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

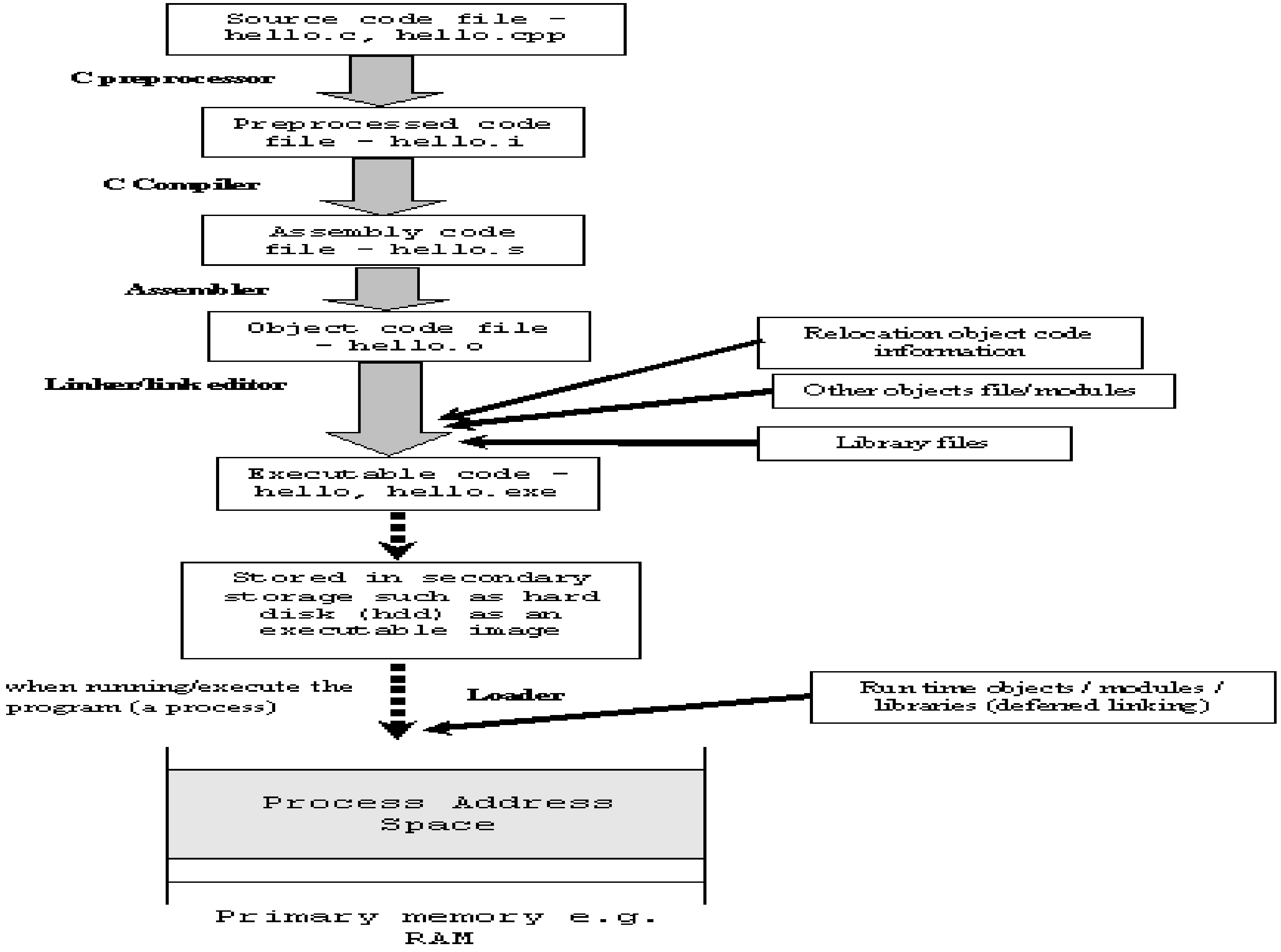
Module 4

Loaders and Linker

Contents as per syllabus

- Introduction
- Functions of loaders
- Relocation and Linking concept
- Different loading schemes
  - Relocating loader
  - Direct Linking Loader
  - Dynamic linking and loading.

Introduction


Source code file – hello.c, hello.cpp
C preprocessor
Preprocessed code file – hello.i
C Compiler
Assembly code file – hello.s
Assembler
Object code file – hello.o
Linker/link editor
Relocation object code information
Other objects file/modules
Library files
Executable code – hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

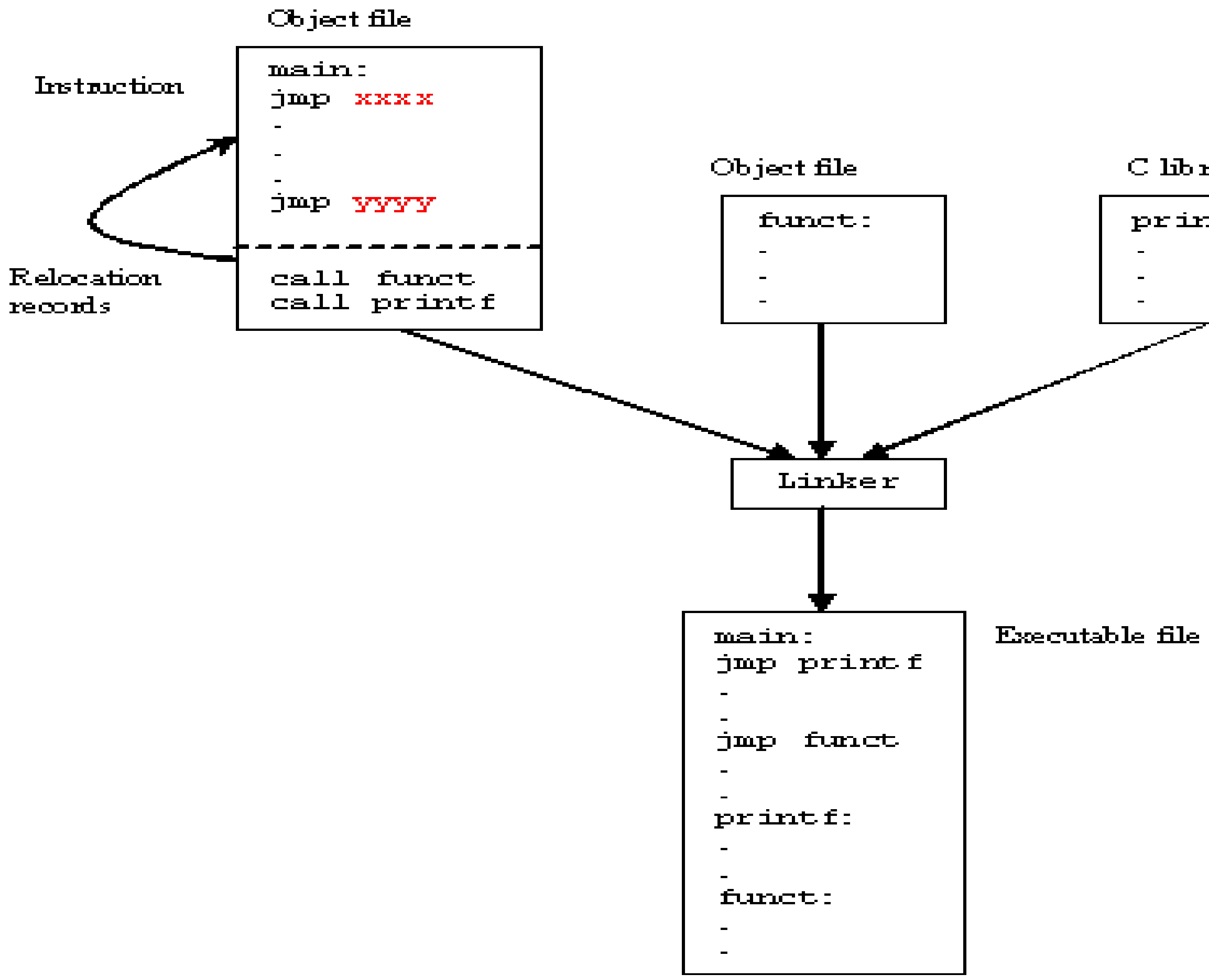
Linkers

- Linker is a computer program, which merges the object files produced by separate compilation or assembly and creates an executable file
- Takes input the object files (from assembler), combines them together, and gives a single executable file as output.
- In addition to combining modules, a linker also replaces symbolic addresses with real addresses. Therefore, you may need to link a program even if it contains only one module.
- Links libraries with object files.

