Loaders

Loaders

- A loader is a system program that performs the loading function.
  - many also support relocation & linking
  - others have a separate linker and loader
- Basic Functions
  - bringing an object program into memory
  - starting its execution

Input

- Object program:
  - contains translated instructions and data from the source program.
  - specifies addresses in memory where these items are to be loaded.

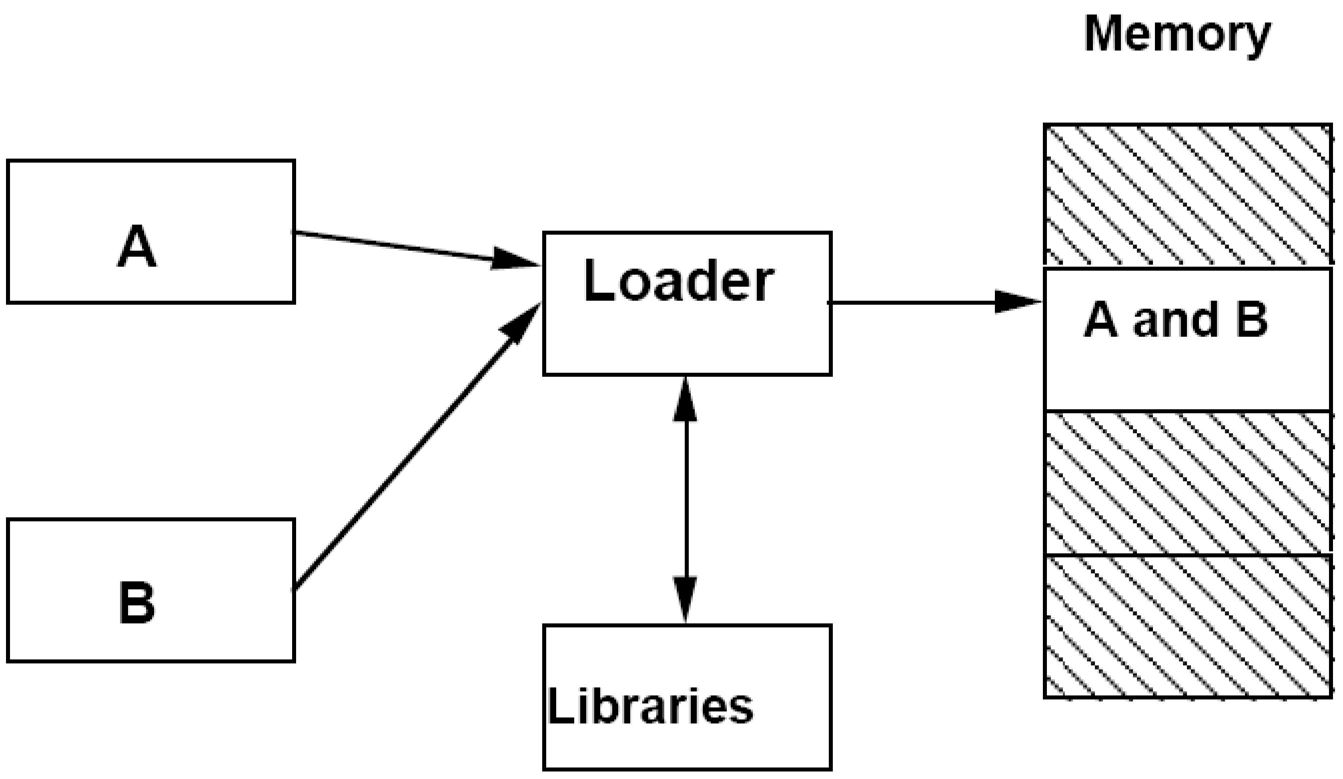
Basic Functions

- ***Allocation***: allocate space in memory for the programs
- ***Linking:*** Resolve symbolic references between object files
  - combines two or more separate object programs
  - supplies the information needed to allow references between them

Basic Functions

- ***Relocation****:* Adjust all address dependent locations, such as address constants, to correspond to the allocated space
  - modifies the object program so that it can be loaded at an address different from the location originally specified
- ***Loading****:* Physically place the machine instructions and data into memory

