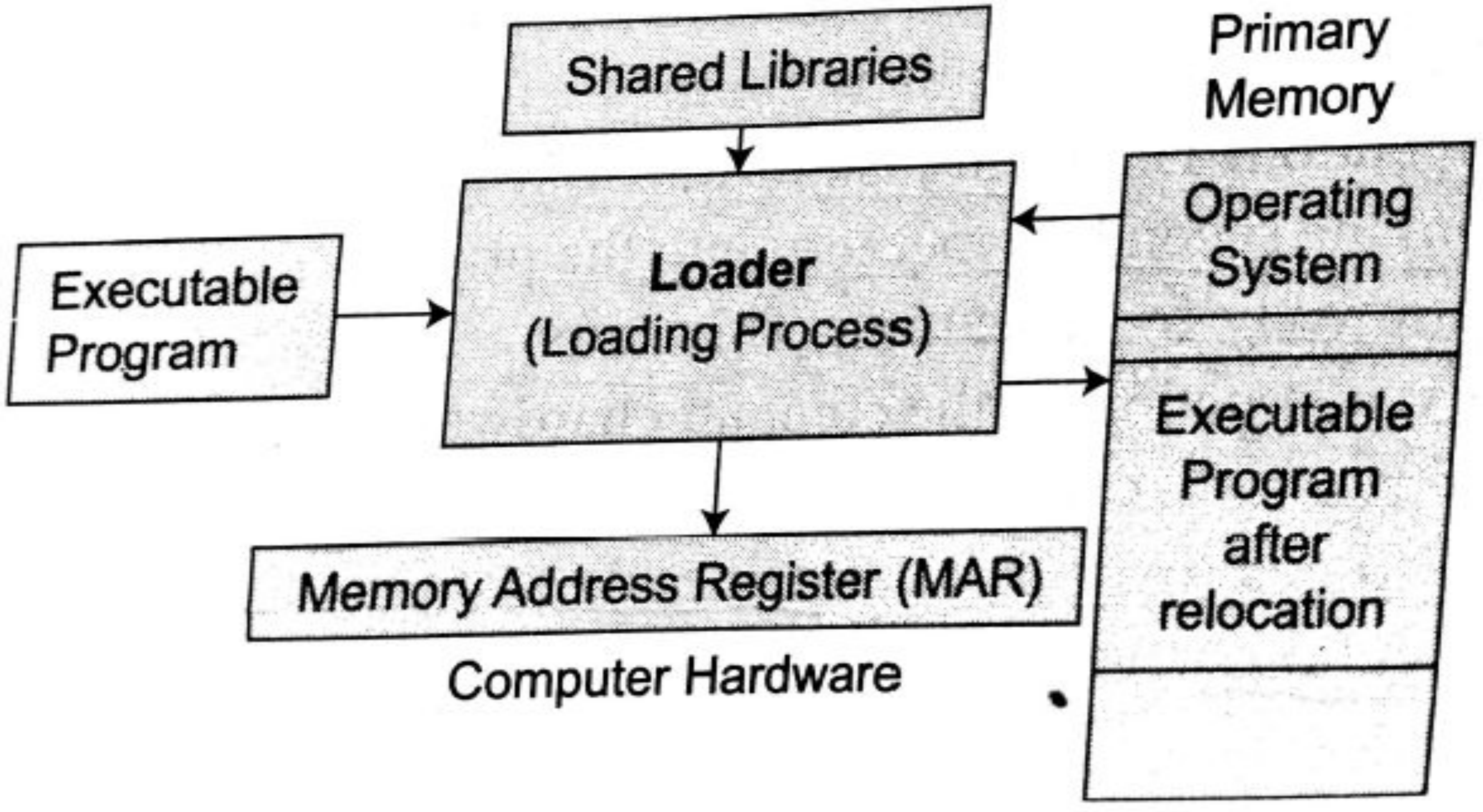
Loader

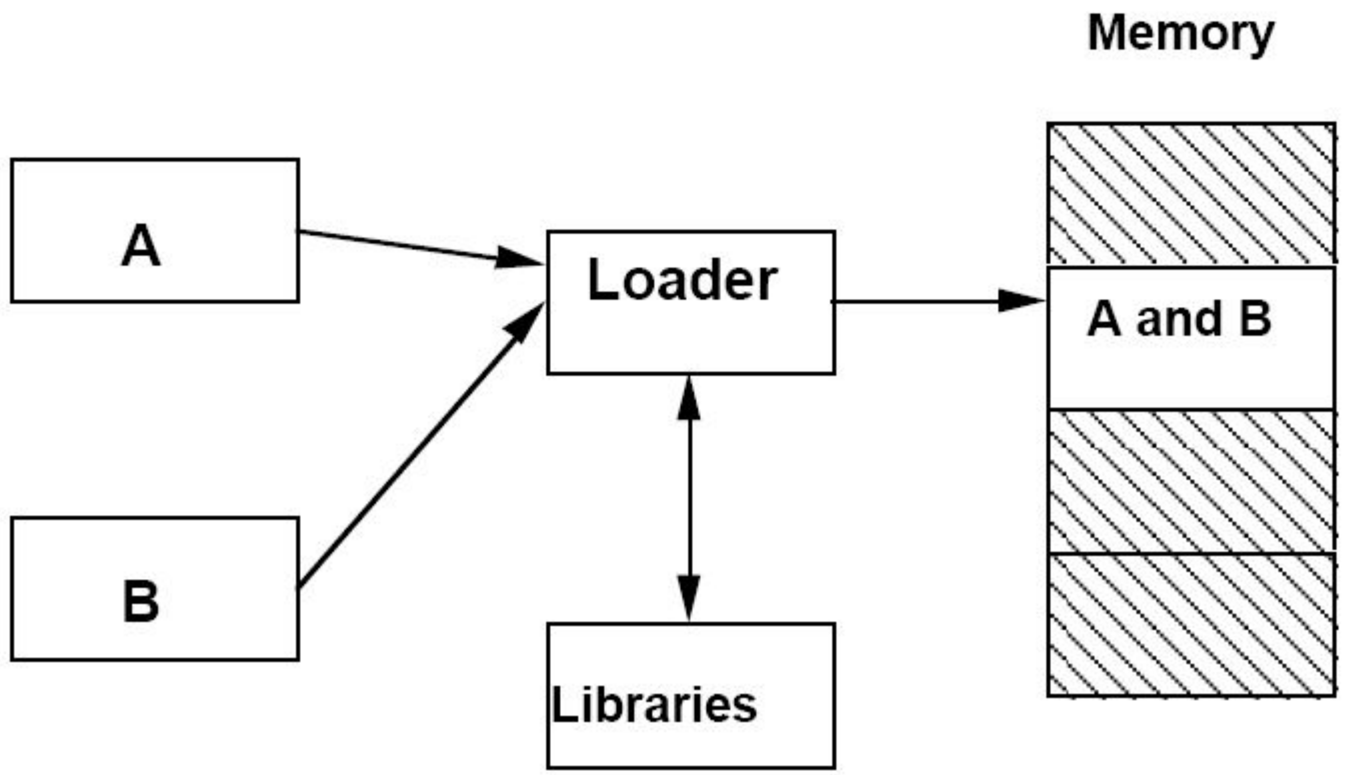
Loader & its Functions

- A loader is a system program, which takes the executable code of a program as input and prepares it for execution.
  - many also support relocation & linking
  - others have a separate linker and loader
- Basic Functions
  - bringing an object program into memory
  - starting its execution

