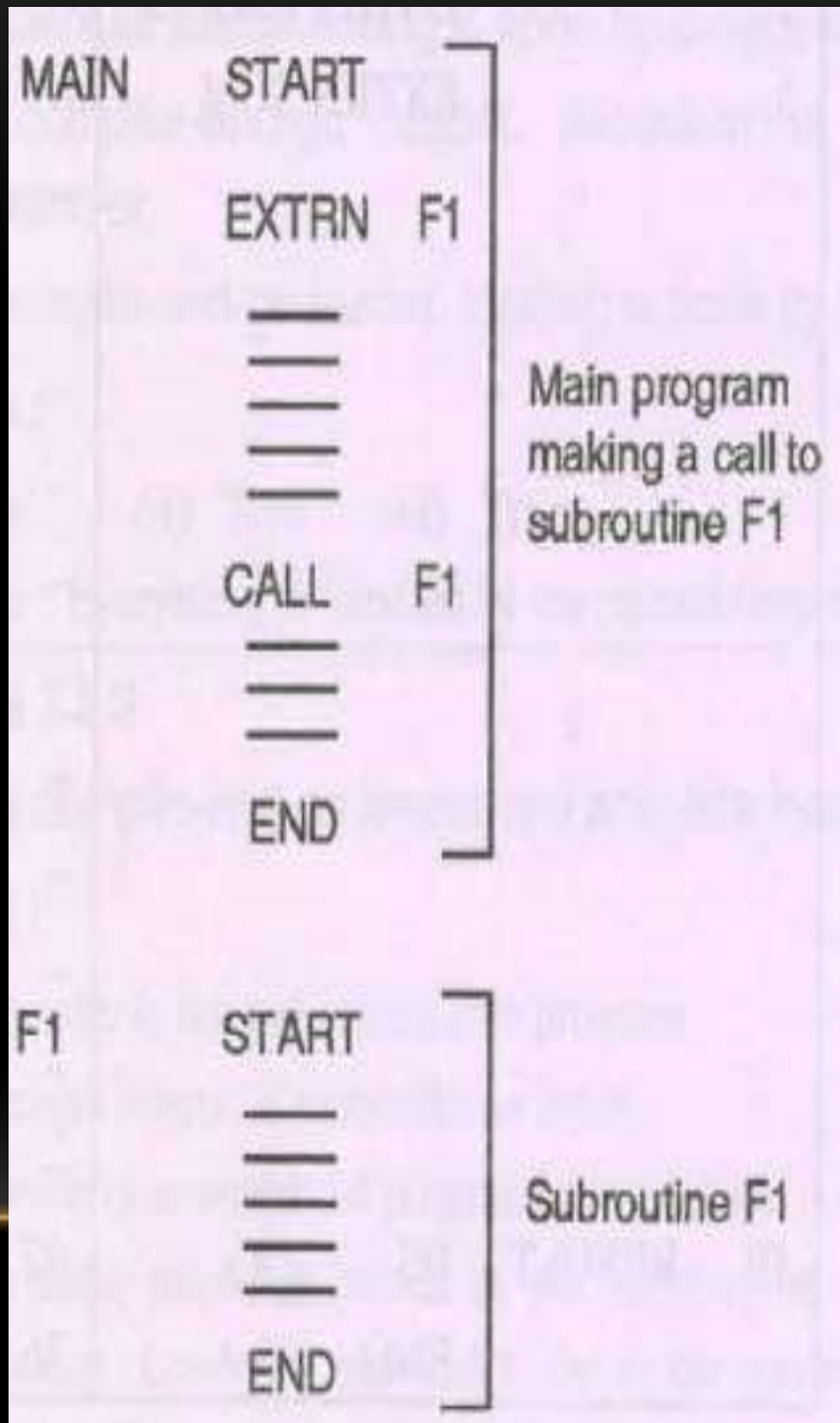
To realize such interactions, A and B must contain public definitions and external references.

EXTRN statements

The EXTRN statement lists the symbols to which external references are made in the current program unit. These symbols are defined in other program units.

ENTRY statements

The ENTRY statement lists the public definitions of a program unit, i.e. it lists those symbols defined in the program unit which may be referenced in other program units.



ADVANTAGES OF SUBROUTINE LINKAGE

1. **Code reuse:** Subroutines can be reused in multiple parts of a program, which can save time and reduce the amount of code that needs to be written.
2. **Modularity:** Subroutines help to break complex programs into smaller, more manageable parts, making them easier to understand, maintain, and modify.
3. **Encapsulation:** Subroutines provide a way to encapsulate functionality, hiding the implementation details from other parts of the program.