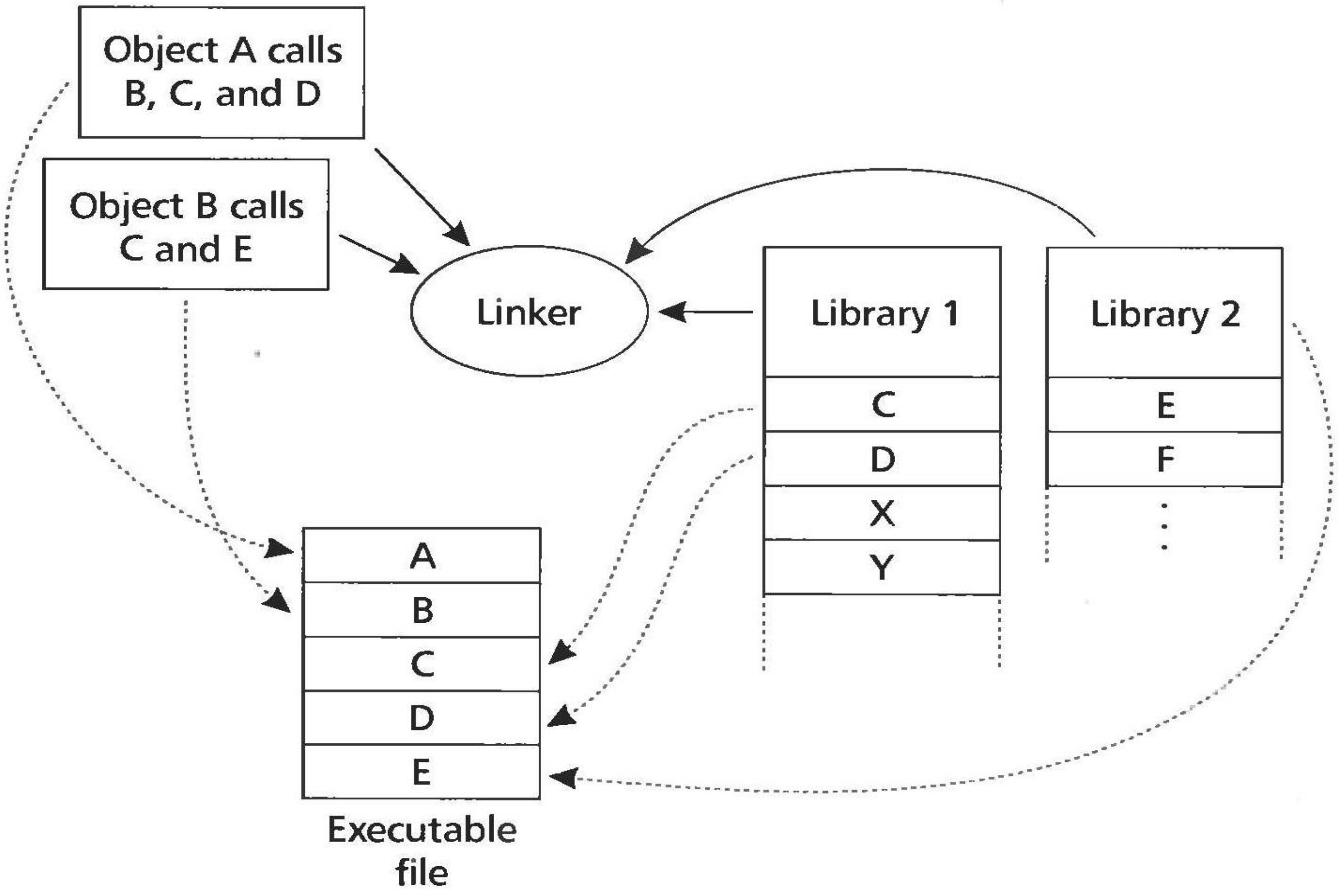
Linker undergoes three task :

1. Searches the program to find library routines used by program, e.g. printf(),sqrt(),strcat() etc.
2. Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references. **Relocation**, which modifies the object program so that it can be loaded at an address different from the location originally specified.
3. It combines two or more separate object programs and supplies the information needed to allow references between them

Linkers

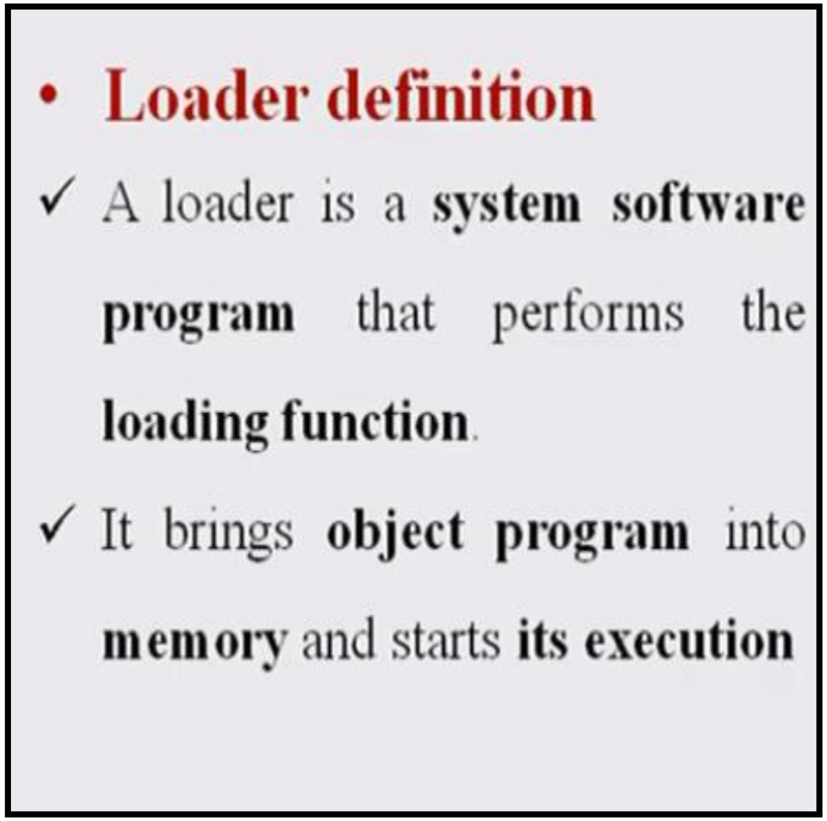

Object file
Instruction
main:
jmp xxxx
-
-
-
jmp yyyy
Relocation records
call funct
call printf
Object file
funct:
-
-
-
C library
printf:
-
-
-
Linker
main:
jmp printf
-
-
jmp funct
-
-
printf:
-
-
funct:
-
-
Executable file

Linkers



Object A calls B, C, and D
Object B calls C and E
Linker
Library 1
C
D
X
Y
Library 2
E
F
A
B
C
D
E
Executable file

Loaders



- **Loader definition**

✓ A loader is a **system software program** that performs the **loading function**.