Basic Functions



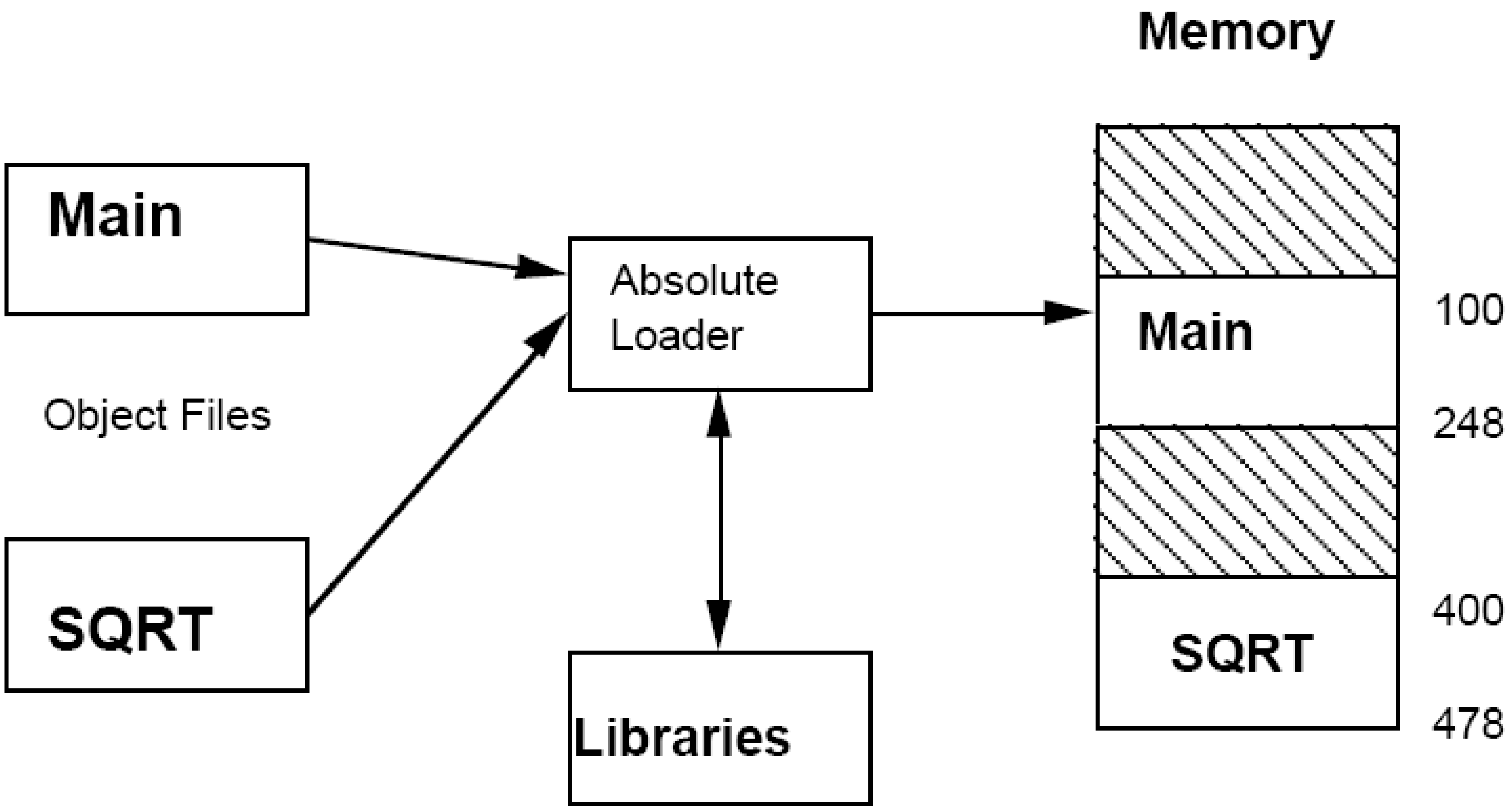
Design of an Absolute Loader

- 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.

Loader Schemes

- Compile and Go
  - The assembler run in one part of memory
  - place the assembled machine instructions and data, as they are assembled, directly into their assigned memory locations
  - When the assembly is completed, the assembler causes a transfer to the starting instruction of the program

Absolute Loader



Disadvantages

- A portion of memory is wasted because the memory occupied by the assembler is unavailable to the object program.
- It is necessary to re-translate (assemble) the user's program file every time it is run.
- It is very difficult to handle multiple segments, especially if the source programs are in different.

Disadvantages

- If changes were made to MAIN that increased its length to more than 300 bytes
  - the end of MAIN (at 100 + 300 = 400) would overlap the start of SQRT (at 400)
  - It would then be necessary to assign SQRT to a new location
    - changing its START and re-assembling it?!
- Furthermore, it would also be necessarily to modify all other subroutines that referred to the address of SQRT.

A Simple Bootstrap Loader

- Automatically executed when the computer is first turned on
- Loads the first program to be run: usually the O/S itself begins at address 0 in memory
  - loads the O/S starting at address 80
  - After all code is loaded, bootstrap jumps to address 80.
  - No H or E records, no control information

Disadvantages of Absolute Loaders

- Actual load address must be specified
- The programmer must be careful not to assign two subroutines to the same or overlapping locations
- Difficult to use subroutine libraries (scientific and mathematical) efficiently
  - important to be able to select and load exactly those routines that are needed

Disadvantages of Absolute Loaders

- Allocation - by programmer
- Linking - by programmer
- Relocation - None required-loaded where assembler assigned
- Loading - by loader

General Loader Scheme

- Linking
- Relocation
- Loading

Subroutine Linkages

- The main program A wishes to transfer to subprogram B.
- The programmer, in program A, could write a transfer instruction (e g, BSR B) to subprogram B.
- The assembler does not know the value of this symbol reference and will declare it as an error

Externals and Entries

- The assembler pseudo-op EXT followed by a list of symbols indicates that the symbols are defined in other programs but referenced in the present program
- If a symbol is defined in one program and referenced in others,
  - insert it into a symbol list following the pseudo-op ENT.

```
MAIN    ORG     $10
        EXT     SUBROUT
        BSR     SUBROUT
DONE    HLT
```

Relocation

- Relocating loaders or relative loaders:
  - loaders that allow for program relocation.
- Two methods for specifying relocation as part of the object program:
- **1. A Modification record**
  - describe each part of the object code that must be changed when the program is relocated
  - M0000_16

Second Method

- **Bit mask:** A relocation bit/byte associated with each word of object code
  - S for Absolute: does not need modification
  - R for Relative: needs relocation
  - X for external.
- Example
- T00106119SFE00S4003S0E01R

Two Pass Direct Linking Loader

- Pass 1
  - **Allocate** and assign each program location in core.
  - Create a symbol table filling in the values of the external symbols.
- Pass 2
  - **Load** the actual program text.
  - Perform the **relocation** modification of any address constants needing to be altered.
  - Resolve external references. (**linking**)

Data Structures

- External Symbol Table (ESTAB)
  - stores the name and address of each external symbol in the set of programs being loaded.
  - Indicates in which program the symbol is defined.
  - A hash table is generally used.
- Program Load Address (PROGADDR)
  - beginning address in memory where the linked program is to be loaded.
  - supplied by the O/S

More Databases

- Control Section Address (CSADDR)
  - starting address assigned to the CS currently being scanned by the loader
  - Its value is added to all relative addresses within the control section to convert them to actual addresses

The Algorithm

- Pass 1: concerned only w/Header records
  - PROGADDR is obtained from O/S
  - CSADDR is set accordingly
  - All external symbols are entered into External Symbol Table (ESTAB)
    - Their addresses are obtained by adding values specified in header to CSADDR (- First ORG?!)
  - Starting address and length of each CS are determined. CSADDR = CSADDR + CSLEN
  - Print Load Map