DISADVANTAGES OF SUBROUTINE LINKAGE

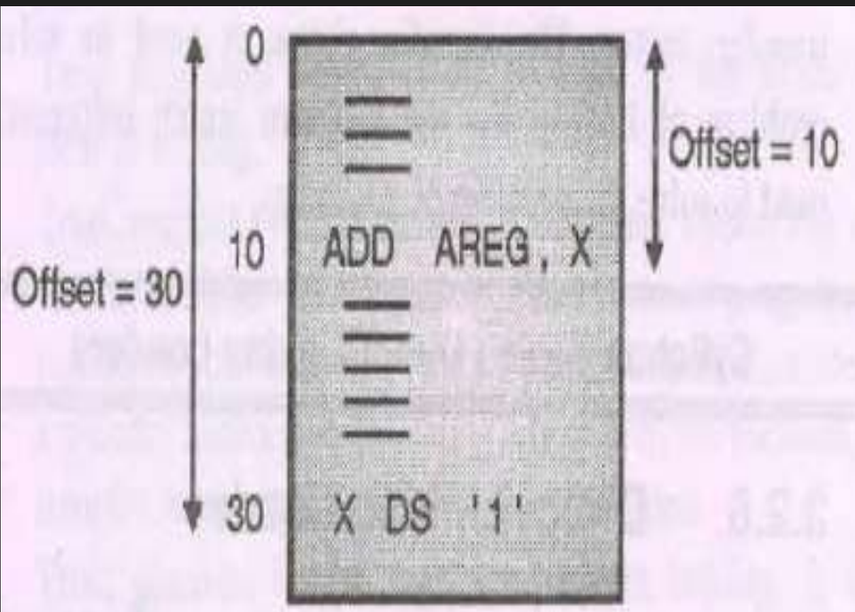
1. **Overhead:** Calling a subroutine can incur some overhead, such as the time and memory required to push and pop data on the stack.
2. **Complexity:** Subroutine nesting can make programs more complex and difficult to understand, particularly if the nesting is deep or the control flow is complicated.
3. **Side Effects:** Subroutines can have unintended side effects, such as modifying global variables or changing the state of the program, which can make debugging and testing more difficult.

RELOCATING LOADER

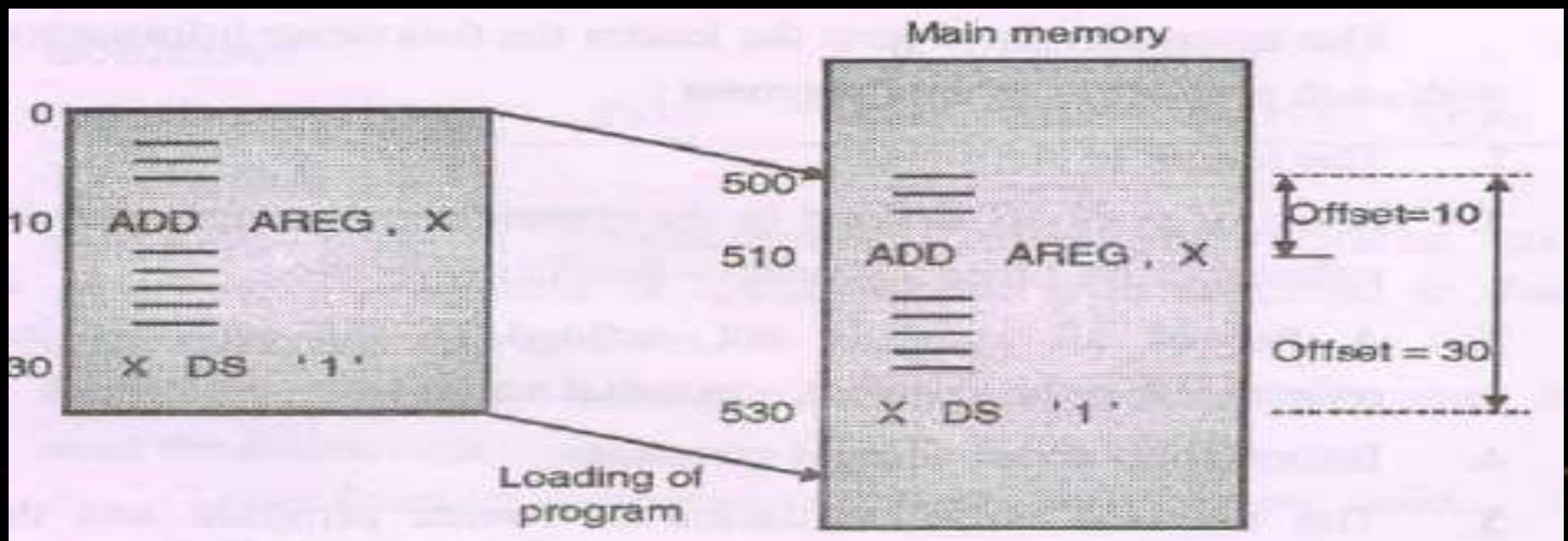
A relocating loader is a type of loader that is used in **system software** to load programs into memory and adjust their addresses. It is responsible for **dynamic loading, address translation,** and using a symbol table to adjust the addresses of a program's instructions and data.

5. RELOCATING LOADER

- Binary symbolic loader (BSS) is an example of relocating loader.
- The BSS loader allows many code segments but only one data segment.
- The output of the assembler using a BSS loader is :
  1. Object program
  2. Reference about other programs to be accessed.
  3. Information about address sensitive entities.