✓ It brings **object program** into **memory** and starts **its execution**

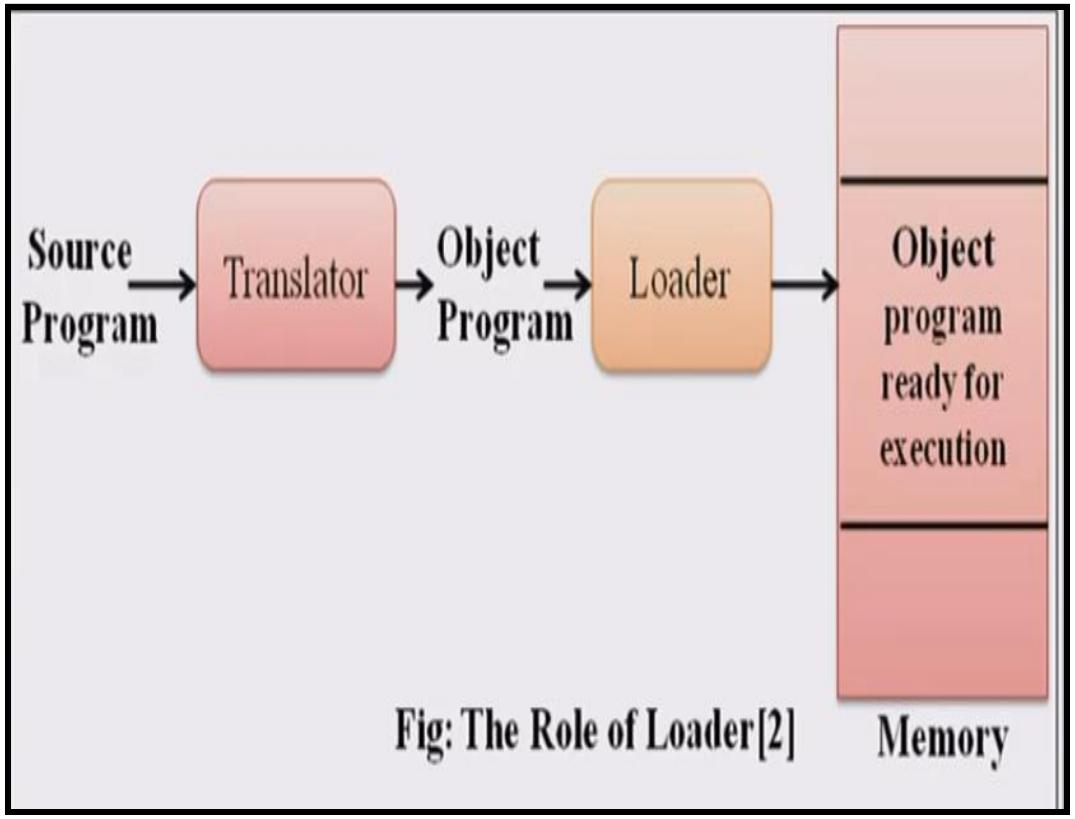


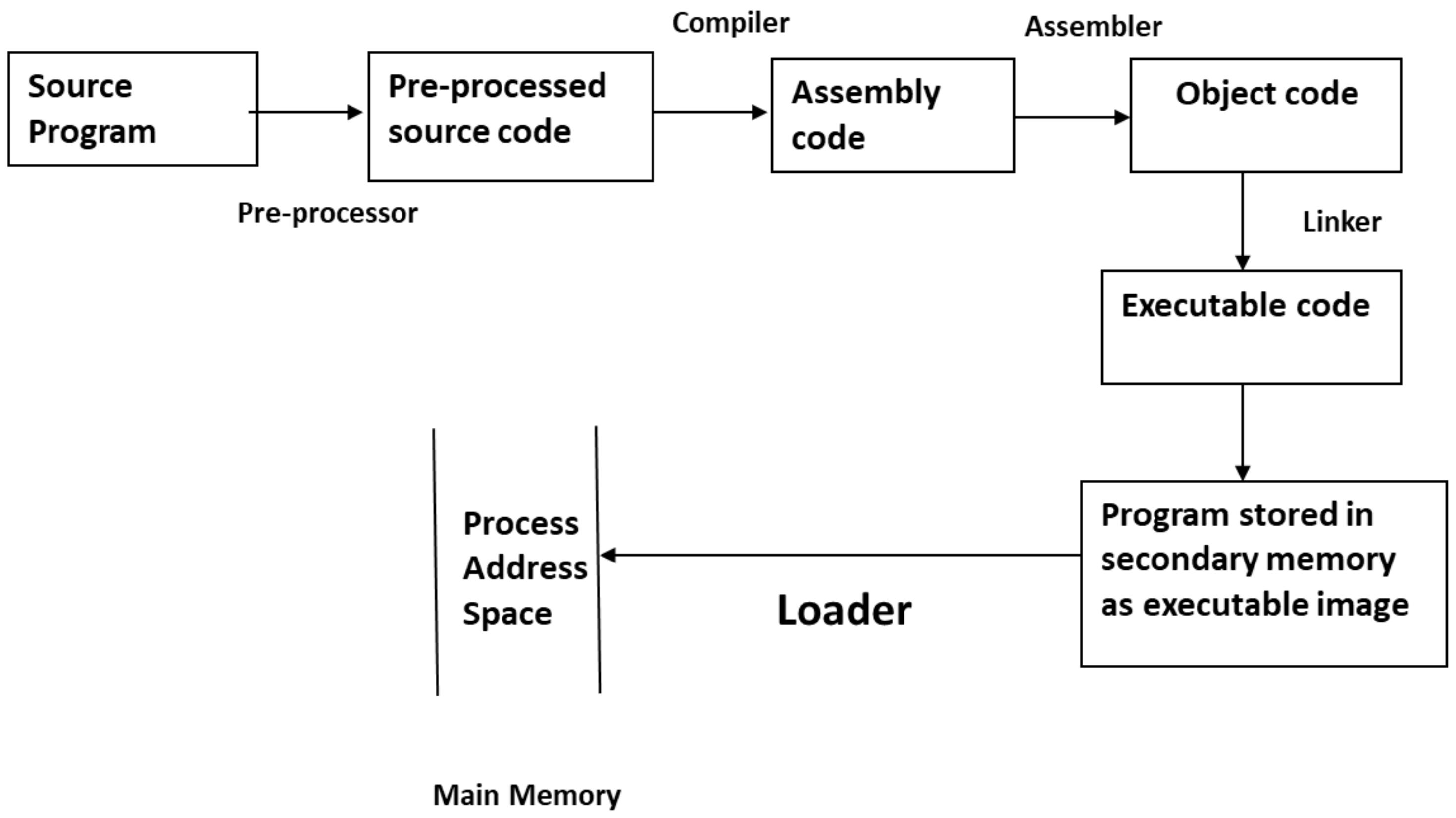
Fig: The Role of Loader[2]

Loaders

- Loader is a part of operating system and is responsible for loading executable files into memory and execute them.
- Takes input object program decks and prepares them for execution.
- It initializes various registers to initiate execution.

Loaders

C Program Building Process



Fundamental process (Function) of Loaders

- **Allocation -** The space for program is allocated in the main memory, by calculating the size of the program.
- **Linking**, which combines two or more separate object programs and supplies the necessary information.
- **Relocation –** modifies the object program so that it can be loaded at an address different from the location originally specified.
- **Loading –** brings the object program into memory for execution.

Loaders Procedure

Step 1 : Reads the header of the executable file to find out the size of the text and data

Step 2 : Creates an address space which is large enough

Step 3 : Copies the instructions and data into primary memory

Step 4 : Copies parameters to the main program onto the stack

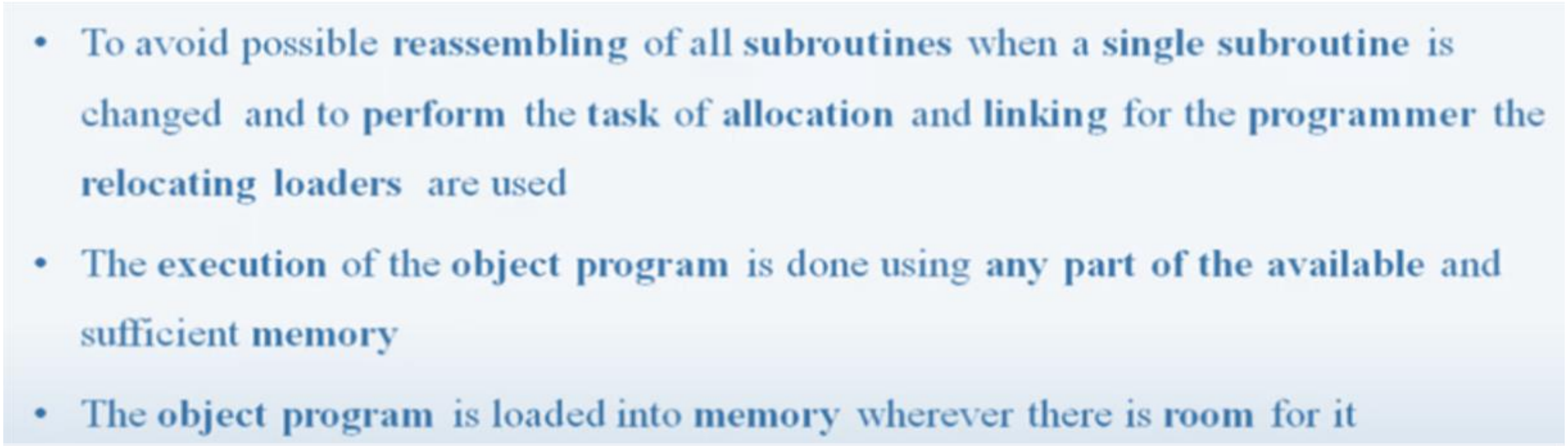
Step 5 : Initialize the machine registers and sets the stack pointer.

Step 6 : Jumps to a start-up procedure that copies the parameters into the argument registers and calls the main procedure

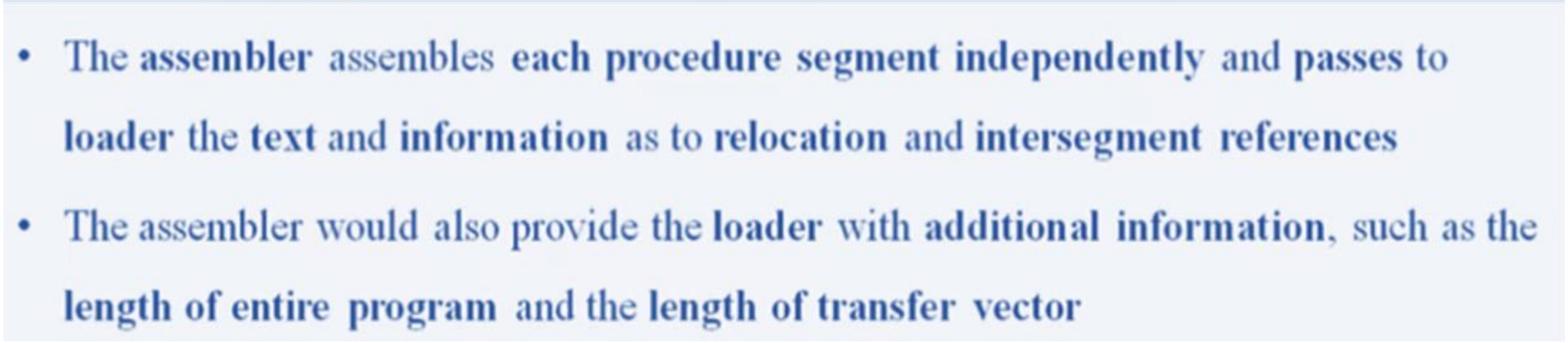
Types of Loader

- Compile and Go Loader
- General Loader Scheme
- Absolute Loader
- Relocating Loaders
- Direct linking Loaders
- Dynamic linking and loading.

Loader-Relocating Loader



- To avoid possible **reassembling** of all **subroutines** when a **single subroutine** is changed and to **perform** the **task** of **allocation** and **linking** for the **programmer** the **relocating loaders** are used
- The **execution** of the **object program** is done using **any part of the available** and sufficient **memory**
- The **object program** is loaded into **memory** wherever there is **room** for it



- The **assembler** assembles **each procedure segment independently** and **passes** to **loader** the **text** and **information** as to **relocation** and **intersegment references**
- The assembler would also provide the **loader** with **additional information**, such as the **length of entire program** and the **length of transfer vector**

Loader-Relocating Loader

- Introduced to avoid reassembling of all subroutines when a single subroutine is changed.
- Performs tasks of allocation and linking for programmer.
- **Each procedure is assembled independently and the text and information for relocation and intersegment references is passed on to the loader**.

