Pune Vidyarthi Griha's

COLLEGE OF ENGINEERING, NASHIK.

" LINKER "

By

Prof. Anand N. Gharu

(Assistant Professor)

PVGCOE Computer Dept.

08th Jan 2018

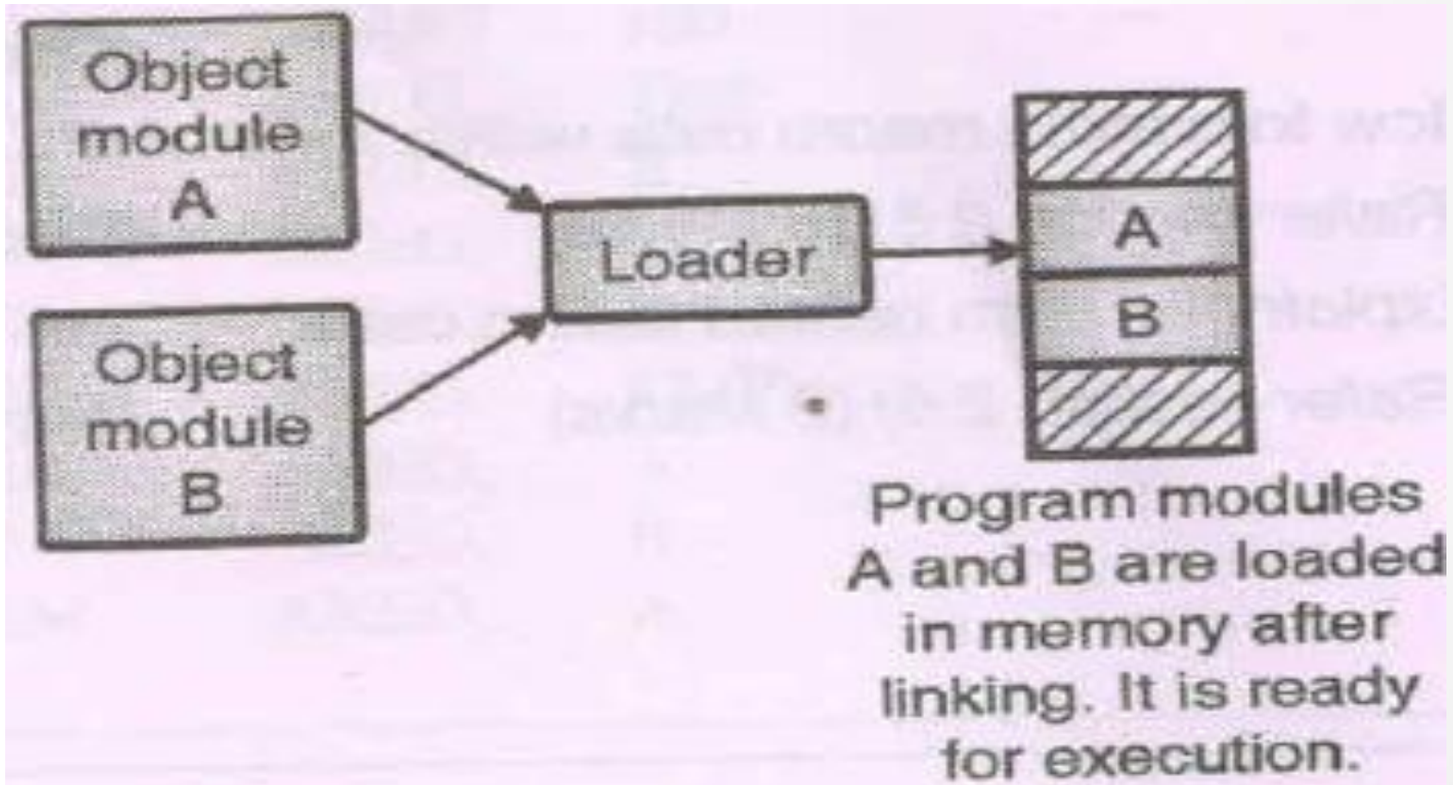
CONTENTS :-

1. Relocation & Linking Concepts
2. Self Relocating Programs
3. Static & Dyanamic link library
4. Dynamic link library (DLL)
5. Callback Function
6. Compare callback & Normal function
7. Loading phases using java