(S5.6)**Fig. 3.2.4 : A sample program segment considered for relocation**



(S5.7)**Fig. 3.2.5 : Relocation of the program**

5. RELOCATING LOADER

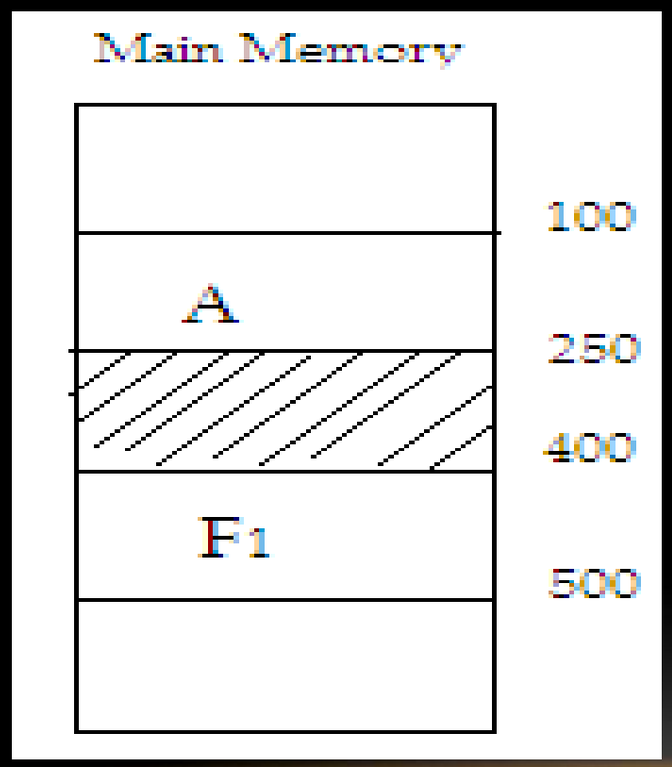
Relocation is the process of modifying the address used by program such that program can execute correctly.

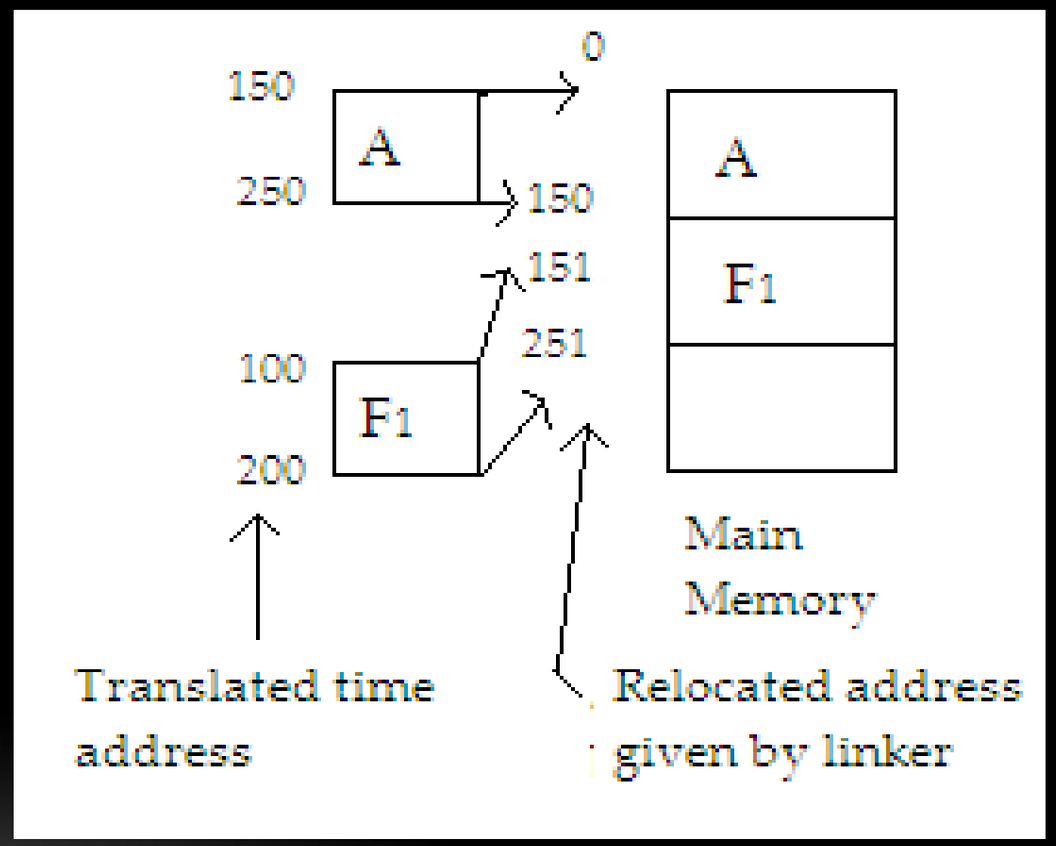
E.g. assume that a program A calls a function F1. The program A and the function F1 must be linked with each other. But where we have to load in main storage? A possible solution would be to load them according to address assigned when they were translated.

Case (I) At the time of translation A has been given storage area from 100 to 250 while F occupies area between 400 to 500. If we were to load these programs at their translated address , a lot of storage will be wasted.