Loader-Relocating Loader

- Example – **Binary Symbolic Subroutine (BSS)** used in IBM 7094, IBM 1130, GE 635 and UNIVAC 1108.
- Output of relocating loader using BSS scheme is object program and information about all other programs it references.
- Additionally, Information regarding change of location in the program is present if it is to be loaded arbitrarily in core i.e the location which are independent on core allocation

Loader-Relocating Loader

- Example – **Binary Symbolic Subroutine (BSS)** used in IBM 7094, IBM 1130, GE 635 and UNIVAC 1108.
- This loader allows only **many procedure or code segment** but only **one data segment.**
  - Text , subroutines, variable → code segment
  - Object files created along with its location → data segment
- Assembler assembles each procedure segment independently and passes on to the loader the text and the information and intersegment references. So output of the assembler is object program and info about all other programs it referenced.
- BSS loaders scheme is used on computers with fixed length 'Direct address instruction'.

Loader-Relocating Loader

- For each source program, the assembler gives output which contains text, prefixed by a **transfer vector.**
- Transfer vector consists of addresses containing names of subroutines referenced by source program.
- Assembler also provides loader with length of the program and length of the transfer vector position.

Loader-Relocating Loader

Source Program

Program length = 48 bytes
Transfer vector = 8 bytes

```
MAIN      START
          EXTRN     SQRT
          EXTRN     ERR
          ST    14,SAVE     save return
          address
          L     1,=F'9'     load test value
          BAL   14,SQRT     call SQRT
          C     1,=F'3'     compare ans
          BNE   ERR   transfer to ERR
          L     14,SAVE     get return
          address
          BR    14    return to caller

SAVE DS   F     temp loc
          END
```

Rel. addr	**Relocation**	**Object code**
0	00	'SQRT'
4	00	'ERR'
8	01	ST 14,36
12	01	L 1,40
16	01	BAL 14,0
20	01	C 1,44
24	01	BC 7,4
28	01	L 14,36
32	0	BCR 15,14
34	0	(skipped for alignment)
36	00	(Temp location)
40	00	9
44	00	3

Loader-Relocating Loader

Relocation bits	Meaning
01	Half word is relocated relative to the procedure segment
10	Half word is relocated relative to the address of the single common data segment
00 or 11	Half word is not relocated

Loader-Relocating Loader

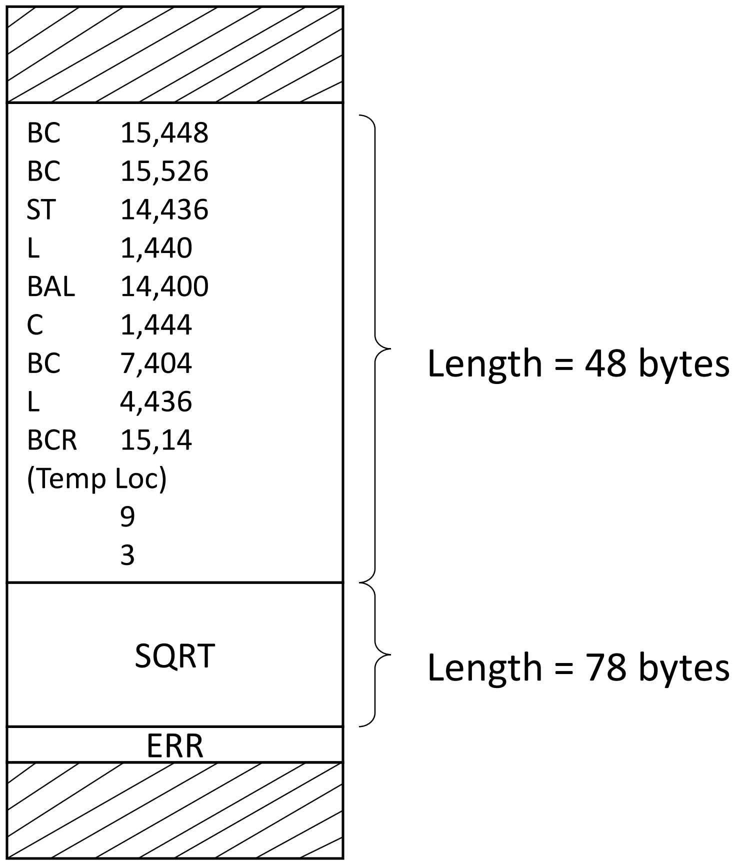
Note:

- For each external symbol the assembler generates a four byte full word at the beginning of the program, Containing EBCDIC characters for the symbol
- But, for simplicity we assume symbols are not more than 4 character long. These extra words are called transfer vector

"Every reference to an external symbol is assigned address of the corresponding transfer vector word"

Loader-Relocating Loader

Absolute Address	Relative Address		
.	.		
.	.		
.	.		
400	0	BC	15,448
404	4	BC	15,526
408	8	ST	14,436
412	12	L	1,440
416	16	BAL	14,400
420	20	C	1,444
424	24	BC	7,404
428	28	L	4,436
432	32	BCR	15,14
436	36	(Temp Loc)	
440	40		9
444	44		3
.			
.			
.			
.			
526		ERR	
.			
.			



Loader-Relocating Loader

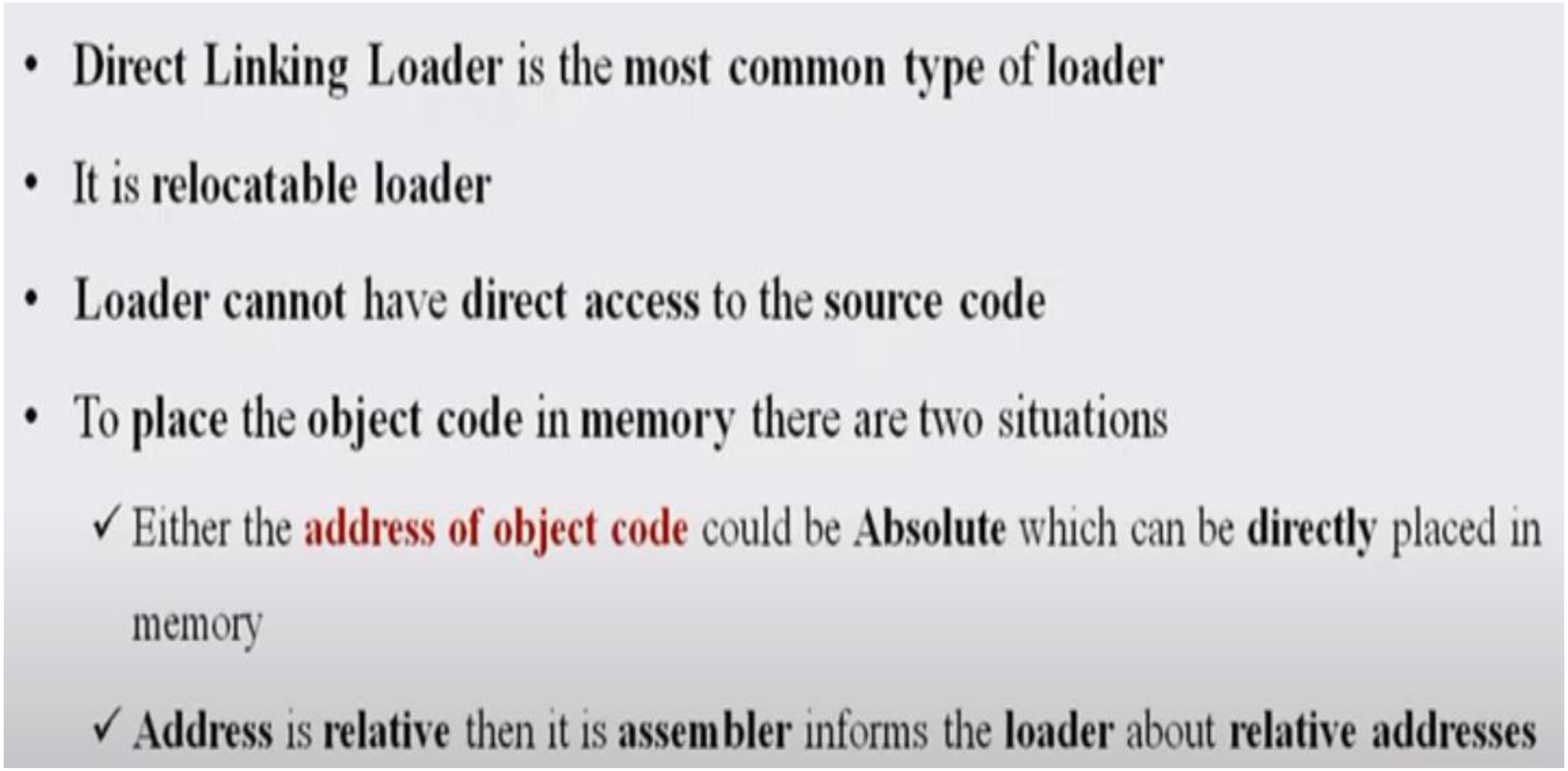
Pros:

Relocation bits are used to solve the problem of relocation , transfer vector is used to solve the problem of linking & program length info to solve allocation.