ESTAB

Program/ CS	Symbol	Address	Length
Test		0040	0046
	EXE	0060	
ProgA		0086	0010
	LISTA	0090	
ProgB		0096	6

Pass II

- Does actual loading, relocation, and linking
- As each Text record is read
  - The object code is moved to the specified address (plus the current value of CSADDR)
  - When “R” is encountered, the value is added or subtracted from the indicated location in memory
  - When “X” is encountered resolve symbol from ESTAB
  - Last step: transfer control to loaded program to begin execution, as indicated in the End record

Relocating Loaders

- Allocating subroutines to prevent reassembling the code every time a subroutine changes
- Binary Symbolic Subroutine (BSS) Loader
  - The program length information is for allocation.
  - Bit Mask is used for relocation
  - The transfer vector is used to solve the problem of linking

Binary Symbolic Subroutine Loader

- The assembler assembles Provides the loader
  - Object program + relocation information
  - Prefixed with information about all other program it references (transfer vector).
  - The length of the entire program
  - The length of the transfer vector portion

Transfer Vector

- A transfer vector consists of
  - addresses containing names of the subroutines referenced by the source program
  - if a Square Root Routine (SQRT) was referenced and was the first subroutine called, the first location in the transfer vector could contain the symbolic name SQRT.
  - The statement calling SQRT would be translated into a branch to the location of the transfer vector associated with SQRT

The loader

- loads the text and the transfer vector
- loads each subroutine identified in the transfer vector.
- place a transfer instruction to the corresponding subroutine in each entry in the transfer vector.
  - The execution of the call SQRT statement result in a branch to the first location in the transfer vector
  - which contains a branch to the location of SQRT.

Example

```
MAIN    START
        EXTERNAL    SORT
        EXTERNAL    ERR4
        LOAD        1,=F9
        BALINK      14,SQRT
        COMPARE     1,=F3
        BNE ERR
        HLT
=9      DATA        9
=3      DATA        3
END
```

```
LC      r / s / e
0       00          SORT
4       00          ERR
8       01          LOAD 1,1C
C       01          BALINK 14, 0
10      01          COMPARE 1,20
14      01          BNE 4
18      00          HLT
1C      00          0009
20      00          0003
```

After Loading Using BSS Scheme

- Program Length
  - 20 bytes
- Transfer vector
  - 8 bytes

0	400	BALINK 14,448
4	404	BALINK 14,526
8	408	LOAD 1,41C
C	40C	BALINK 14,400
10	410	COMPARE 1,420
14	414	BNE 404
18	418	HLT
1C	41C	0009
20	420	0003

BSS Scheme Disadvantages

1. the transfer vector increases the size of the object program in memory
2. the BSS loader does not facilitate access to data segments that can be shared
   – the transfer vector linkage is only useful for transfers or BSRs
   – not well suited for loading or storing external data (data located in another procedure segment)

Direct Linking Loader

- The assembler provides
  1. The length of segment
  2. A list of all entries and their relative location within the segment
  3. A list of all external symbols
  4. Information as to where address constants are loaded in the segment and a description of how to revise their values.
  5. The machine code translation of the source program and the relative addresses assigned

Example

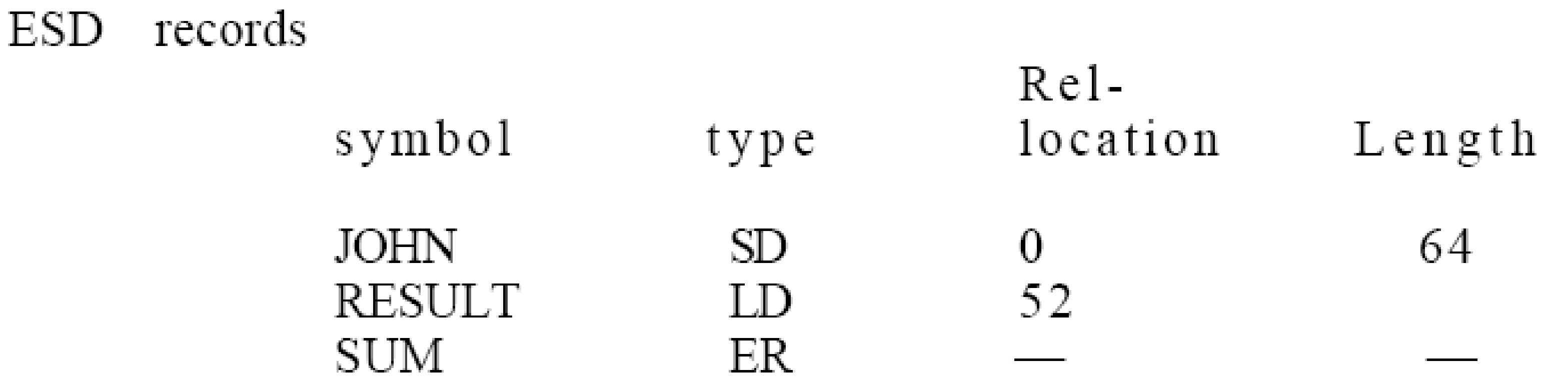
```
John     START
         ENTRY      RESULT
         EXTERNAL                         SUM
         LOAD       1,POINTER             0     LOAD      1,48
         LD-ADDR    15 ASUM               4     LD-ADDR   15,56
         BALR       14 15                 8     BALR      14 15
         STORE      l RESULT              12    STORE     1,52
         HLT                              1C    HLT       0,0


TABLE    DC         1,7,9,10,13           28    0001
                                          2A    0007
                                          2C    0009
                                          30    000A
                                          34    000D


POINTER  DATA       ADDR(TABLE)           48    0028
RESULT   NUM        0                     52    0000
ASUM     DATA       ADDR(SUM)             56    ????      EXTERNAL
         END
```

Assembler records

- External Symbol Dictionary (ESD) record: Entries and Externals
- (TXT) records control the actual object code translated version of the source program.
- The Relocation and Linkage Directory (RLD) records relocation information
- The END record specifies the starting address for execution