5. RELOCATING LOADER

Case (II) at the time of translated, both A and F1 may have been translated with the identical start address 100. A goes from 100 to 250 and F1 goes from 100 to 200. These two modules cannot co-exist at same. The linker must relocate A and F1 to avoid address conflict or storage waste A possible relocation is shown in figure.





It may be noted that relocation more than simply moving a program from one are to another in the storage. It refers to adjustment of address fields and not to movement of a program.

Relocating Loaders :Advantages

- It avoid reassembling of all subroutines when single subroutine change.
- All 4 function are performed (i.e. Allocation, Loading, Linking & reallocation).
- Transfer vector is used to solve the problem of linking & Program length info to solve allocation.

Relocating Loaders :DisAdvantages

- No suited for loading external data.
- Transfer vector increase the size.
- Does not facilitate access to data segment that can be shared.

6. DIRECT LINKING LOADER

It is a general relocatable loader, and is perhaps the most popular loading scheme presently used.

- It is a relocatable loader
- It allows multiple procedure segments and multiple data segments.

The assembler must give the loader the following information with each procedure or data segment :

1. The length of segment.
2. A list of symbols defined in the current segment that may be referenced by other segments – public declaration.
3. A list of all symbols not defined in the segment but referenced in the segment – external variables.
4. Information about address constants
5. The machine code translation of source program and the relative addresses assigned.

The object module produced by the assembler is divided into 4 sections :

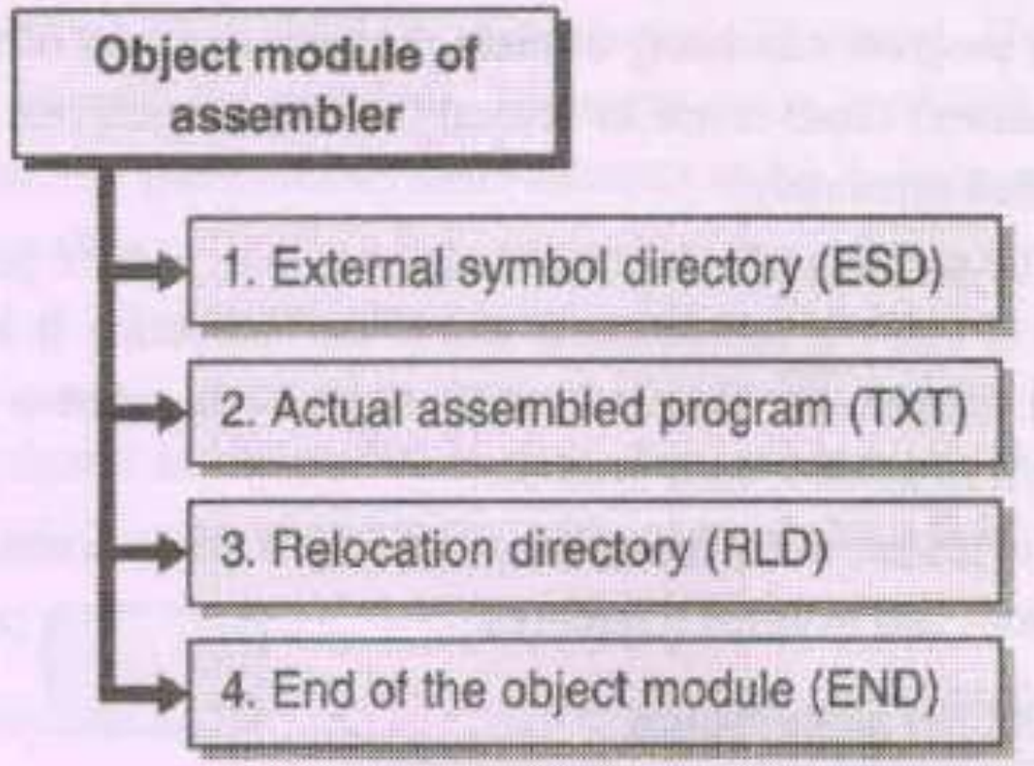


Fig. C3.2 : Object module of assembler

				LC
1.	MAIN	START		0
2.		ENTRY RESULT		–
3.		EXTRN SUM		–
				–
				–
				–
				–
				–
				–
				–
				–
				–
10.	RESULT	DS	4	32
		END		36

Fig. 3.2.6(a) : A sample source program

Line No.	Symbol	Type	Relative location	Length
1.	MAIN	SD	0	36
2.	RESULT	LD	32	–
3.	SUM	ER	–	–

Fig. 3.2.6(b) : ESD for program shown in Fig. 3.2.6(a)

SD – symbol is a segment definition

LD – symbol is defined in this program but it can be referenced by other programs

ER – symbol is an external reference. It is defined in some

Direct linking loader:

- It is type of Re-locatable Loader.
- It is most common type of loader.
- It allows the programmer to use multiple procedure segments and multiple data segments.
- The assembler should give the following information to the loader:

1. The length of the object code segment.
2. A list of all symbols which are not defined in the current segment but can be used in the current segment.

Direct Linking Loader...

3. A list of all symbols which are defined in the current segment but can be referred in the current segment.
4. Information about address constants.
5. Machine code translation of the source program and relative address.