Cons:

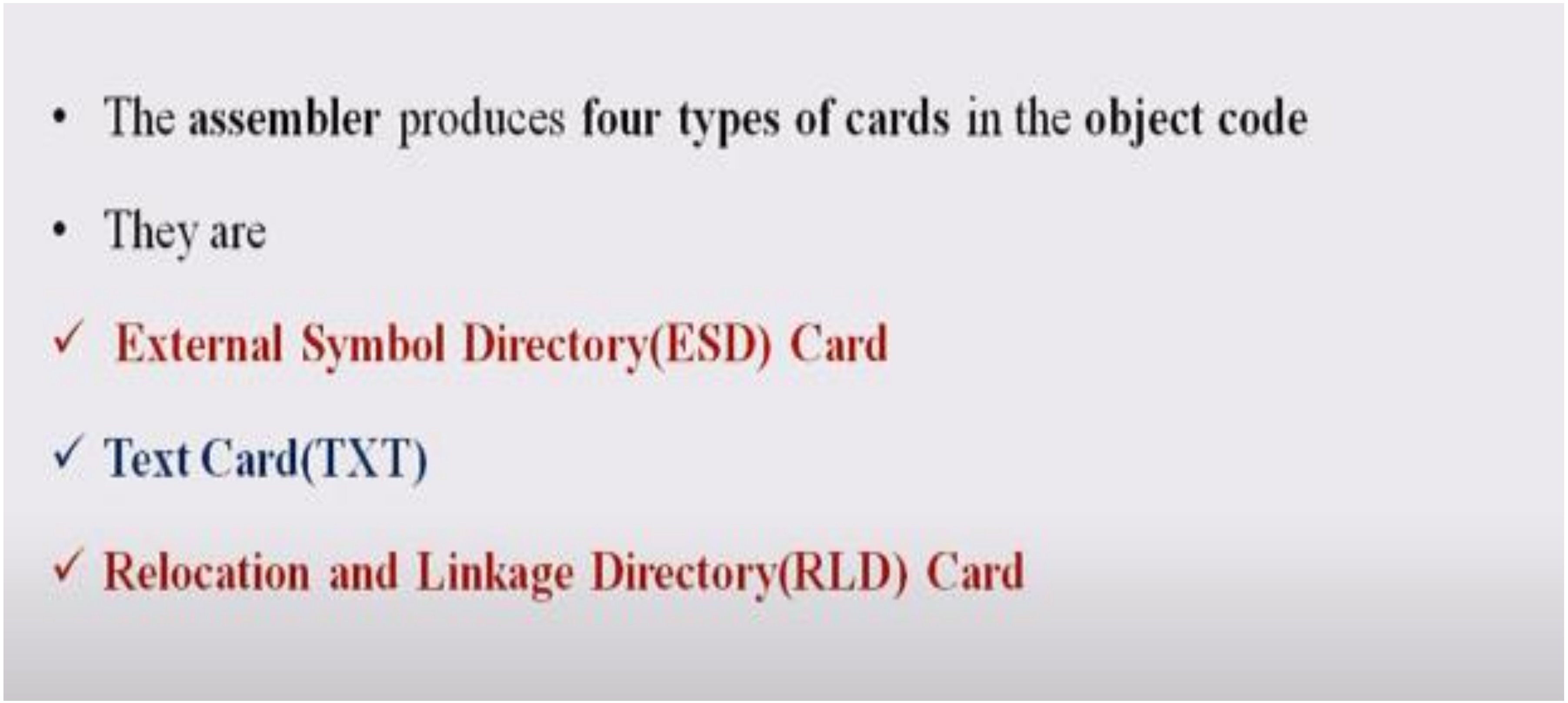
- ❑Not suited for loading external data.
- ❑Transfer vector increases the size
- ❑Does not facilitate access to data segments that can be shared

Loader-Direct Linking Loader



- Direct Linking Loader is the most common type of loader
- It is relocatable loader
- Loader cannot have direct access to the source code
- To place the object code in memory there are two situations
  - ✓ Either the address of object code could be **Absolute** which can be **directly** placed in memory
  - ✓ **Address** is **relative** then it is **assembler** informs the **loader** about **relative addresses**

Loader-Direct Linking Loader



- The assembler produces four types of cards in the object code
- They are

✓ External Symbol Directory(ESD) Card

✓ Text Card(TXT)

✓ Relocation and Linkage Directory(RLD) Card

Loader-Direct Linking Loader

- **External Symbol Directory(ESD)** cards contain information about all symbols that are defined in the present program, but they may be referenced somewhere.
- It contains
- Reference No.
- Symbol Name
- TYPE
- Relative Location
- Length

Loader-Direct Linking Loader

- **External Symbol Directory(ESD)** cards contain information about all symbols that are defined in the present program, but they may be referenced somewhere.
- It contains
- Reference No.
- Symbol Name
- TYPE
- Relative Location
- Length

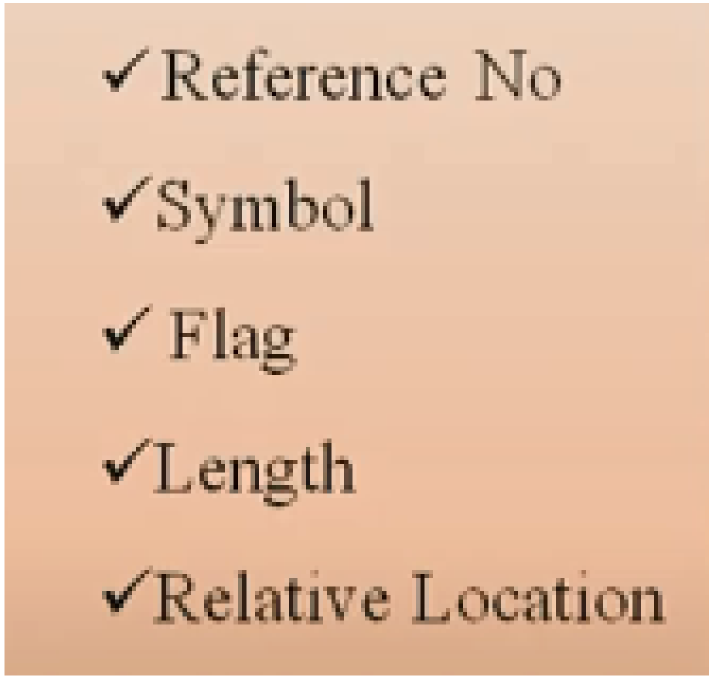
TYPE
SD- Segment Definition
LD- Local Definition
ER-External Reference

Loader-Direct Linking Loader

- **The Text Cars(TXT)** cards contains the actual object code translated version of source program.
- **The Relocation and linkage Directory (RLD)** card contains information about those locations in the program whose contents depends on the address at which program is placed.
  - The RLD Cards contains the following information:-
  - The location of each constants that needs to be changed due to relocation
  - By What it has to be changed
  - The operation to be performed

Loader-Direct Linking Loader

- **Format of RLD**



✓Reference No

✓Symbol

✓ Flag

✓Length

✓Relative Location

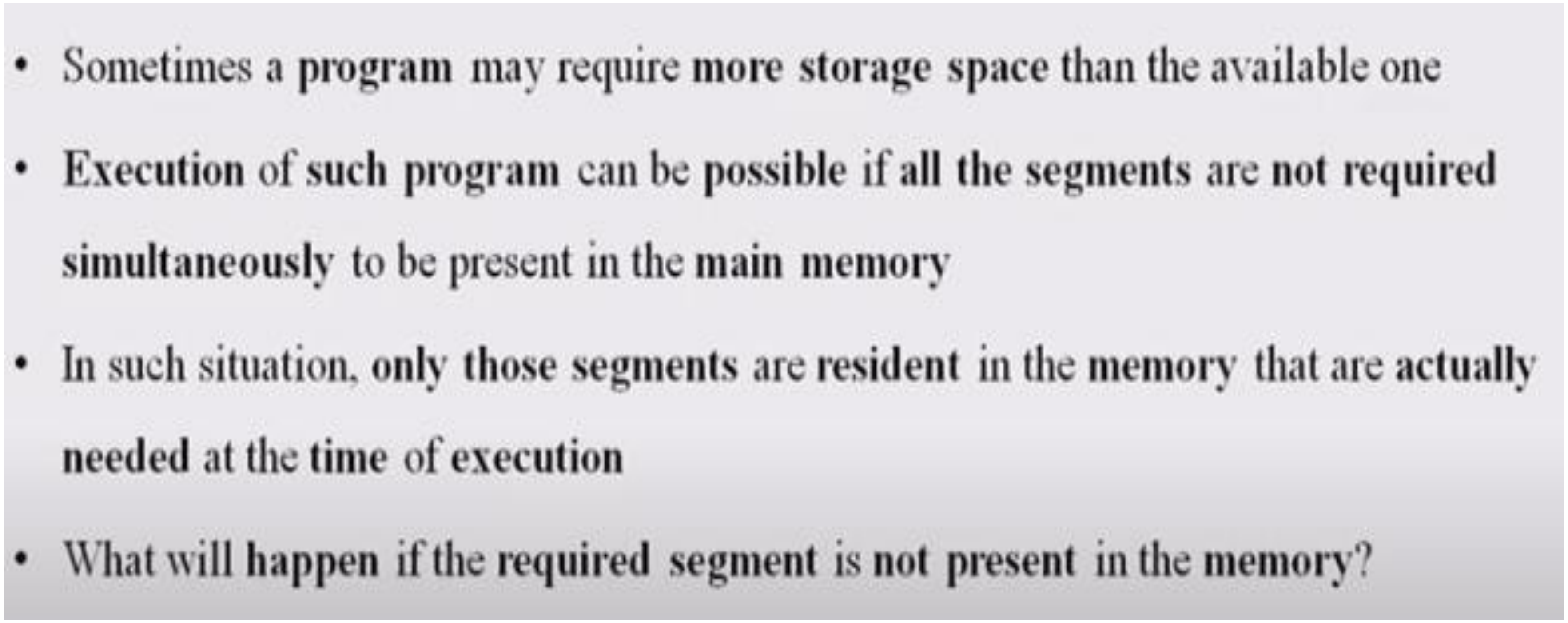
Loader-Direct Linking Loader

- It is a general relocating loader and most popular loading scheme.
- Allows programmer multiple procedure segments and multiple data segments thus giving complete freedom in referencing data or instructions in other segments.
- This provides flexible intersegment referencing and accessing ability while allowing independent translations of programs.