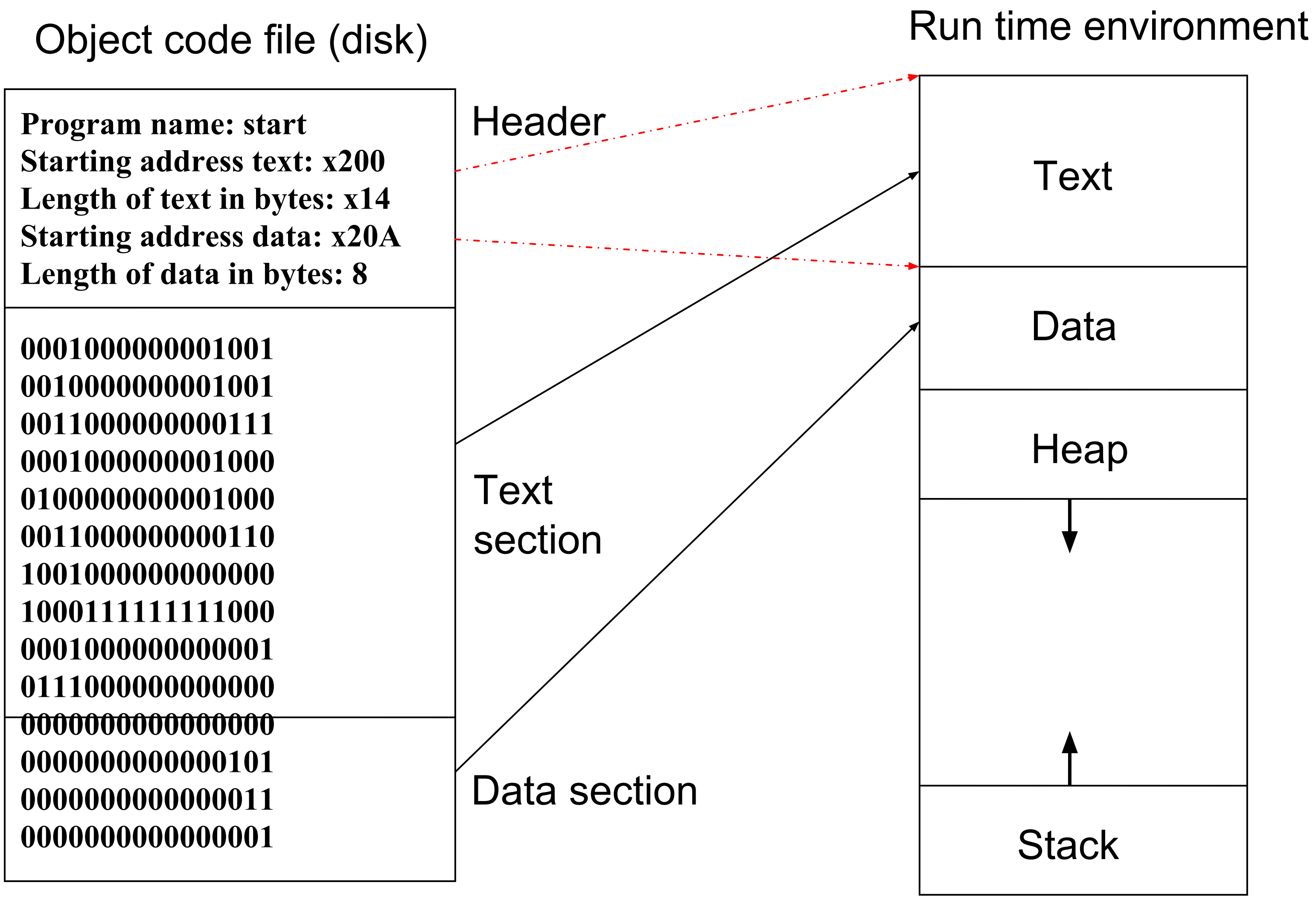
A Simple Loading Process



Loading Process with Linking



Loading object code into memory



Loader & its Functions

- **Loader Function** : The loader performs the following functions :
  - ***Allocation* -** The loader determines and allocates the required memory space for the program to execute properly.
  - ***Linking* --** The loader analyses and resolve the symbolic references made in the object modules.
  - ***Relocation* -** The loader maps and relocates the address references to correspond to the newly allocated memory space during

STEPS INVOLVED IN LOADING