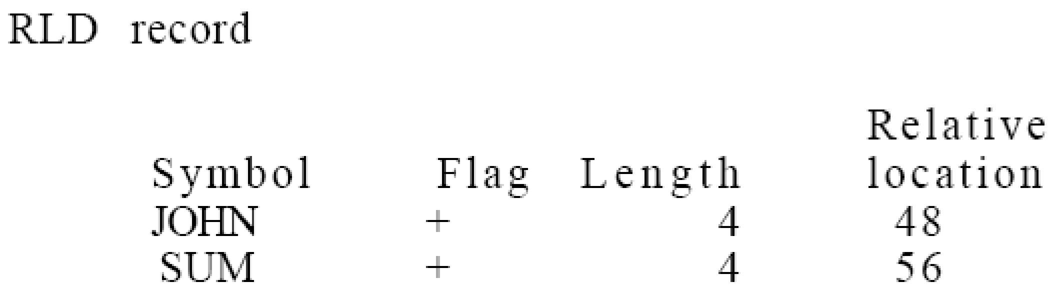
ESD and RLD

- SD: Segment Definition
- Local Definition
- External Reference



ESD records

symbol	type	Rel- location	Length
JOHN	SD	0	64
RESULT	LD	52	
SUM	ER	—	—



RLD record

Symbol	Flag	Length	Relative location
JOHN	+	4	48
SUM	+	4	56

Disadvantages of Direct Linking

- 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.
- Though smaller than the assembler, the loader absorbs a considerable amount of space
  - Dividing the loading process into two separate programs a binder and a module loader can solve these problems.

Binder

- A binder is a program that performs the same functions as the direct linking loader
  - allocation, relocation, and linking
- Outputs the text in a file rather than memory
  - called a **load module**.
- The module loader merely has to physically load the module into memory.

Binder Classes

- Core image builder:
  - Produces a load module that looks very much like a "snapshot" or "image" of a section of core,
  - Called Core image module.
- Link editor, can keep track of the relocation Information
  - The load module can be further relocated
  - The module loader must perform allocation and relocation as well as loading
  - No linking.

Disadvantage

- If a subroutine is referenced but never executed
  - if the programmer had placed a call statement in the program but was never executed because of a condition that branched around it
  - the loader would still incur the overhead or linking the subroutine.
- All of these schemes require the programmer to explicitly name all procedures that might be called.

Dynamic Loading

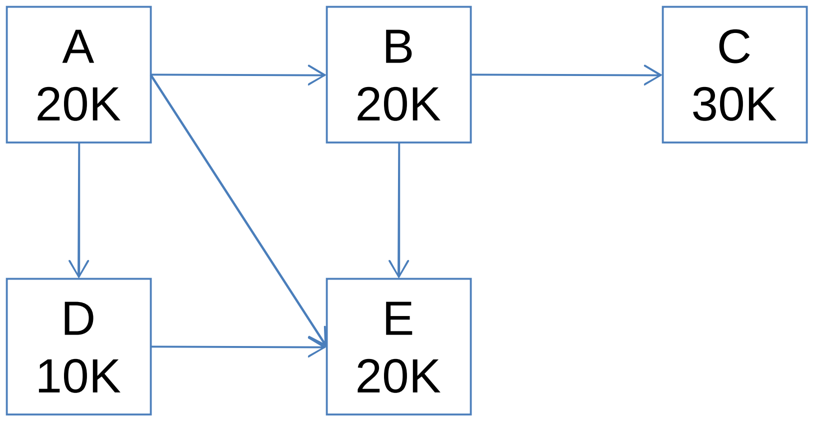
- If the total amount of memory required by all subroutines exceeds the amount available
- The module loader loads the only the procedures as they are needed.
  - Allocating an overlay structure
- The Flipper or overlay supervisor is the portion of the loader that actually intercepts the "calls" and loads the necessary procedure.

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.

Longest Path Overlay Structure

100k vs 70k needed



Dynamic Linking

- The loading and linking of external references are postponed until execution time.
- The loader loads only the main program
- If the main program should
  - execute a branch to an external address,
  - reference an external variable
- The loader is called
  - Only then has the segment containing the external reference loaded.

Design of Direct Linking Loader

```
1     O    PGA       START
2                    ENTRY      A1,A2
3                    EXTERNAL   B1,PGB
4    20    A1
5    30    A2
6    40              ADDR       A(A1)
7    44              ADDR       A(A2+15)
8    48               ADDR      A(A2-A1-3)
9    52              ADDR       A(PGB)
10   56               ADDR      A(B1+PGB-A1+4)
11                   END

12    0    PGB       START
13                   ENTRY      B1
14                   EXTERNAL   A1,A2
15   16    B1
16   24              ADDR       A(A1)
17   28              ADDR       A(A2+15)
18   32              ADDR       A(A2-A1-3)
19                   END
```

ESD Records

Variable Name	Type	address	Length	Reference: to source line
PGA	Program Name	0	60	1
A1	Local variable	20		2
A2	Local variable	30		2
PGB	External Var			3
B1	External Var			3

TXT Records

Relative Address	Contents	What the assembler did	Reference to source line
40	20		6
44	45	=30+15	7
48	7	30-20-3	8
52	0	unknown	9
56	-16	-20+4	10

RLD Records

Relative Address	Arithmetic Operator	Variable Name	Length	Reference to source line
40	+	PGA	4	6
44	+	PGA	4	7
52	+	PGB	4	9
56	+	B1	4	10
56	+	PGB	4	10
56	-	PGA	4	10

ESD Records

Variable Name	Type	address	Length	Reference: to source line
PGB	Program Name	0	36	12
B1	Local Variable	16	N/A	13
A1	External Var	unknown	N/A	14
A2	External Var	unknown	N/A	14

TXT Records

Relative Address	Contents	What the assembler did	Reference to source line
24	0	=0 A1 unknown	16
28	15	=15 A2 unknown	17
32	-3	=-3 A1,A2 unknown	18

RLD Records

Relative Address	Arithmetic Operator	Variable Name	Length	Reference to source line
24	+	A1	4	16
28	+	A2	4	17
32	+	A2	4	18
32	-	A1	4	18

Algorithm

- Pass 1
  - Allocate Segments
    - Initial Program Load Address (IPLA)
    - Assign each segment the next table location after the preceding segment.
  - Define Symbols
    - SD
    - LD
    - ER?!