Loader-Direct Linking Loader

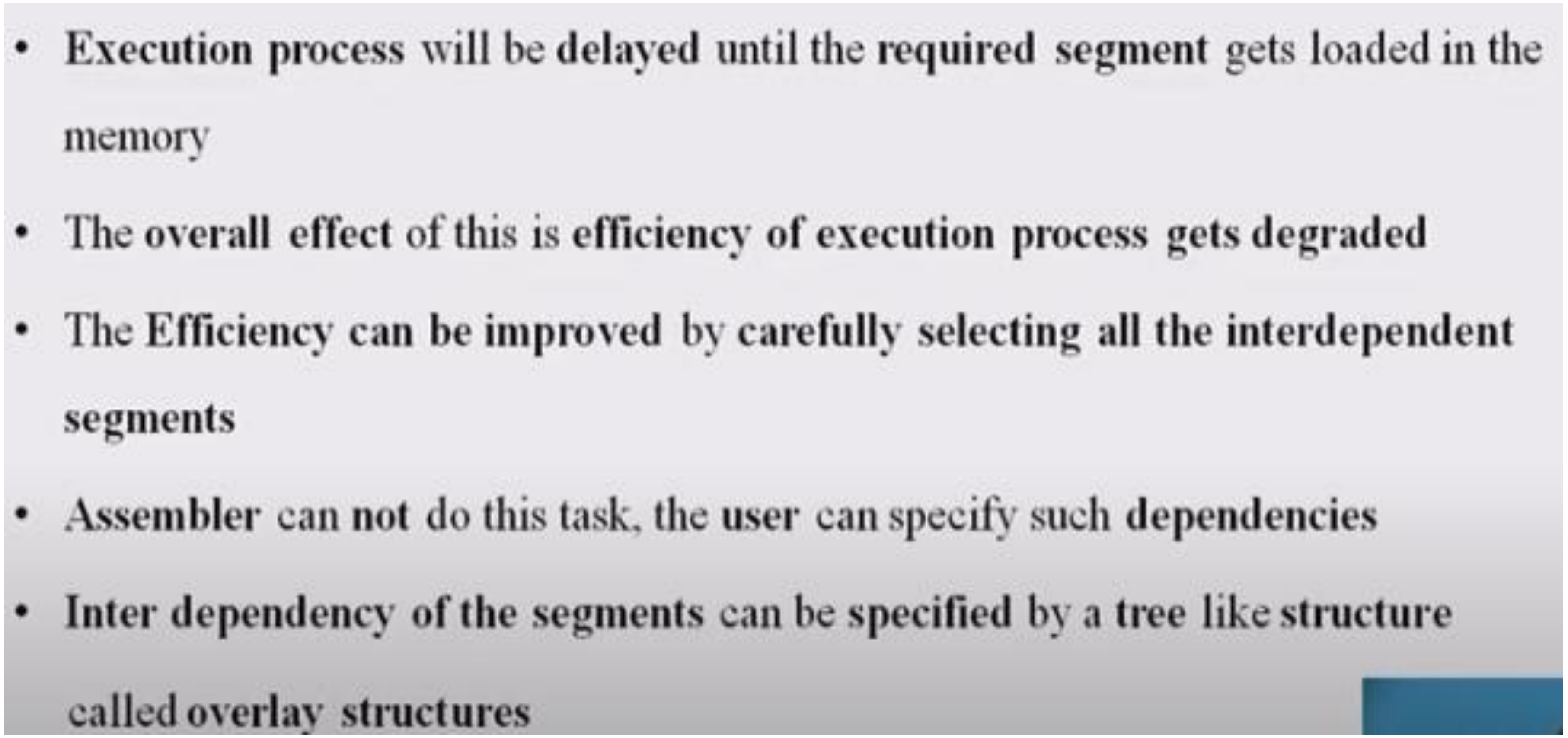
- With each procedure or data segment, the assembler must give the loader the following info –
  - Length of the segment
  - List of all the symbols and their relative location
  - List of all the symbols not defined in the segment but referenced in the segment
  - Info as to where address constants are located in the segment and description of how to revise their values
  - Machine code translation of the source program and the relative addresses assigned

Loader- Dynamic Loading



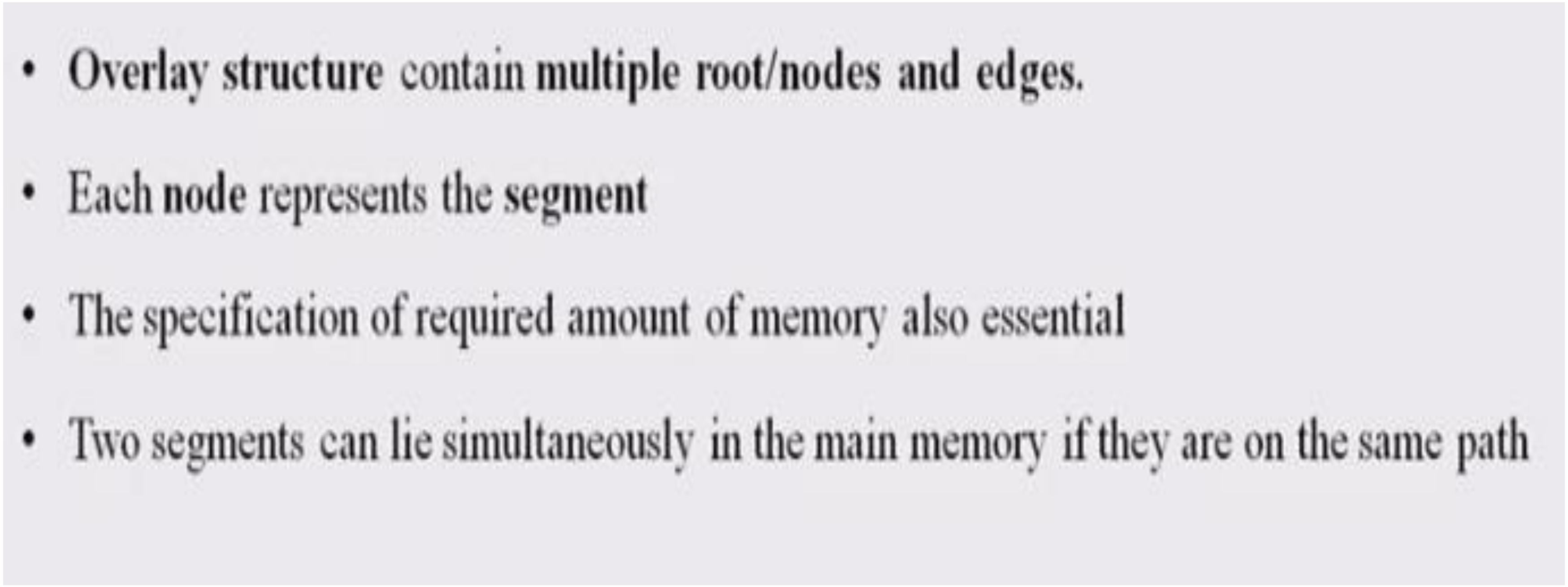
- Sometimes a **program** may require **more storage space** than the available one
- **Execution** of **such program** can be **possible** if **all the segments** are **not required simultaneously** to be present in the **main memory**
- In such situation, **only those segments** are **resident** in the **memory** that are **actually needed** at the **time** of **execution**
- What will **happen** if the **required segment** is **not present** in the **memory**?

Loader- Dynamic Loading



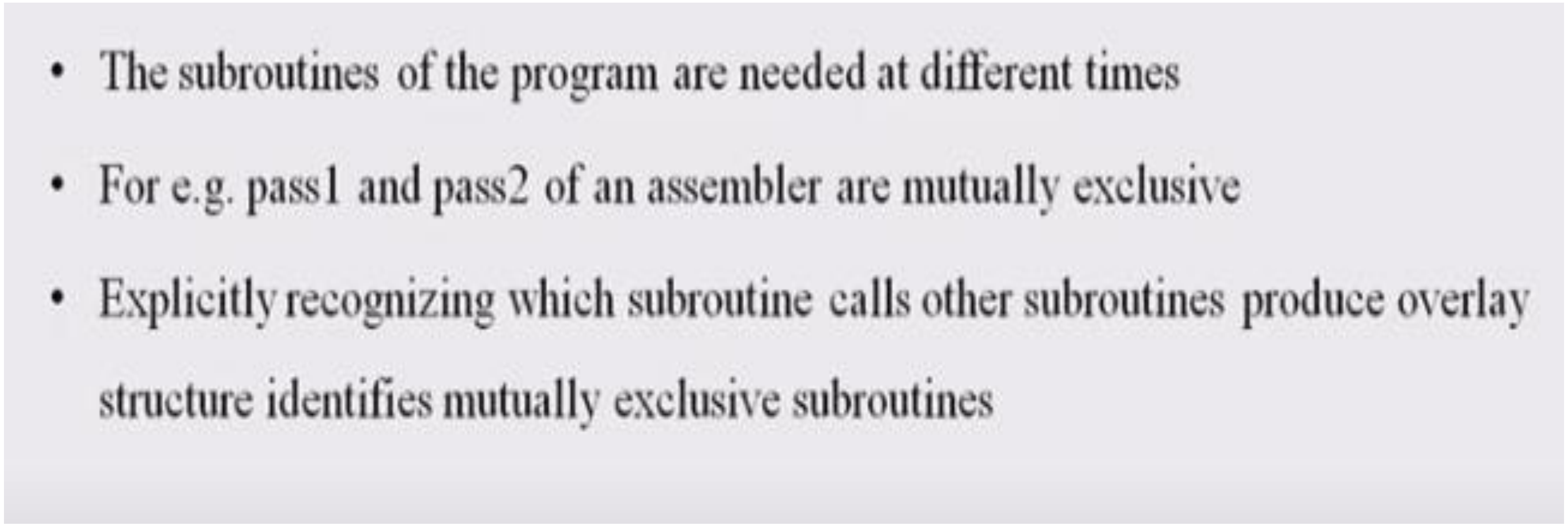
- Execution process will be delayed until the required segment gets loaded in the memory
- The overall effect of this is efficiency of execution process gets degraded
- The Efficiency can be improved by carefully selecting all the interdependent segments
- Assembler can not do this task, the user can specify such dependencies
- Inter dependency of the segments can be specified by a tree like structure called overlay structures