- Executable file’ s header is read to determine the size of text and data segments.
- Instructions and data are copied into address space.
- Arguments passed to the program are copied on the stack
- Machine registers including the stack pointer are initialized.
- The control is transferred to a start-up routine that copies the program’ s arguments from the stack to registers and calls the program’ s main routine.

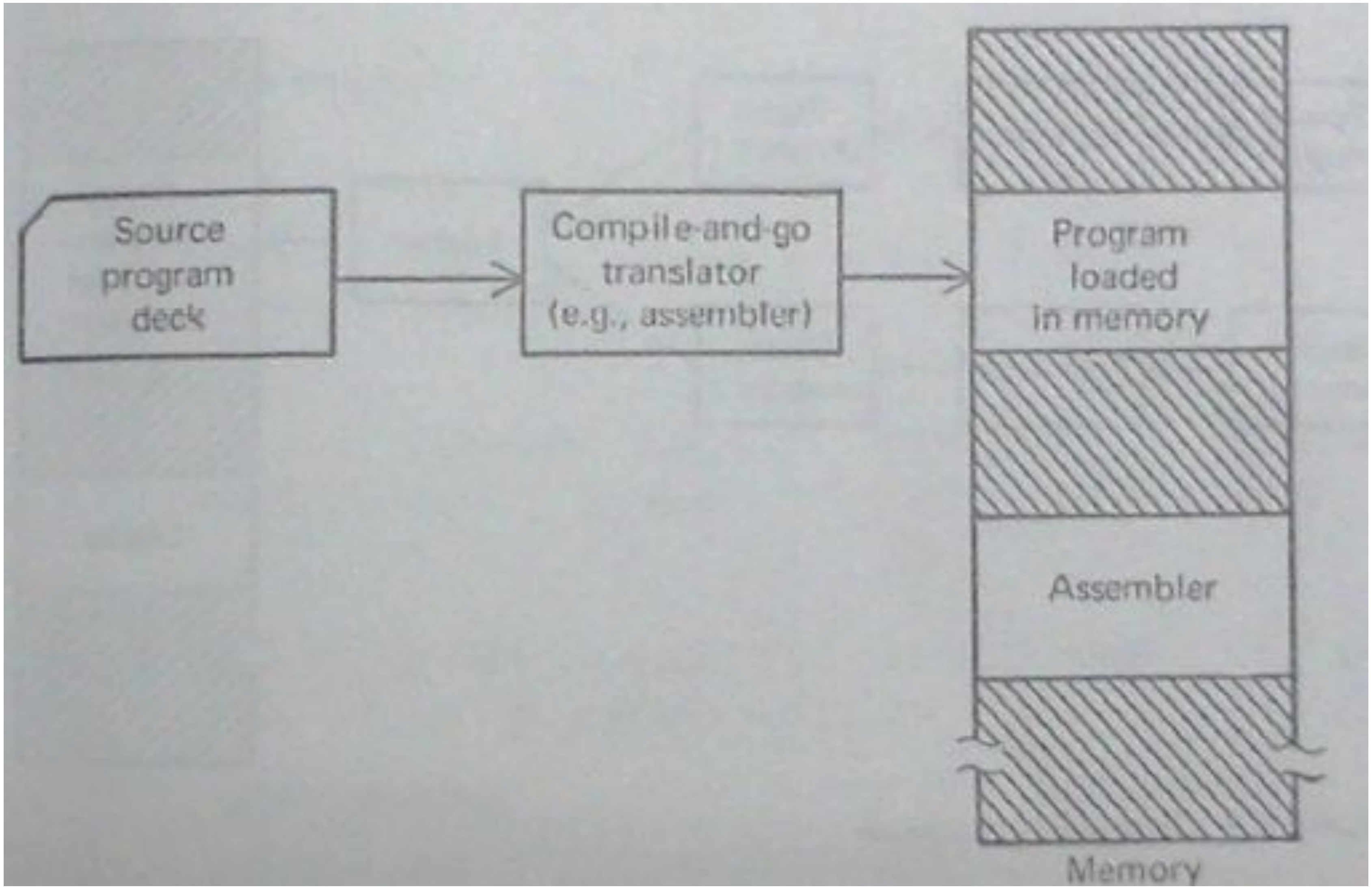
Types of Loaders

- Type of loaders
  - Compile-and-go loader
  - absolute loader (bootstrap loader)
  - relocating loader (relative loader)
  - direct linking loader
  - General Loader Scheme