Pass 2: load text and relocate/link

- ESD record types is processed differently.
  - SD The LENGTH of the segment is temporarily saved in the variable SLENGTH.
  - LD does not require any processing during pass 2.
  - ER The Global External Symbol Table (GEST) is searched for match with the ER symbol
  - If found in the GEST, Substitute value
  - If it is not found → error

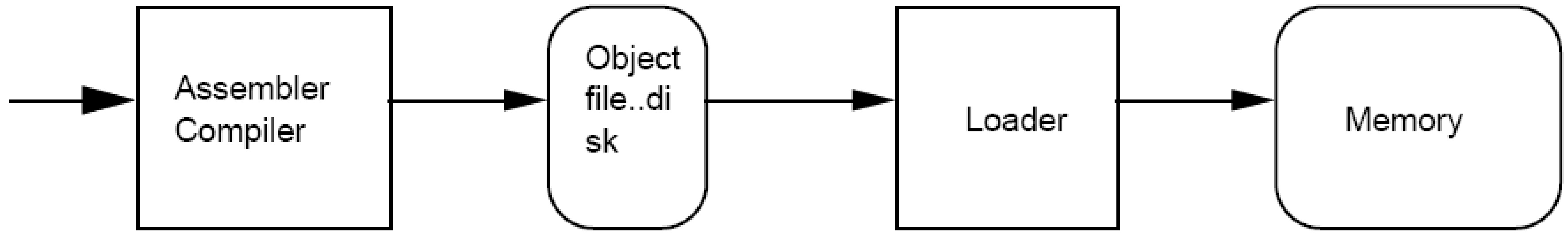
Pass 2

- TXT: the text is copied from the record to the relocated core location (PLA + ADDR).
- RLD: The value to be used for relocation and linking is extracted from the GEST
  - If Flag is Plus the value is added, if Flag is minus the value is subtracted from the address constant
- The relocated address of the address constant is the sum of the PLA and the ADDR field specified on the RLD record.

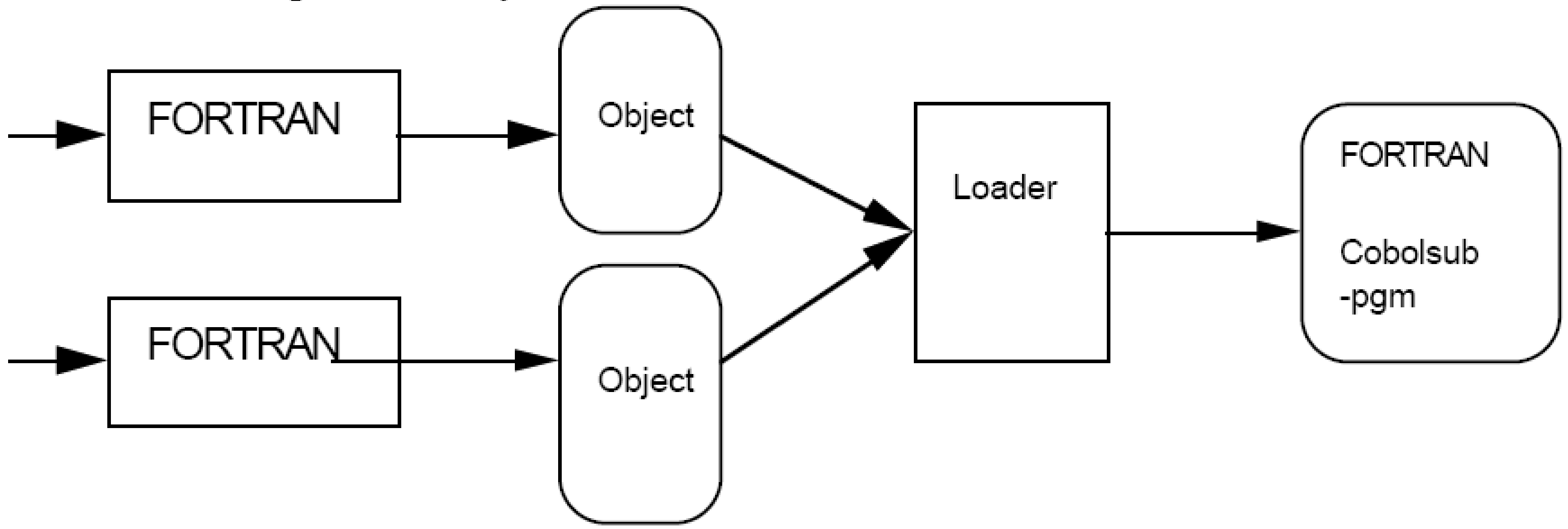
Pass 2

- END: The execution start address is relocated by the PLA
  - The Program Load Address is incremented by the length of the segment and saved in SLENGTH, becoming the PLA for the next segment.
- LDT/E0F record
  - The loader transfers control to the loaded program at the address specified by current contents of the execution, address variable (EXADDR)