Introduction

BASIC FUNCTION OF LINKING:-

- ***Allocation***
- ***Linking***
- ***Relocation***
- ***loading***

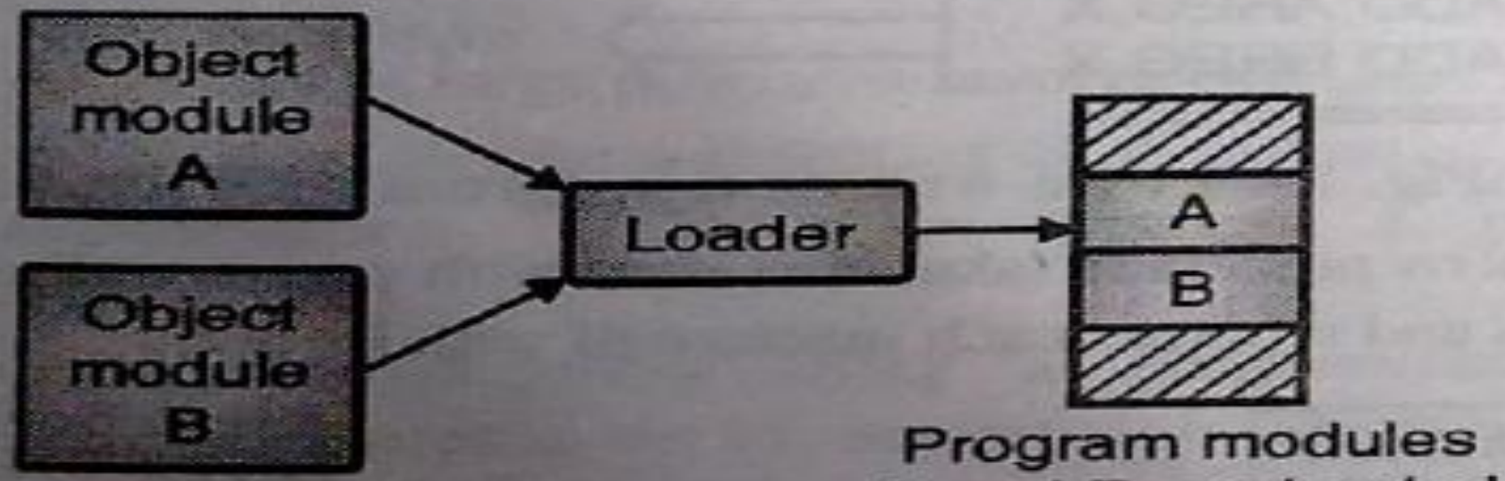


(S5.1)**Fig. 3.1.1 : General loading scheme**

BASIC FUNCTION OF LOADER

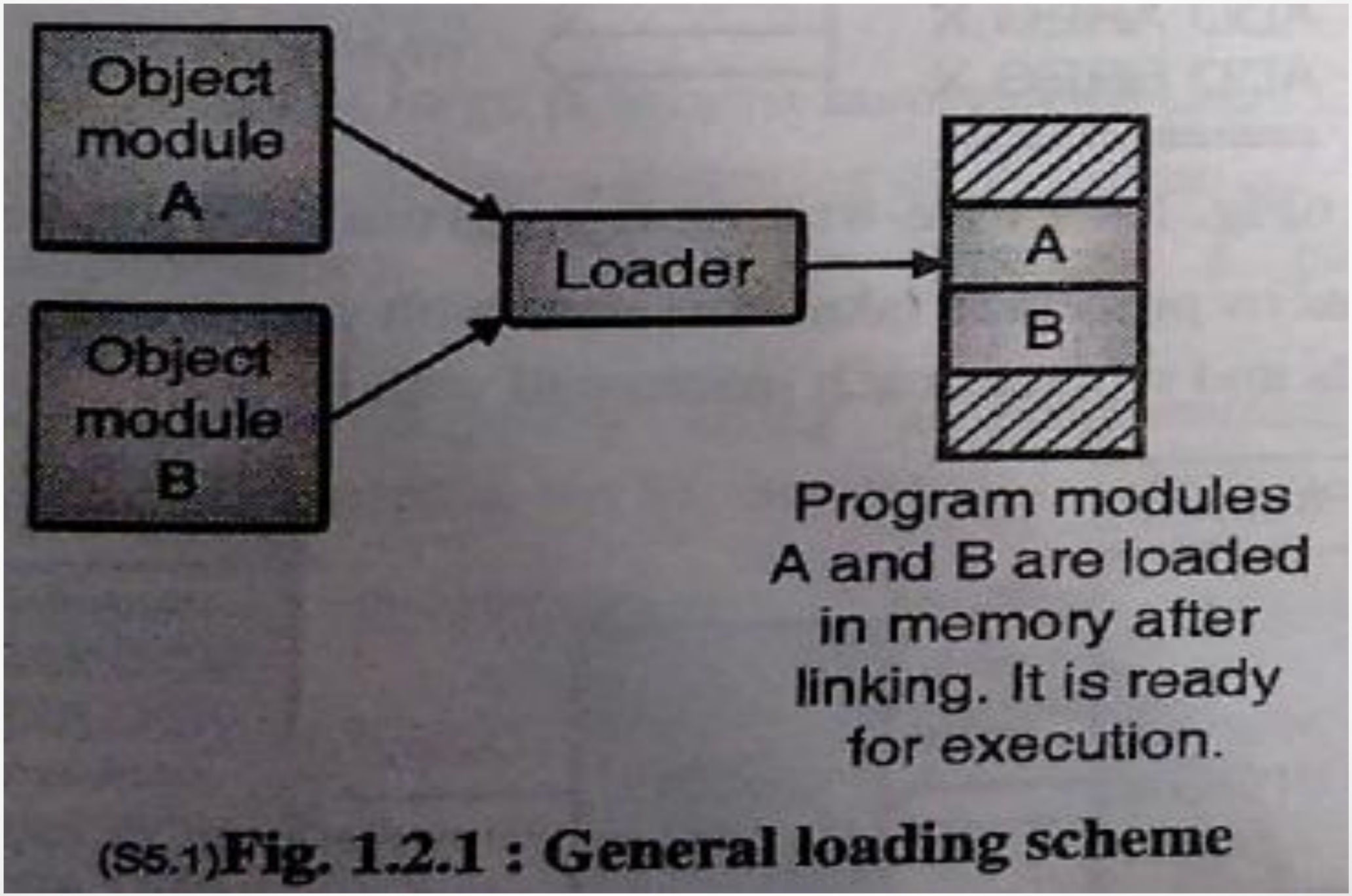
- These functions are :
  1. **Allocation** of space in main memory for the programs.
  2. **Linking** of object modules with each other. Linking involves resolving of symbolic references between object modules.
  3. Adjust all address dependent locations, such as address constants, to correspond to the allocated space. It is also called **relocation.**
  4. Physically **loading** the machine instructions and data into the main memory.