Compile and Go Loader

- **In compile and go loader is a link editor/program loader in which the assembler itself places the assembled instruction directly into the designated memory locations for execution.**
  - **The assembler run in one part of memory**
  - **Place the assembled machine**

Compile and Go Loader



Advantages and Disadvantages

- **Advantages of Compile-and-go loaders**:

- Simple and easier to implement.
- No additional routines are required to load the compiled code into the memory.

- **Disadvantages of Compile-and-go loaders**:

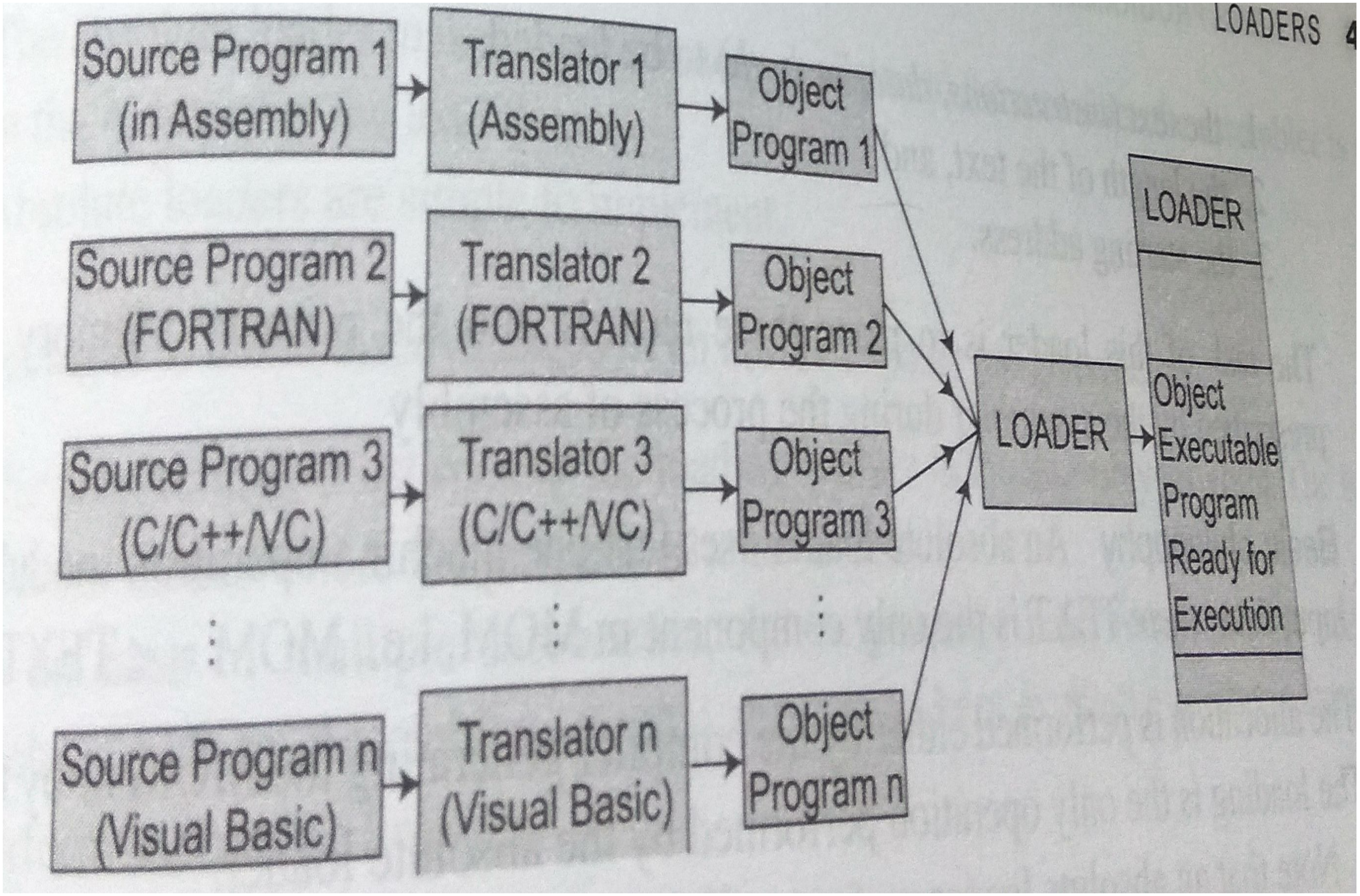
- Wastage of memory space due to the presence of the assembler.
- There is a need to re-assemble the code every time it is to be run.
- It becomes increasingly difficult to handle large number of segments when the input code is written in a variety of HLL say one routine in Pascal and one in FORTRAN and so on.