- To place the object code in the memory there are two situations:1. Address of the object code could be absolute.2. The address of object code can be relative.

Direct linking loader:

- The list of symbols not defined in the current segment but used in the current segment are stored in a data structure called ***USE table.***
- The lists of symbols defined in the current segment and referred by the other segments are stored in a data structure called ***DEFINITION table.***

Assembler records

- Assembler generates 4 types of cards in the object desk:
- ESD:External Symbol Dictionary (ESD) record:
- TXT: (TXT) records.
- RLD: Relocation and Linkage Directory (RLD):
- END :End of object deck

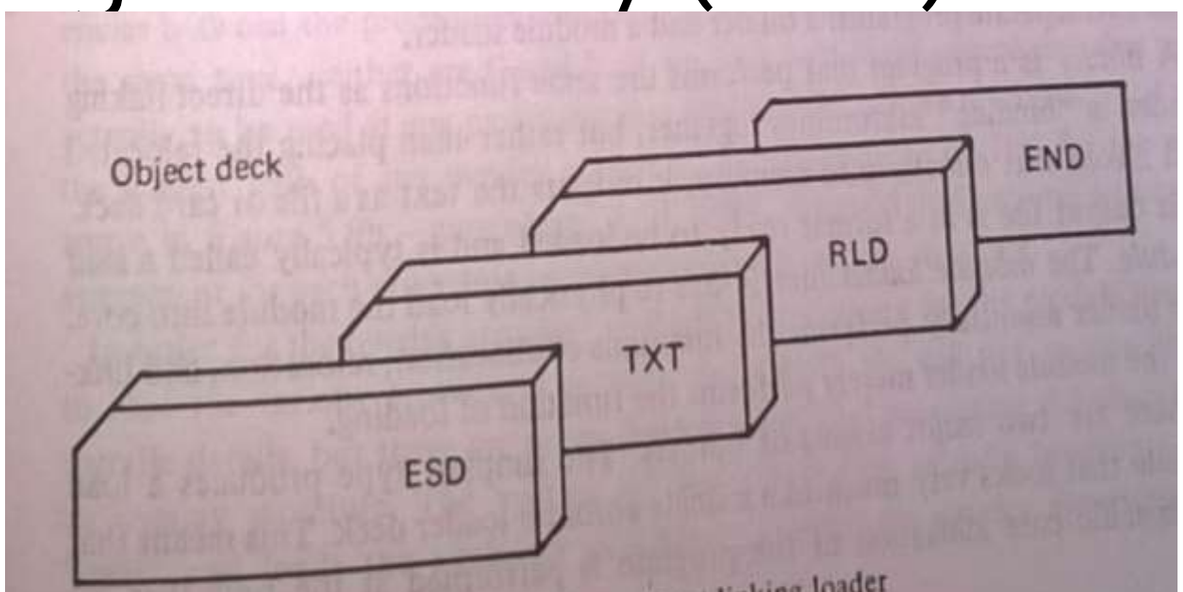


FIGURE 5.8 Example object deck for a direct-linking loader

Assembler records…ESD

1.ESD:External Symbol Dictionary (ESD) record: Card contain information about all symbol that are defined in this program, but that may reference elsewhere, and all symbol referenced in this program but defined elsewhere.

Source object record no./card no.	symbol	Type	Relative Location	Length
1	MAIN	SD	0	36
2	RESULT	LD	32	-
3	SUM	ER	-	-

Assembler records…ESD

- **Type:**
- **SD**: segment definition: symbol in segment
- **LD**: Local definition: symbol is defined in this program but it can be referenced by other program.
- ER: External symbol: defined in some external program.

Assembler records…TXT

- 2. TXT: (TEXT) records.
- Text card contains actual object code.(translated source code).

Source object record no.	Relative Location	Object code
1	0	
2	32	
3	-	

Assembler records…RLD

3. RLD: Relocation and Linkage Directory (RLD): Card contain information about those locations in the program whose content depend on the address at which the program is placed

Source object record no.	ESD_ID	Length(in Byte)	Flag + or -	Relative address

+ sign denote that something must be added to the constant

Assembler records…END

4.END: card indicate the end of object code and specifies the starting address for execution if assembled routine is the main program.

Advantages of Direct Linking

- Allow multiple procedure and data segment.
- Allow independent translation of the program.
- Relocation Facility.

Disadvantages of Direct Linking

- Not suitable in multitasking.
- It is necessary to allocate, relocate, link, and load all of the subroutines each time in order to execute a program
- loading process can be extremely time consuming.