  A general loading scheme is shown in the Fig. 1.2.1.



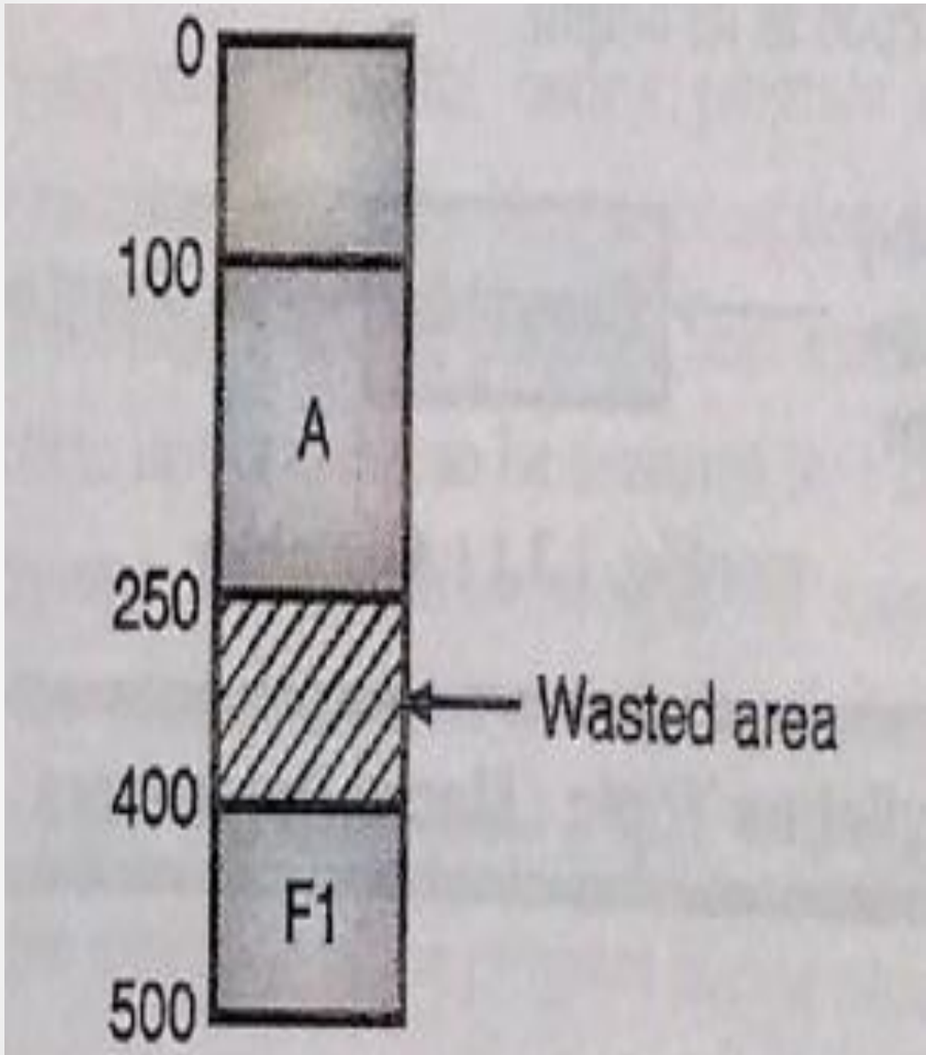
(S5.1)**Fig. 1.2.1 : General loading scheme**