Loader- Dynamic Loading



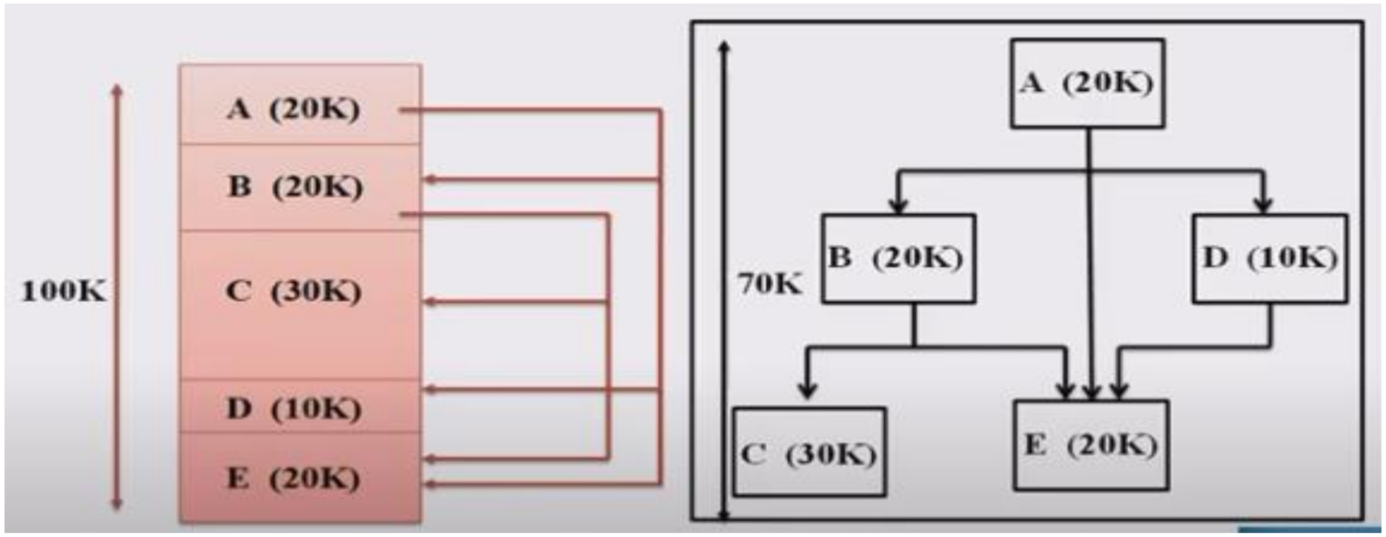
- Overlay structure contain multiple root/nodes and edges.
- Each node represents the segment
- The specification of required amount of memory also essential
- Two segments can lie simultaneously in the main memory if they are on the same path

Loader- Dynamic Loading



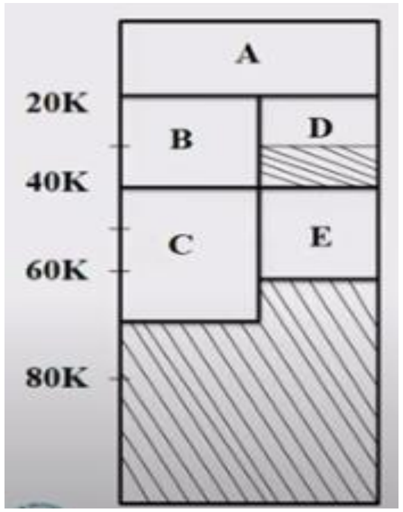
- The subroutines of the program are needed at different times
- For e.g. pass1 and pass2 of an assembler are mutually exclusive
- Explicitly recognizing which subroutine calls other subroutines produce overlay structure identifies mutually exclusive subroutines