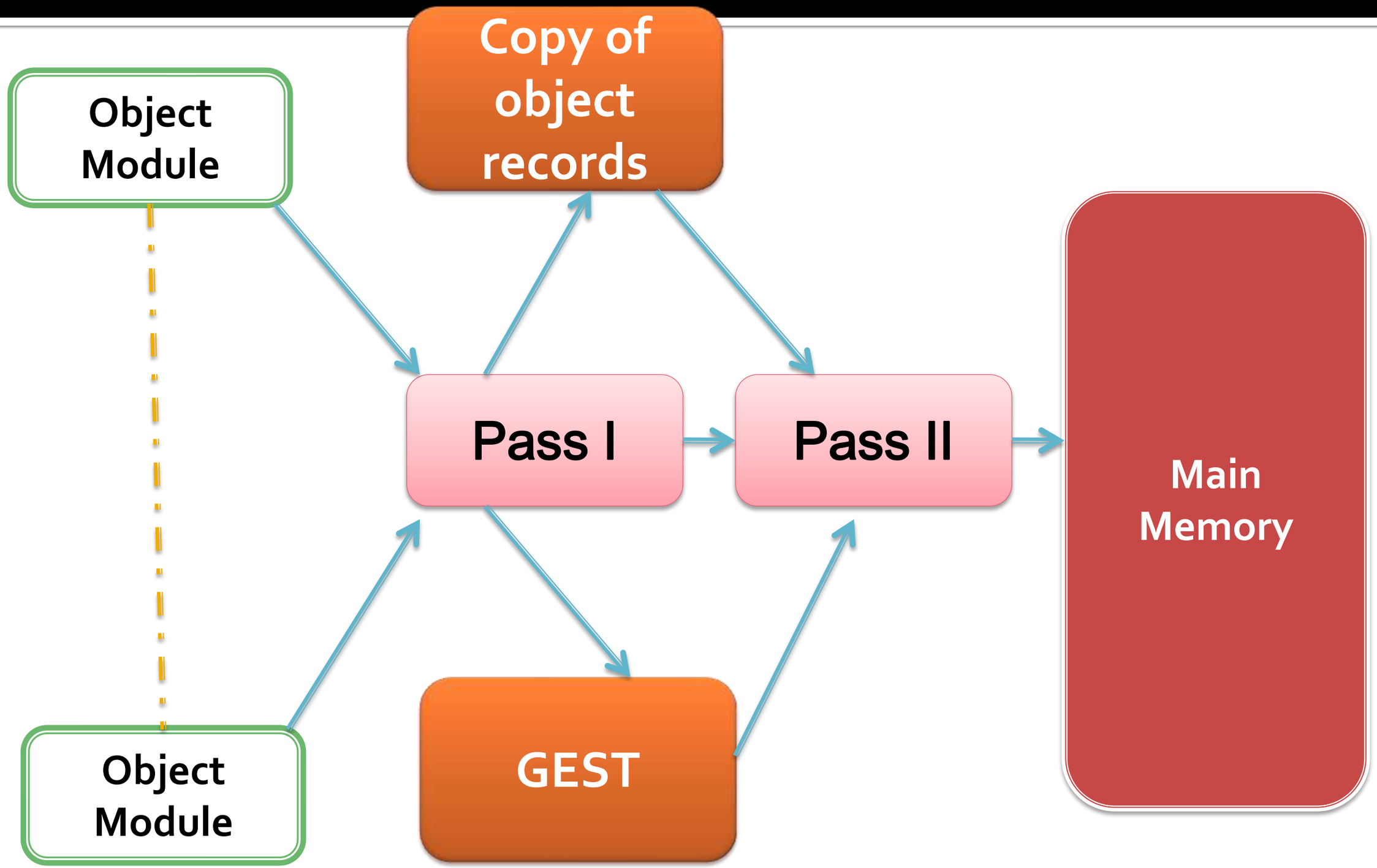
Design of Direct Linking Loader

- Design of direct linking loader is more complicated than absolute loader.
- Input to loader is object module and this is divide into 4 section ESD,TXT,RLD,END
  - It requires **two passes** to complete the linking process
- Pass I: assigns addresses to all external symbols (Uses Global EST: GEST)
- Pass II: performs actual loading, relocation and linking.

Design of Direct linking loader…

- In Pass I ,a Global External Symbol table(GEST) is prepared.
- It contain every external symbol and corresponding absolute address value.

Fig. Two pass direct linking loader scheme



OVERLAY STRUCTURE

Overlays: The loader can load programs in chunks called overlays, which allows programs to run in a smaller memory space by only loading the needed parts of the program at a time.

OVERLAY STRUCTURE

- Program execution need not be achieved by loading all the parts of large program in memory.
- So,that we can reduce memory requirement of such program by loading required parts of program in memory.
- Hence some parts of program are given the same load address during linking.
- Therefore,at any time,only one of these parts of program can be loaded in memory because loading of other parts that has same load address will overwrite it.

OVERLAY STRUCTURE

- Overlay is a part of program that has same load origin as some other parts of the program .and the program which contains overlays is called as overlay structured program.
- It consist of Permanently resident portion known as ROOT.
- Set of overlays that will be loaded in memory as per the requirements.
- Overlay manager is linked with the root.
- Root is loaded in memory .

OVERLAY STRUCTURE

EXAMPLE

- Suppose a program consisting of five subprograms (A{20k},B{20k}, C{30k}, D{10k}, and E{20k}) that require 100K bytes of core.
  - Subprogram A only calls B, D and E;
  - subprogram B only calls C and E;
  - subprogram D only calls E
  - subprogram C and E do not call any other routines
- Note that procedures B and D are never in used the same time; neither are C and E.

OVERLAY STRUCTURE

Consider a program containing 5 subroutines A,B,C,D,E. which require total 100kb memory.