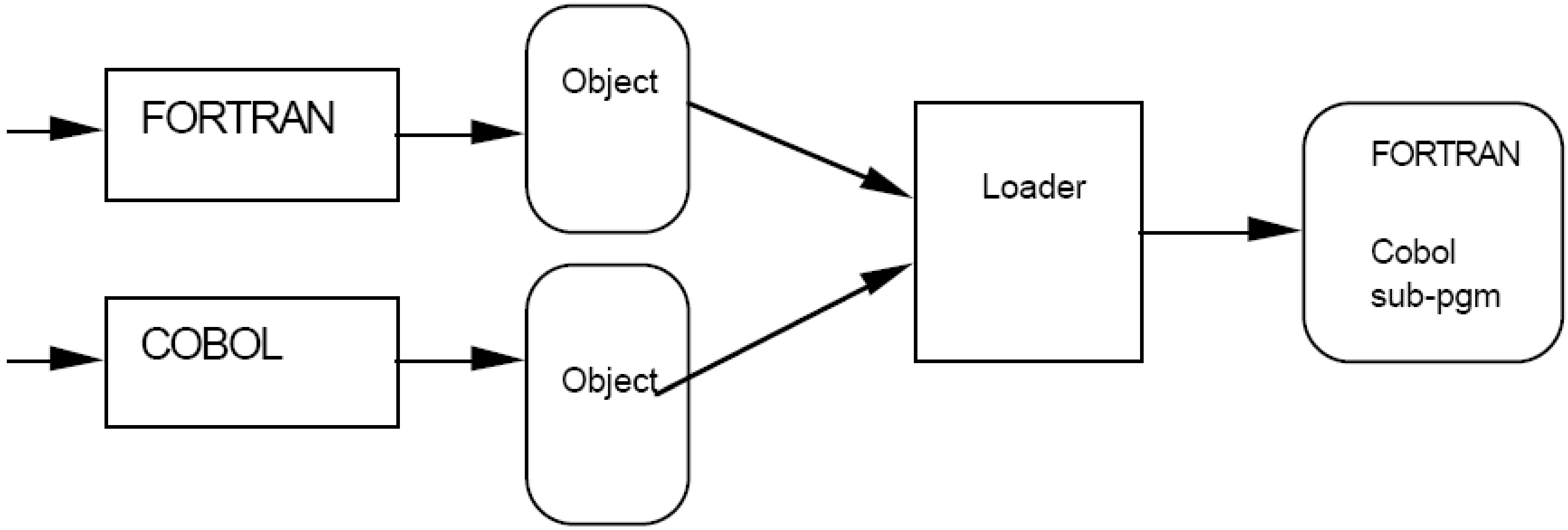
Case 1: Initial Case Absolute Loader



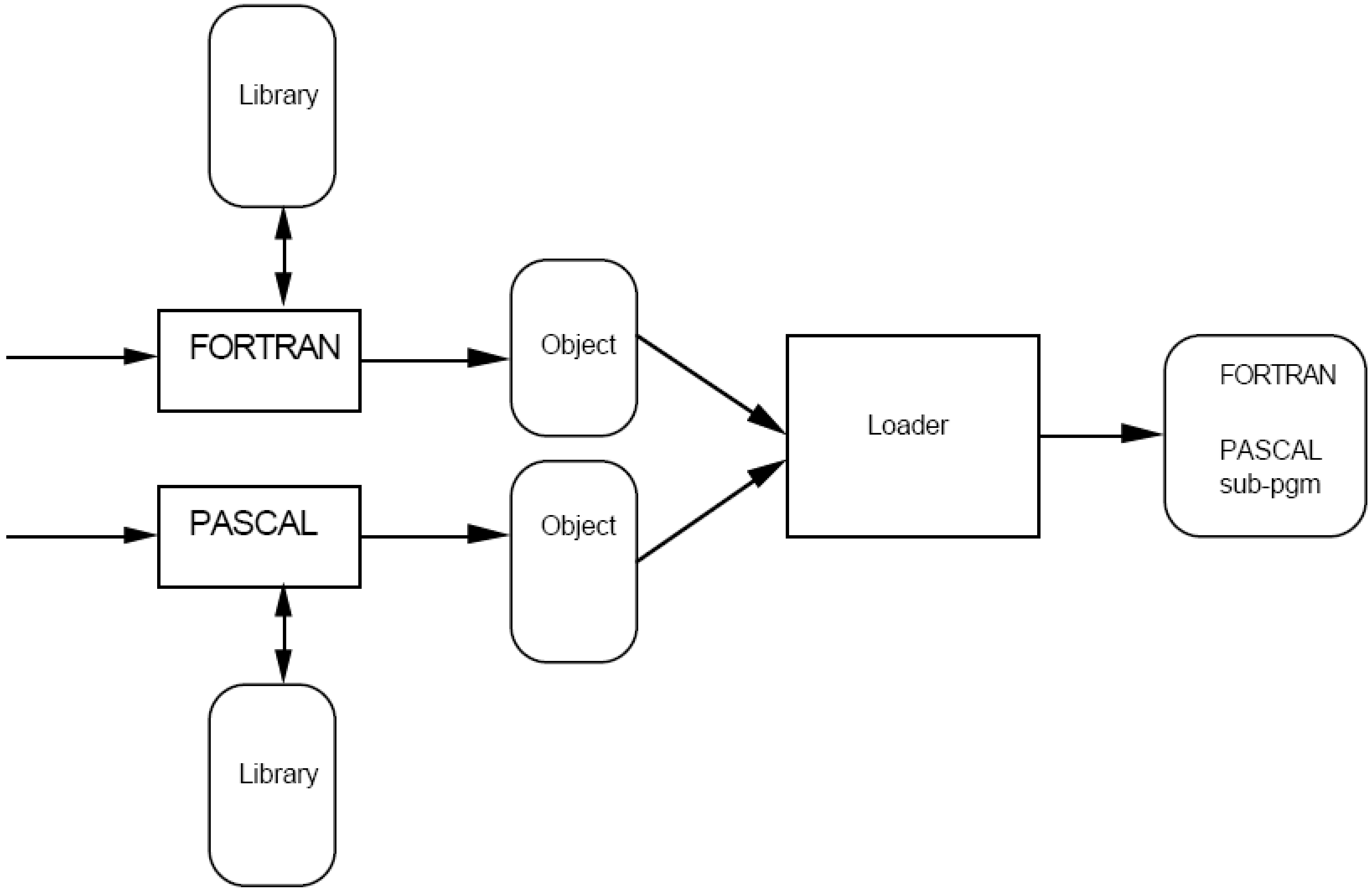
Case 2: Multiple Programs in the same Language, Compiled Independently