- Such loader make designing modular programs and systems near impossible.

General Loader Scheme

- In “Compile-and-Go” the outputting instruction and data are assembled. In which assembler is placed in main memory that results in wastage of memory.
- To overcome that we requires the addition of the new program of the system, a loader.
- Generally the size of loader is less than that of assembler.

General Loader Scheme


LOADERS 4
Source Program 1 (in Assembly)
Translator 1 (Assembly)
Object Program 1
Source Program 2 (FORTRAN)
Translator 2 (FORTRAN)
Object Program 2
Source Program 3 (C/C++/VC)
Translator 3 (C/C++/VC)
Object Program 3
Source Program n (Visual Basic)
Translator n (Visual Basic)
Object Program n
LOADER
LOADER
Object Executable Program Ready for Execution

Advantage of General Loader Scheme

- In this scheme the source program translators produce **compatible object program deck formats** and it is possible to write subroutines in several different languages since the object decks to be processed by the loader will all be in the same “language” that is in “machine language”.

Absolute Loader

Design of an Absolute Loader

- Its operation is very simple
  - no linking or relocation
- Single pass operation
  - check **H record to verify that correct program** has been presented for loading
  - read each **T record, and move object code into** the indicated address in memory
  - at **E record, jump to the specified address to** begin execution of the loaded program.

ABSOLUTE LOADER

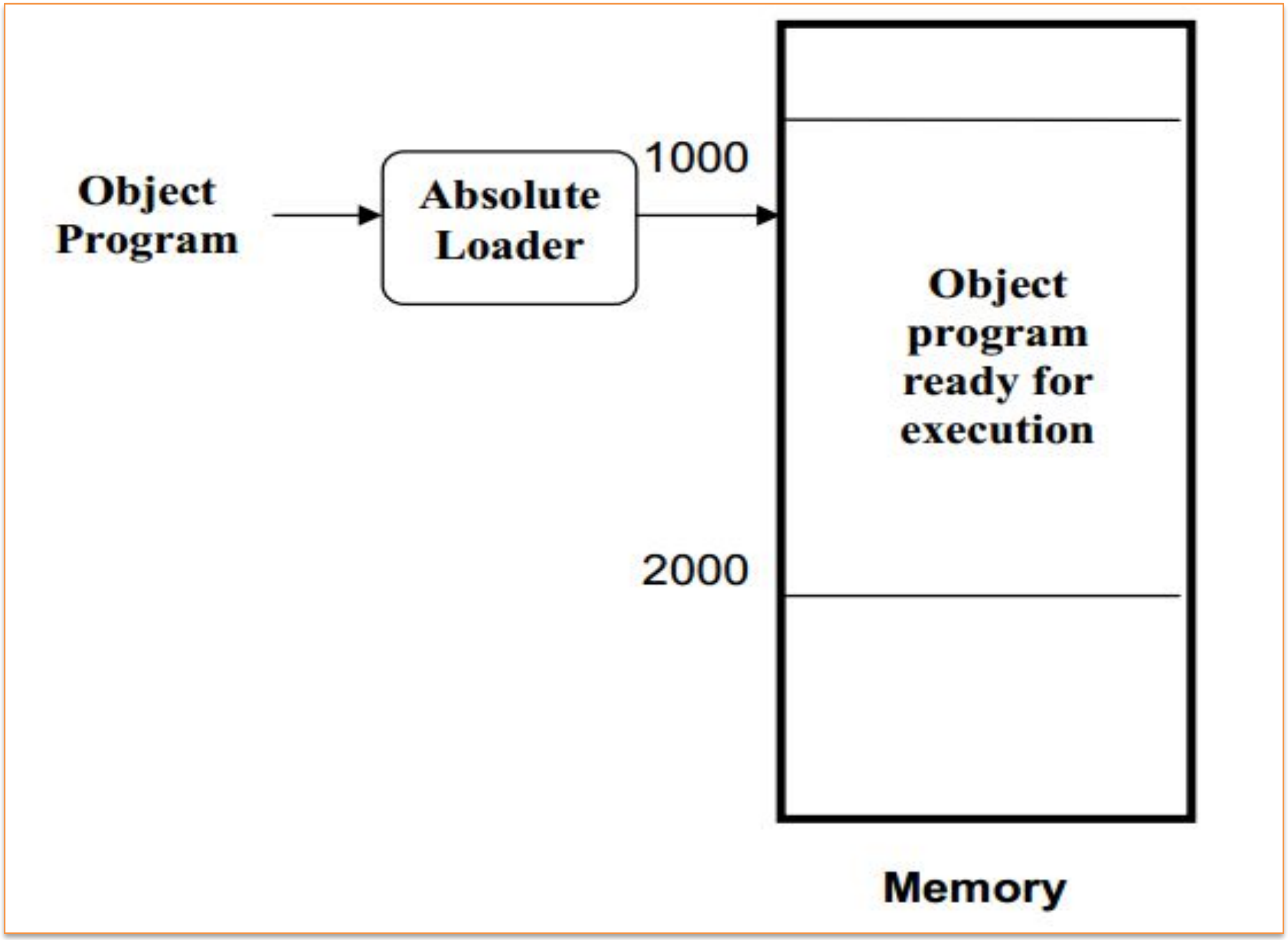
- Absolute loader loads the object code to specified locations in the memory.
- At the end the loader jumps to the specified address to begin execution of the loaded program.