GENERAL LOADING SCHEME



(S5.1)**Fig. 1.2.1 : General loading scheme**

RELOCATION



(S5.2)Fig. 1.2.2 : Case I of relocation

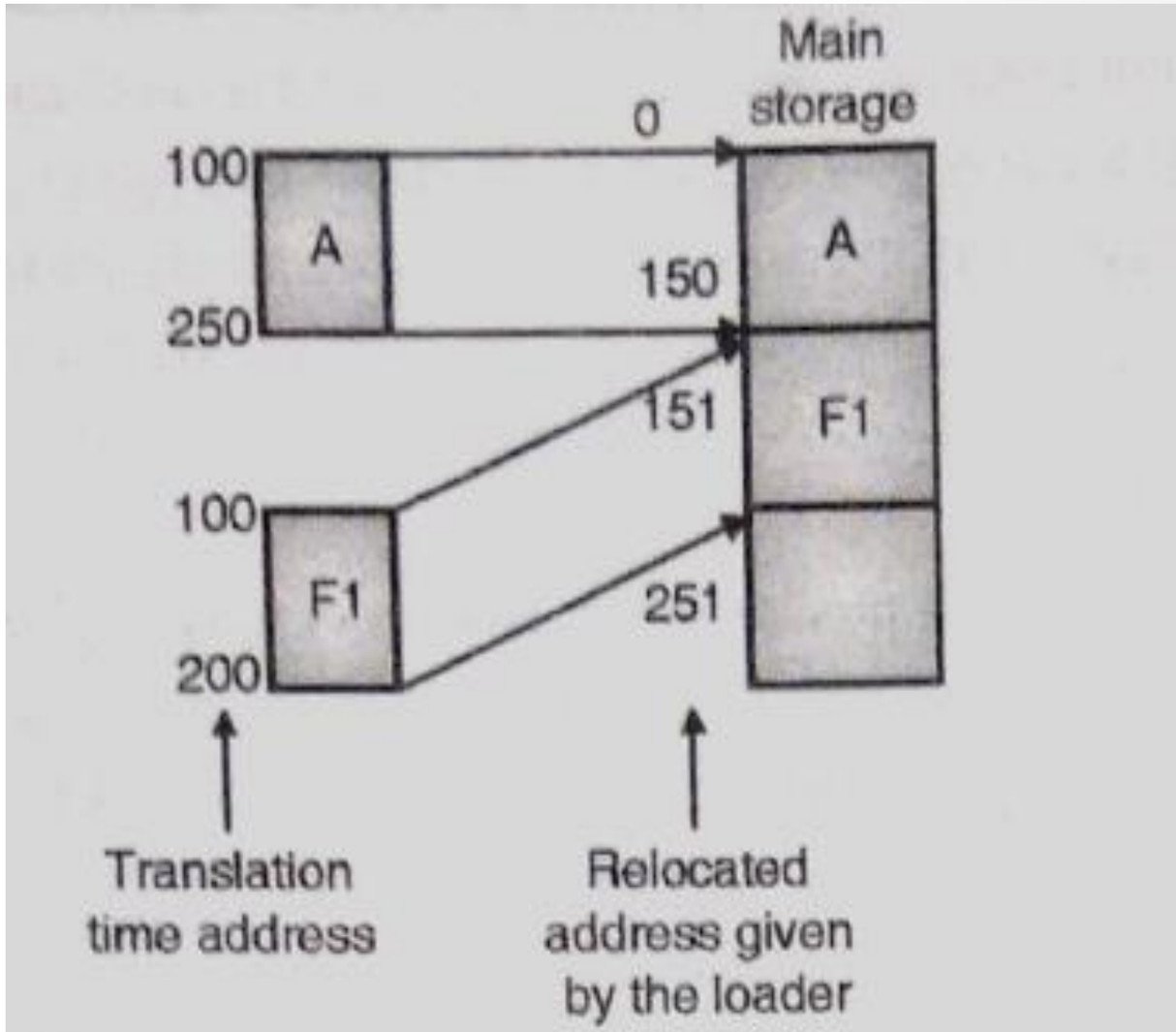


Fig. 1.2.3 : Relocation to avoid address conflict or storage waste

Self Relocating Programs

Program relocatability 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 scheme, there are two basic types of relocation :

1. Static
2. Dynamic

A self relocating program is a program which can perform the relocation itself. It contains the followings :

1. A table of information about address sensitive instructions in the program
2. Relocating logic that can perform the relocation of the address sensitive instructions.