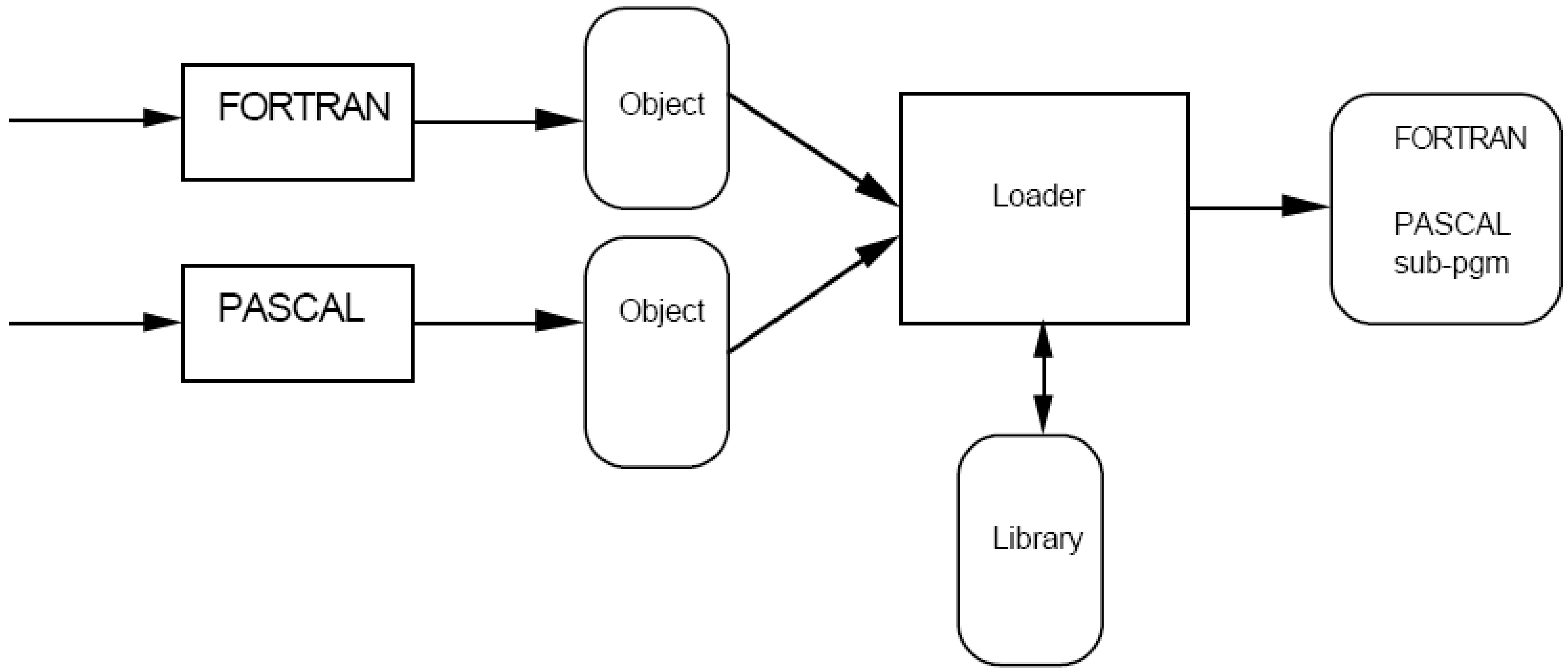
Case 3: Multiple Source Programs Different Languages