Loader- Dynamic Loading

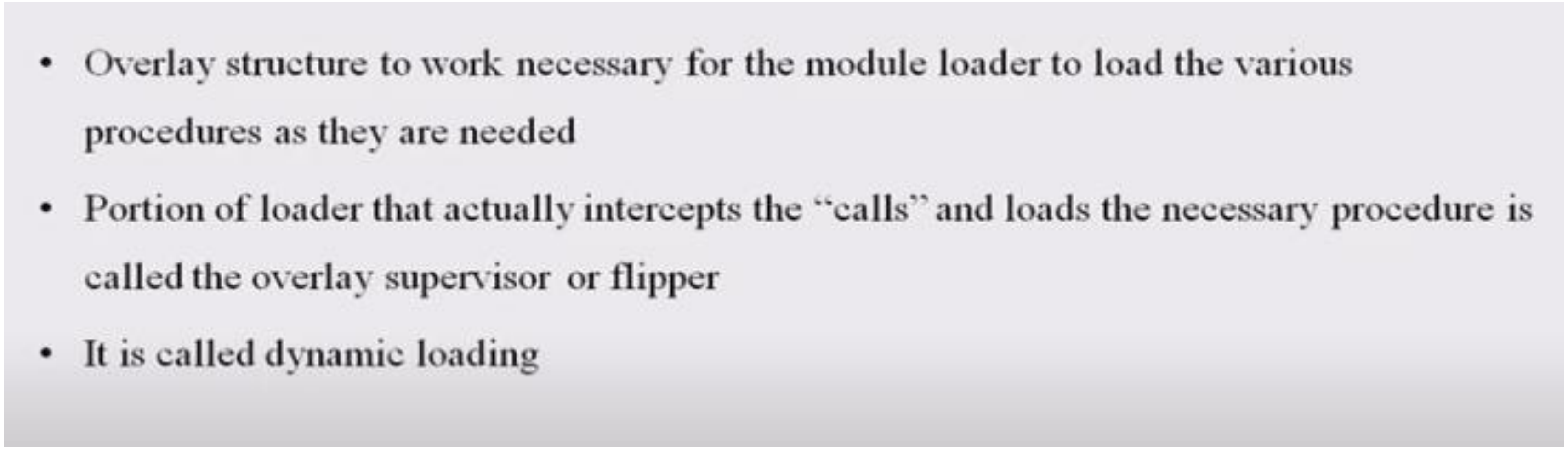


Subroutine calls between the procedure

Loader- Dynamic Loading

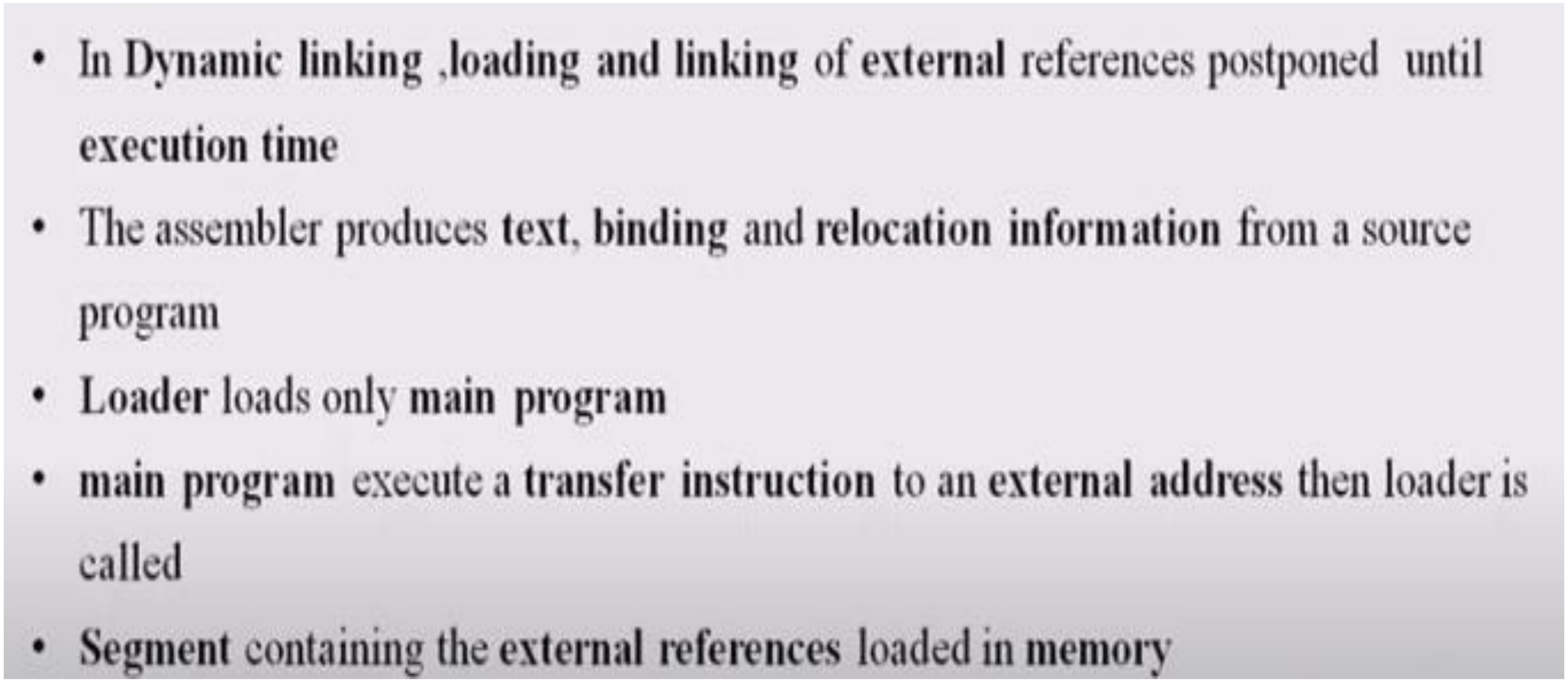


Possible storage assignment of each procedure



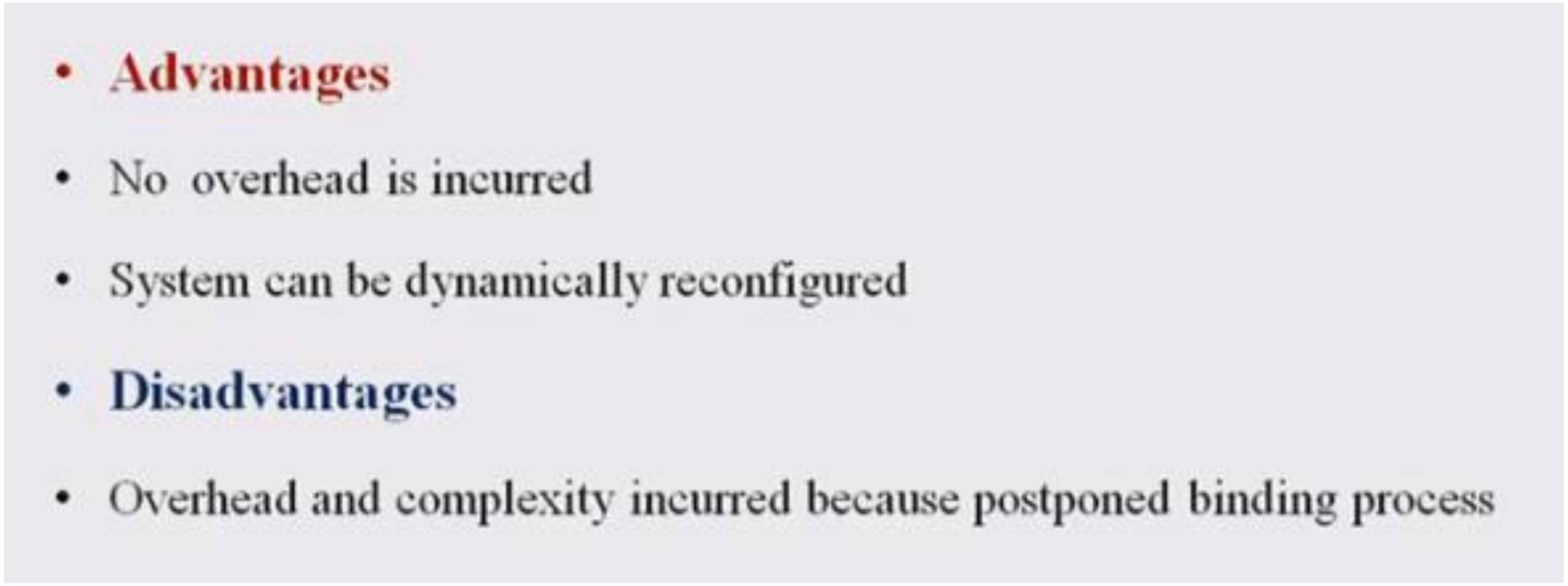
- Overlay structure to work necessary for the module loader to load the various procedures as they are needed
- Portion of loader that actually intercepts the "calls" and loads the necessary procedure is called the overlay supervisor or flipper
- It is called dynamic loading

Loader- Dynamic Linking



- In Dynamic linking ,loading and linking of external references postponed until execution time
- The assembler produces text, binding and relocation information from a source program
- Loader loads only main program
- main program execute a transfer instruction to an external address then loader is called
- Segment containing the external references loaded in memory

Loader- Dynamic Loading & Linking



- **Advantages**
- No overhead is incurred
- System can be dynamically reconfigured
- **Disadvantages**
- Overhead and complexity incurred because postponed binding process

How does loader gets loaded ?

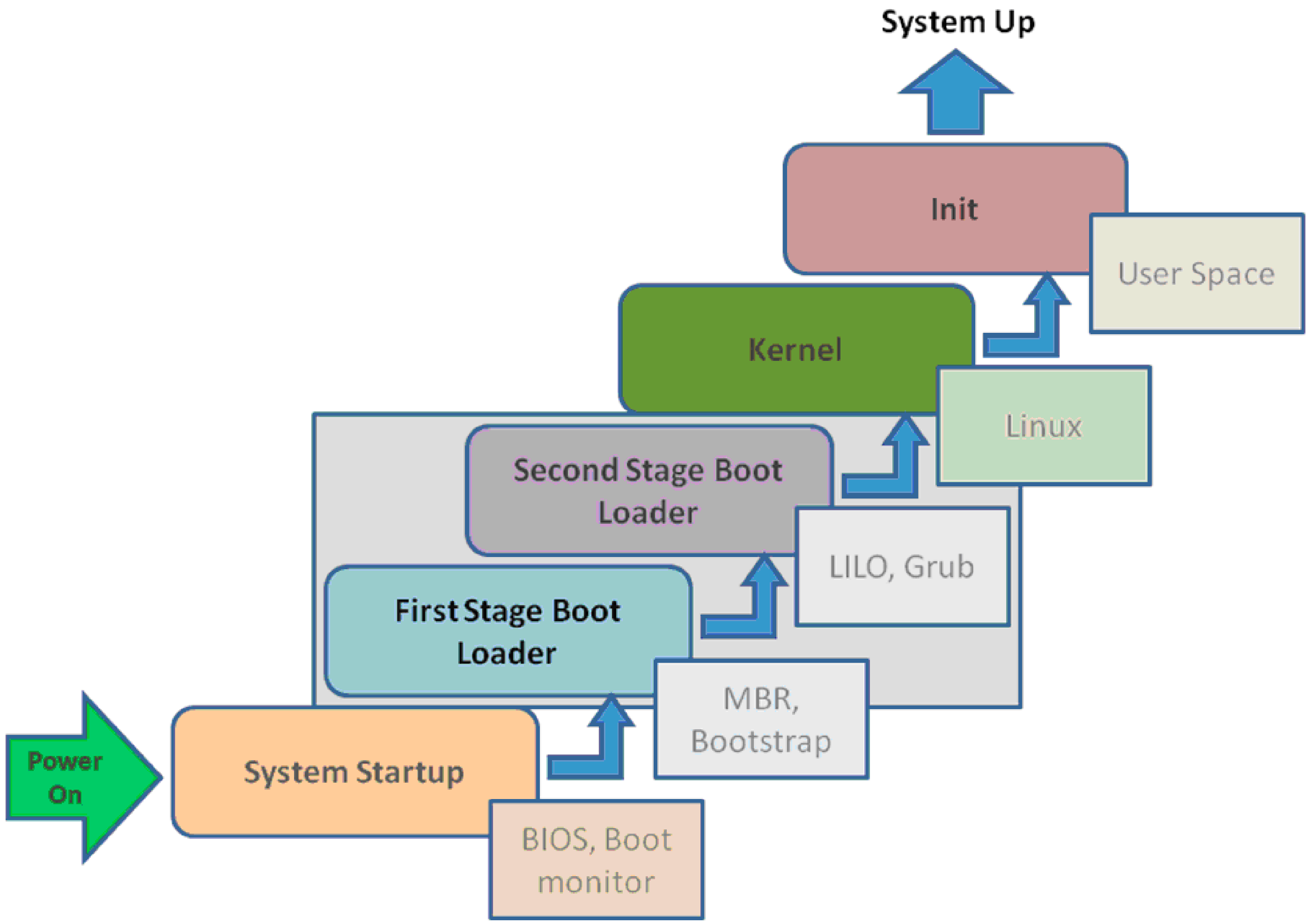
Answer: **Bootstrap Loader**

A **boot**strap loader is a computer program that loads the main operating system or runtime environment for the computer after completion of self-tests.

Bootstrap Loader

- Also known as bootstrapping, boot loader or boot program.
- Program that resides in non-volatile memory.
- Resides in the MBR (master boot record).
- Automatically executed by processor when switching on the system after basic BIOS checks.
- It reads the boot sector to load OS.

Bootstrap Loader



THE END!

Have a nice day!