Self relocating program contain the relocating logic that can perform the relocation of the address sensitive instructions. So there is no need of a linker in self-relocating programs.

STATIC & DYNAMIC LINK LIBRARIES

4.3.1 Static Linking

A static linker takes object files produced by the compiler including library functions and produces an 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.

— Another disadvantage of static linking is that newer versions of the library routines must be re-linked into executable.

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.

There are several advantages to this approach :

1. Dynamically linked shared libraries are easier to create.
2. Dynamically linked shared libraries are easier to update.
3. Dynamic linking provides automatic sharing of code. If multiple applications require the same routine, operating system loads a single copy of the routine and links it with multiple applications.
4. It is easy to add new functionality to existing library. These functions could be useful in a variety of applications.
5. Dynamic linking permits a program to load and unload routines at runtime.

DYNAMIC LINK LIBRARIES

DLL is Microsoft implementation of shared library in windows. The file formats for DLL are the same as windows EXE files. A DLL can contain :

1. Code 2. Data 3. Resources.

With dynamic linking, shared code is placed into a single, separate file. The programs that call this file are connected to it at run time, with the operating system performing the linking.

While the DLL code may be shared, the data is private except where shared data is explicitly requested by the library. Each process using the DLL has its own copy of all the DLL's data. Sharing of DLL's data section allows interprocess communication through this shared memory.

☞ **Import libraries**

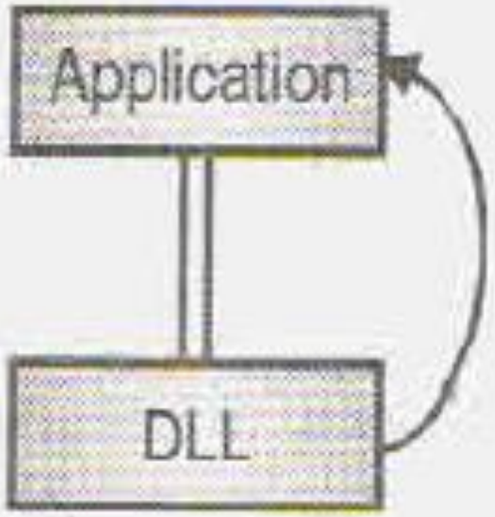
Linking to dynamic libraries is usually handled by linking to an import library.

- The created executable file contains an import address in import address table (IAT). DLL function calls are referenced through this address. Each referenced DLL function contains its own entry in the IAT. At run-time, the IAT is filled with appropriate addresses that point directly to a function in the separately loaded DLL.
- The import libraries for DLL have - lib extension. For example, kernel32.dll, the primary dynamic library for window's base functions such as file creation and memory management, is linked via kernal32.lib.

CALLBACK FUNCTION

A callback function is a function which we write, but it is called by some other program or module, such as windows or DLL's.

For example, a DLL may control several clients, when a certain event occurs from the DLL, the callback function in the client is called.



(S5.14)**Fig. 4.3.1 : A DLL calling a function in an application via a call back**

COMPARE CALLBACK & NORMAL FUNCTION

Normally, an application program calls functions in the DLL. Sometimes, however, the calling application needs periodic information from the DLL, while a processing is taking place. For example, if the DLL is doing some lengthy calculations, it would be useful for the calling application to receive periodic reports of the calculation, such as the percentage complete. The calling application could use this information to display a progress bar.

Callback functions can be used to provide event handling in languages that do not have built in events.

LOADING PHASES USING JAVA

The loading phases using Java can be broken down into three phases :

1. Loading
2. Linking
3. Initializing

The loading phase consists of locating the required class file and loading in the bytecode. The linking can be broken down into three mainly stages.

1. **Bytecode verification :** It should be well formed and well behaved.
2. **Class preparation :** This phase prepares the required data structures representing fields, methods, and implemented interfaces defined in each class.
3. **Resolving :** The class loader loads all the other classes referenced by a particular class.

Thank You

Gharu.anand@gmail.com