ADVANTAGES : Simple and efficient.

DISADVANTAGES :

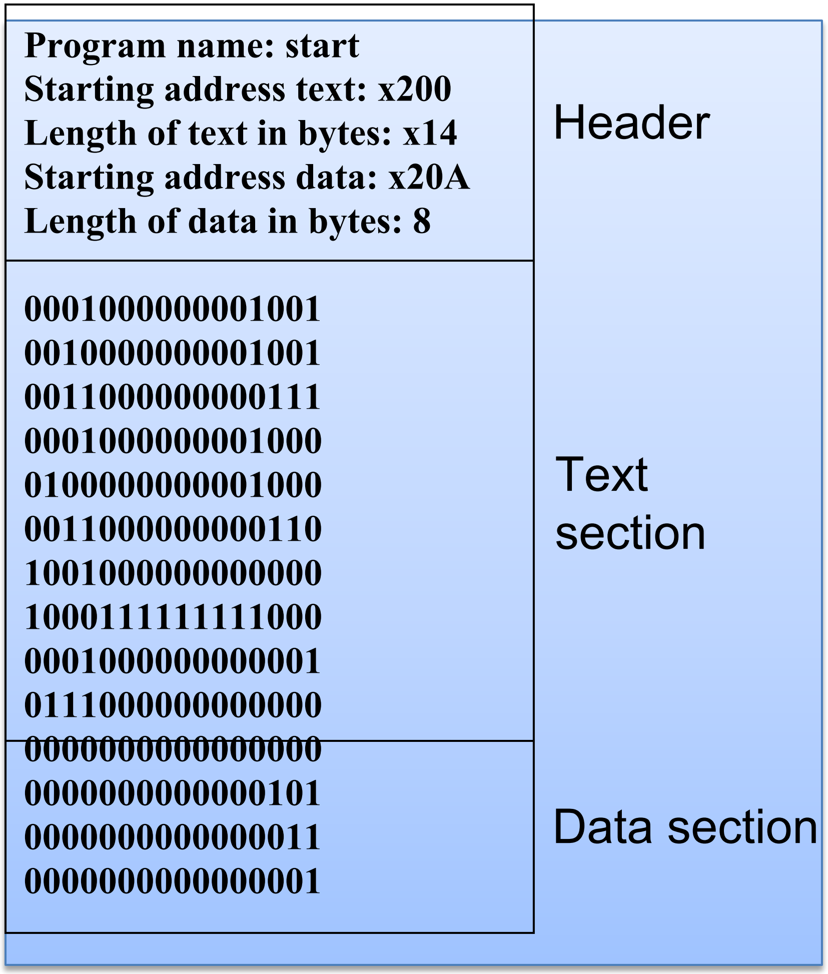
– The need for programmer to specify the actual

ROLE OF ABSOLUTE LOADER



Absolute Loader

Assembler object code file



```
Program name: start
Starting address text: x200
Length of text in bytes: x14
Starting address data: x20A
Length of data in bytes: 8

0001000000001001
0010000000001001
0011000000000111
0001000000001000
0100000000001000
0011000000000110
1001000000000000
1000111111111000
0001000000000001
0111000000000000
0000000000000000
0000000000000101
0000000000000011
0000000000000001
```

Absolute loader:

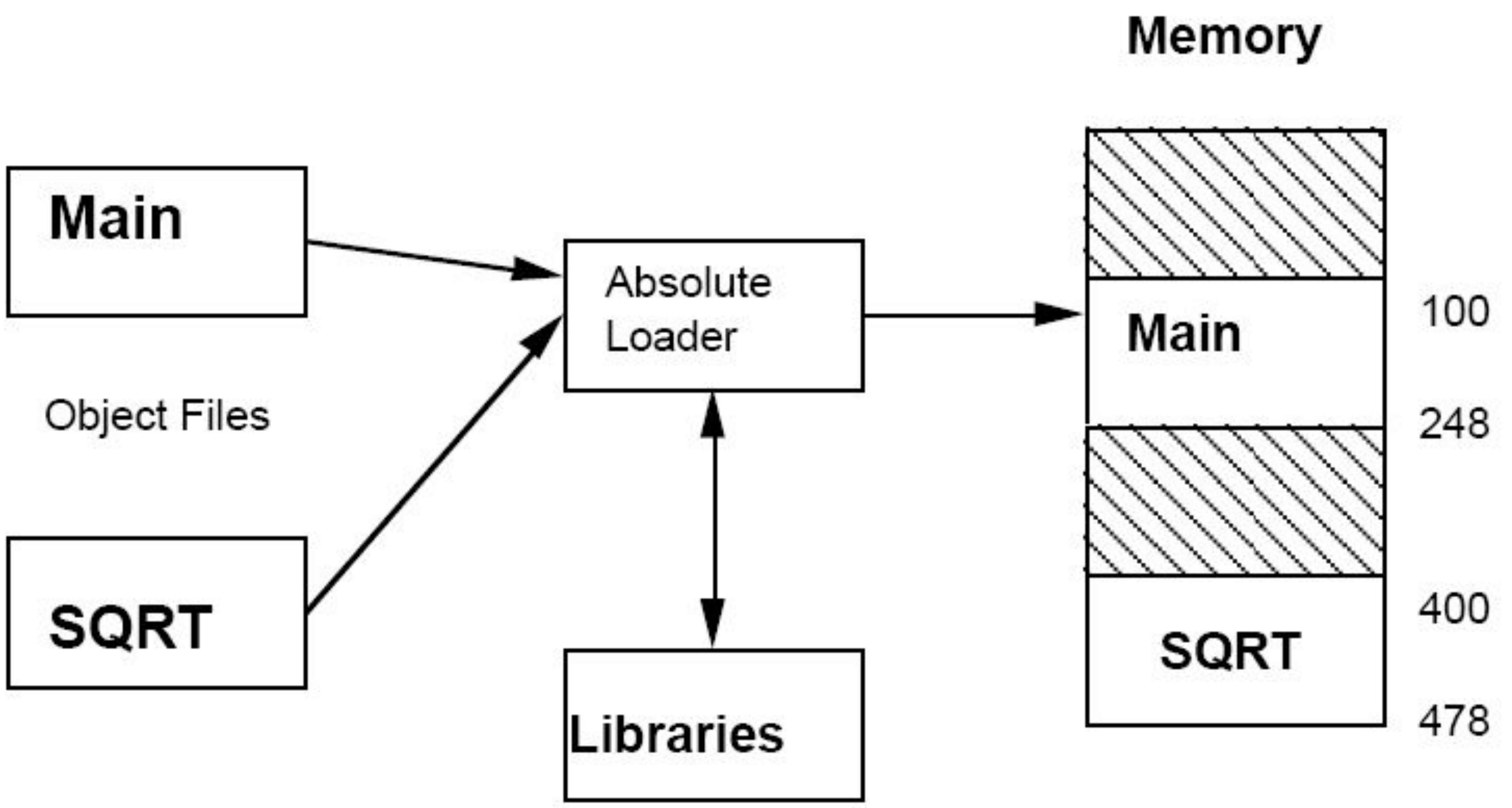
The absolute loader will load the program at memory location x200:

1.- The header record is checked to verify that the correct program has been presented for loading.

2.- Each text record is read and moved to the indicate address in memory

3.- When the "end" record (EOF) is encountered, the loader jumps to the specified address to begin execution.

Absolute Loader



Absolute Loader

- **The Loader accept translated form in form of Image Consisting:**
- **The Text(Instruction & Data both)**
- **The Length of Text**
- **The Starting Address**
- **Advantages of Absolute Loader:**
- **Simple, easy to design and implement.**
- **Since more core memory is available to the user there is no memory limit.**

- **Disadvantages of Absolute Loader:**
- Actual load address must be specified
- The programmer must be careful not to assign two subroutines to the same or overlapping locations

Bootstrap Loader

- Special Type of Absolute Loader.
- When a computer is first tuned on or restarted bootstrap loader is executed.
- This bootstrap loads the first program to be run by computer that is the OS.
- It loads the first address 0x80.

Bootstrapping continued…

- This program does not have the full functionality of an operating system, but it is capable of loading into memory a more elaborated software (i.e. loader2) which in its turn will load the operating system.
- Once the OS has been loaded the loader transfers the control of the computer system to the operating system.

Relocating Loaders

- To avoid possible reassembling of all subroutines when a single subroutine is changed and to perform the tasks of allocation and linking for the programmer the relocating loaders is introduced.
- The execution of the object program is done using any part of the available & sufficient memory.
- The object program is loaded into memory wherever there is room for it.
- The assembler assembles each procedures 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 the entire program and the length of the transfer vector.

Practical relocating loader

- BINARY SYMBOLIC SUBROUTINE (**BSS**) loader such as used in the IBM 7094,IBM 1130,GE 635.
- The output of the relocating assembler using a BINARY SYMBOLIC SUBROUTINE(BSS) scheme is the Object program and information about all other program its references.
- The BSS loader allows many procedure segments but only one data (common)segment.
- The BSS loader scheme is often used on computers with a fixed-length direct address instruction format.

Practical relocating loader

Advantages ::

- Avoids possible reassembling of all Subroutines when a single Subroutine is changed
- Perform the tasks of allocation and linking for the programmer.

Disadvantages ::

- Difficult to implement
- Algorithm is depends on File structure of the object program.
- Slower than Absolute loader

Direct Linking Loaders

- A Direct linking loader is a general relocating loader and is the most popular loading scheme presently used.
- This scheme has an advantage that it allows the programmer to use multiple procedure and multiple data segments.
- In addition, the programmer is free to reference data or instructions that are contained in other segments.
- The direct linking loaders provide flexible

Direct Linking Loaders

- The assembler should give the following information to the loader:

  1)The length of the object code segment

  2)The list of all the symbols which are not defined in the current segment but can be used in the current segment.

  3)The list of all the symbols which are defined in the current segment but can be referred by the other segments.

USE Table

USE Table

- The list of symbols which are not defined in the current segment but can be used in the current segment are stored in a data structure called USE table.
- The USE table holds the information such as name of the symbol, address, address relativity.