Subprogram A calls B,D &E.

Subprogram B calls C & E

D call E

C and E do not call any other program.

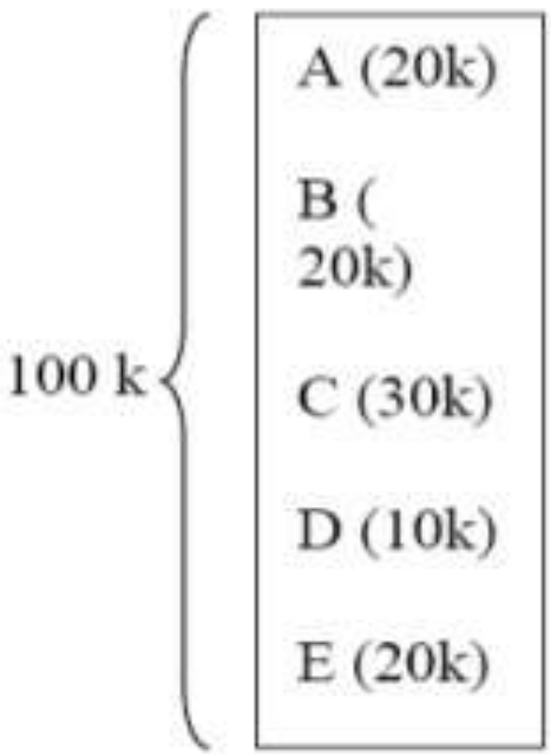


Fig. (a)