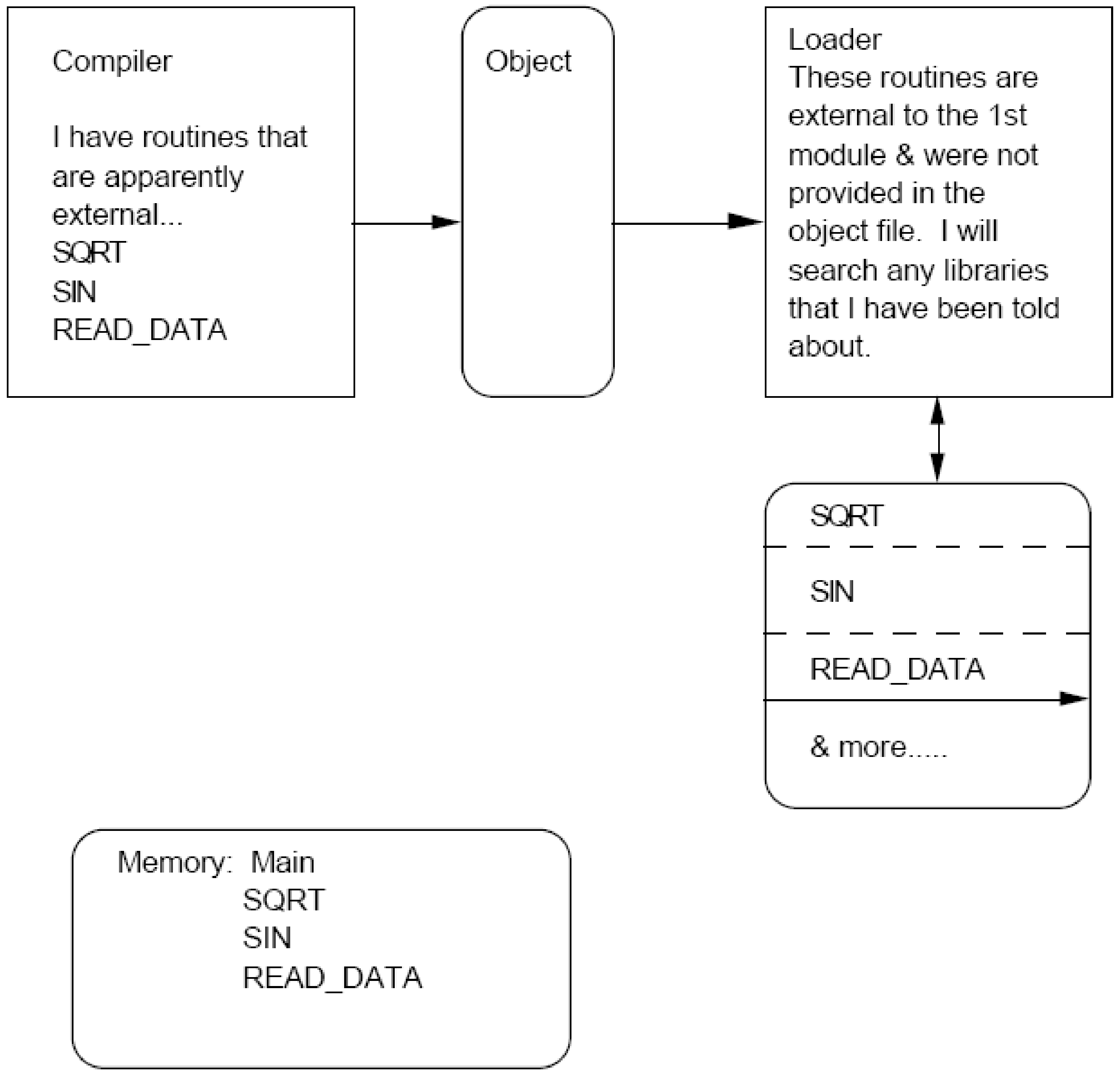
Case 4: Source Libraries

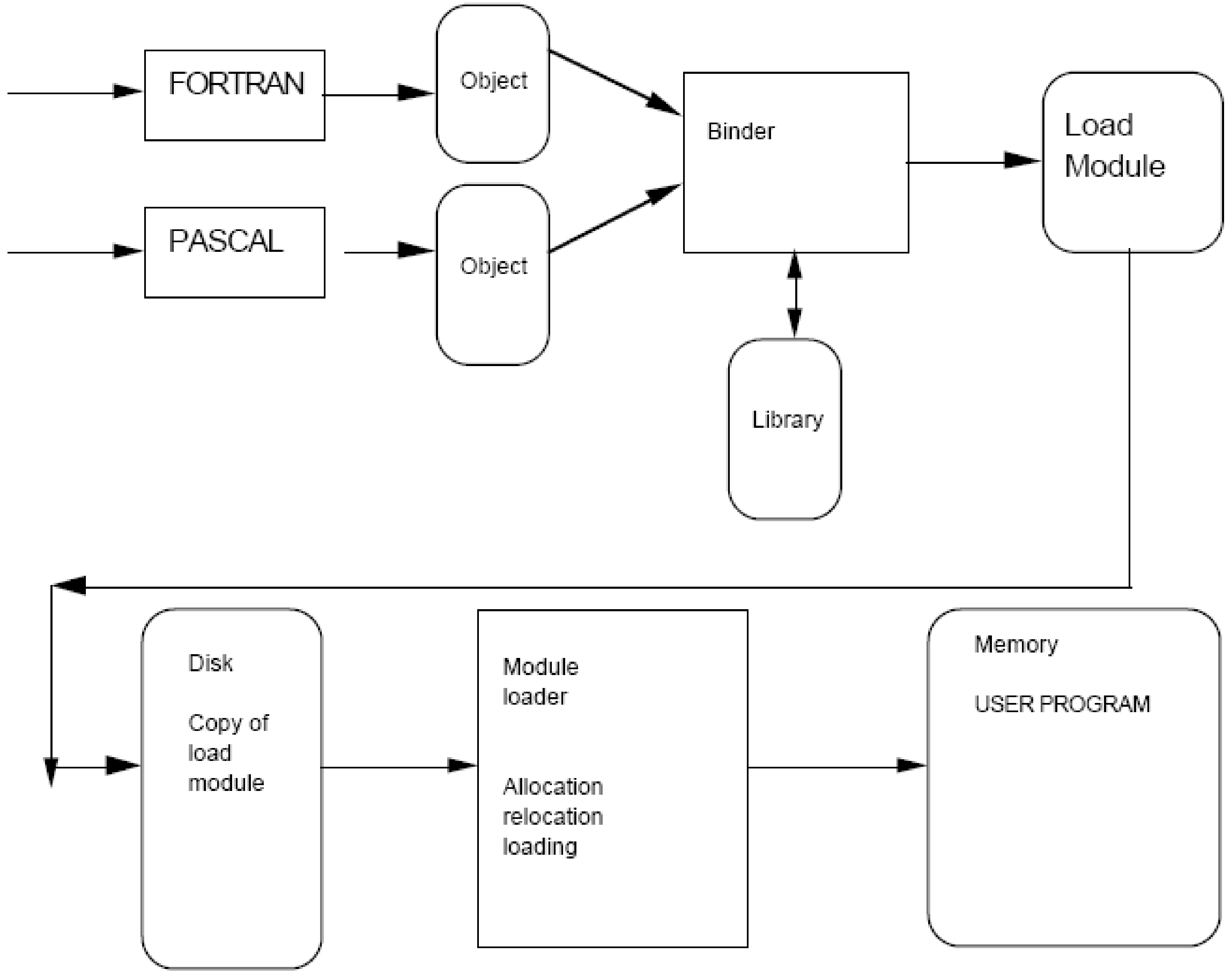


Case 5: Object Libraries



Case 6: Load Libraries


Compiler
I have routines that are apparently external...
SQRT
SIN
READ_DATA
Object
Loader
These routines are external to the 1st module & were not provided in the object file. I will search any libraries that I have been told about.
SQRT
SIN
READ_DATA
& more.....
Memory: Main
SQRT
SIN
READ_DATA


FORTRAN
PASCAL
Object
Object
Binder
Library
Load Module
Disk
Copy of load module
Module loader
Allocation relocation loading
Memory
USER PROGRAM

Final Exam

14/1/2008	الاثنين	12:45-10:45
Salah Al-Deen		