Definition Table

- The list of symbols which are defined in the current segment and can be referred by the other segments are stored in a data structure called DEFINITION table.
- The definition table holds the information such as symbol, address.

Direct Linking Loaders

Object deck:

 The Object Deck is used by various IBM 360 or 370 Direct Linking Loaders.

- There are 4 Sections of Object Desk :
- 1: External Symbol Dictionary cards(ESD).
- 2: Instruction and Data Cards, called “TEXT” of Program (TXT).
- 3: Relocation and Linkage Directory cards (RLD)

ESD card

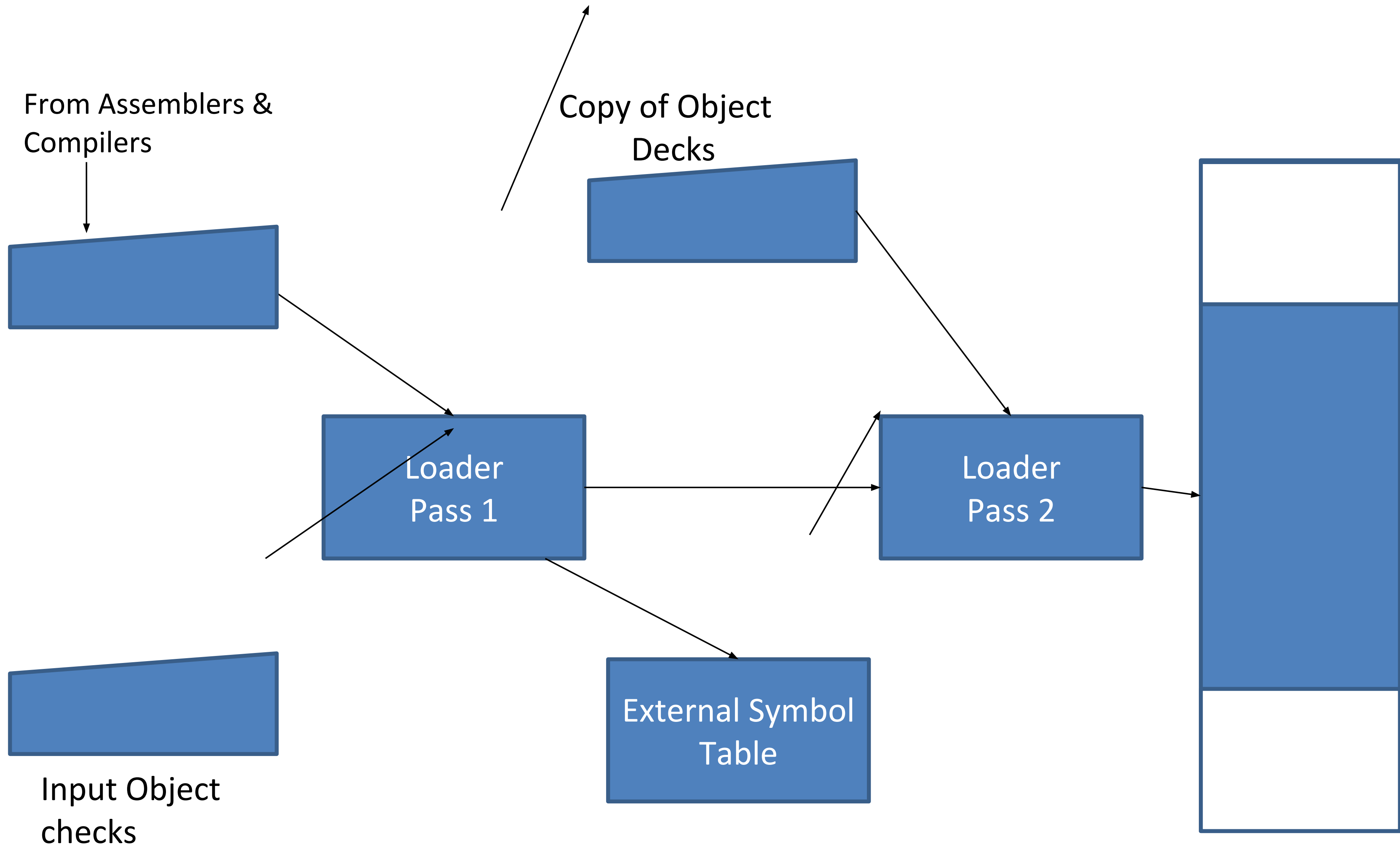
- The ESD cards contain the information necessary to build the external symbol dictionary or symbol table.
- External symbols are symbols that can be referred beyond the subroutine level.

There are 3 types of external symbols:

- 1: Segment Definition (SD).
- 2: Local Definition (LD)

Global External Symbol Table(GEST):

- The GEST is used to store the external symbols defined by means of segment definition or local definition or an External Symbol Dictionary(ESD) card.
- When these symbols are encountered during Pass 1, they are assigned an absolute core address; this address is stored along with the symbol in GEST.
- The reader may wish to review the discussion on symbol tables and searching/sorting


From Assemblers & Compilers
Copy of Object Decks
Loader Pass 1
Loader Pass 2
External Symbol Table
Input Object checks