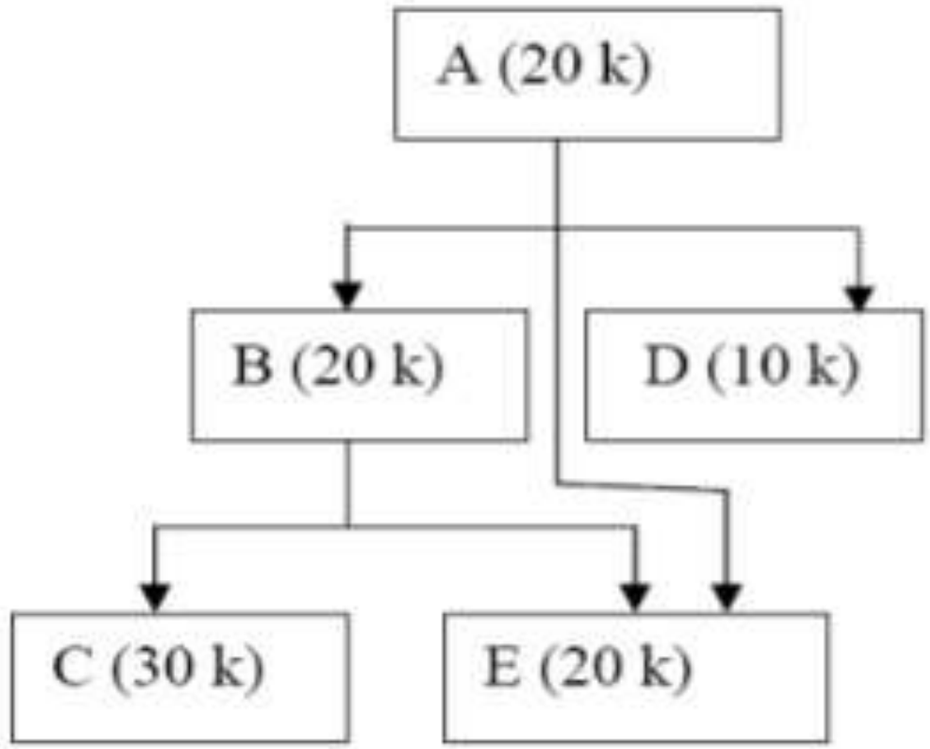


Fig. (b) Overlay Structure

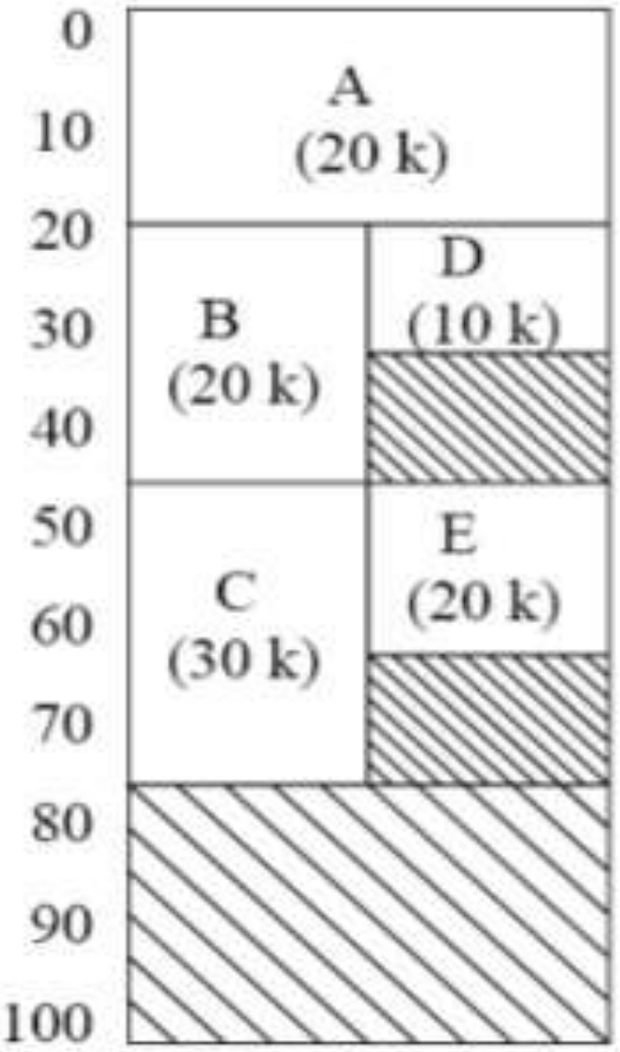


Fig. (c) Possible storage assignment of each procedure

Fig. 2.9

ADVANTAGES OF OVERLAY STRUCTURE

1. **Increased memory utilization:** Overlays allow multiple programs to share the same physical memory space, increasing memory utilization and reducing the need for additional memory.

2. **Reduced load time:** Only the necessary parts of a program are loaded into memory, reducing load time and increasing performance.

3. **Improved reliability:** Overlays reduce the risk of memory overflow, which can cause crashes or data loss.

4. Reduce memory requirement

5. Reduce time requirement

DISADVANTAGES OF OVERLAY STRUCTURE

1. **Complexity:** Overlays can be complex to implement and manage, especially for large programs.
2. **Performance overhead:** The process of loading and unloading overlays can result in increased CPU and disk usage, which can slow down performance.
3. **Compatibility issues**: Overlays may not work on all hardware and software configurations, making it difficult to ensure compatibility across different systems.
4. Overlap map must be specified by programmer
5. Programmer must know memory requirement
6. Overlapped module must be completely disjoint
7. Programming design of overlays structure is complex and not possible in all cases

What is linker? Why program relocation is required and how is it performed?

Any usable program written in any language has to use functions / subroutines. These functions could be either user defined functions or they can be library functions.

For example, consider a program written in C language such a program may contain calls to functions like printf( ). During program execution main program calls the function

1) The linking process makes address of modules known to each other so that transfer of control takes place.

2) Passing of parameters is handled by the linker.

3) An external variable can be defined in one module and can be used in another module.

SELF RELOCATING PROGRAM

Program relocatibility refers to the ability to load and execute a given program into an arbitrary place in memory as opposed to a fixed set of locations specified at program translation time depending on how and when the mapping from virtual address space to the physical address space takes place in given relocation: (a) Static (b) Dynamic .

A self relocating program is a program which can perform the relocation itself. Self relocating program contain the relocating logic, so no need of a linker in that.

STATIC & DYNAMIC LINK LIBRARIES

1. Static Linking :

A static linker takes object files produced by the compiler including library functions and produces and executable file. The executable file contains a copy of every subroutine (user defined or library function.) The biggest disadvantage of the static linking is that each executable file contains its own copy of the library routines. If many programs containing same library routines are executed then memory is wasted.

STATIC & DYNAMIC LINK LIBRARIES

2. Dynamic Linking :

Dynamic linking defers much of the linking process until a program starts running. Dynamic linking involves the following steps:

1) A reference to an external module during run time causes the loader to find the target module and load it.

2) Perform relocation during run time

Dynamic linking permits a program to load and unload routines at run time.

STATIC & DYNAMIC LINK LIBRARIES

Static Linking	Dynamic Linking
Copies all the libraries your program will need directly into the final executable file	External or shared libraries are copied into the executable file by name right at the point of run time
Simplifies the process of distributing binaries to multiple environments or operating systems	Lower maintenance costs and a reduced need for support
Results in a slightly faster start-up depending on program complexity	You can update routines in libraries without needing to relink
Programs can be much larger and use a lot of resources	Leads to smaller file sizes
If any changes occur in external programs, they won't automatically be reflected in your executable file	If any changes are made to a library, you might run into compatibility issues

LINKER VS LOADER

BASIS FOR COMPARISON	LINKER	LOADER
Basic	It generates the executable module of a source program.	It loads the executable module to the main memory.
Input	It takes as input, the object code generated by an assembler.	It takes executable module generated by a linker.
Function	It combines all the object modules of a source code to generate an executable module.	It allocates the addresses to an executable module in main memory for execution.
Type/Approach	Linkage Editor, Dynamic linker.	Absolute loading, Relocatable loading and Dynamic Run-time loading.

THANK YOU!!!

My Blog : https://anandgharu.wordpress.com/

Email : gharu.anand@gmail.com

REFERENCES

1. https://www.geeksforgeeks.org/basic-functions-of-loader/
2. https://www.geeksforgeeks.org/loader-in-compiler-design/
3. https://www.geeksforgeeks.org/compiler-and-go-loader/
4. https://www.geeksforgeeks.org/subroutine-subroutine-nesting-and-stack-memory/
5. https://www.geeksforgeeks.org/overlays-in-memory-management/