LINKER – AN INTRODUCTION

STEPS IN PROGRAM EXECUTION

Translator

Linking

Relocation

Loading

AGENDA

- Linking
- Linker
- Historical Perspective
- Linker vs Loader
- Linking Process
  - Two pass Linking
- Object Code Libraries
- Relocation and Code modification
- Linking an Example

LINKING

- Binding Abstract Names -> Concrete Names
- Eg:
  - Getline -> Abstract Name
  - 0x00100101 -> Concrete Name

Definition:

Linking is a process of binding an external reference to the correct link time address

What is a Linker?

- A System Software that Combines two or more separate object programs and supplies the information needed to allow references between them

Historical Perspective in Address Binding

- Linking in Low Level Programming
  - Hand written
  - Problem???
    - Hand Inspection
    - Address bound to the names too early
- Assemblers made it simple…
  - Programmer -> Computers
- After Advent of Operating System
  - Separation of Linkers and Loaders

- At the time of Programs became larger than available memory
  - Overlays, A technique that let programmers arrange for different parts of a program to share the same memory, with each overlay loaded on demand when another part of the program called into it.
  - Popular in 1960 but faded after the advent of Virtual Memory on PCs at 1990s

TYPES OF LINKING

- **Two Types**
  - Dynamic Linking
  - Static Linking

- **Static Linking:**
  - *Static linkers takes input a collection of relocatable object files and command line* arguments and generate as output a fully linked executable object file that can be loaded and run.

TYPES...

- Dynamic Linking:
  - The addresses of called procedures aren't bound until the first call.
  - Programs can bind to libraries as the programs are running, loading libraries in the middle of program execution.
  - This provides a powerful and high-performance way to extend the function of programs
  - MS Windows extensively uses DLL (Dynamic Linked Libraries)

Difference b/w Linker and Loader

- Linker is a program that takes one or more objects generated by a compiler and combines them into a single executable program.

- Loader is the part of an operating system that is responsible for loading programs from executables (i.e., executable files) into memory, preparing them for execution and then executing them.

- Linkers are the system softwares that are used to link the functions,resources to their respective references.
  EX-
  source code(.c) is compiled and converted into object code(.obj) in C.
  After this Linker comes into the act, linker resolve all the references in .obj file via linking them to their respectives codes and resources.In short linker performs the final step to convert .obj into executable or machine readable file(.exe).

- eg:

```
#include<stdio.h>
int main()
{
printf("ashwin");
return 0;
}
```

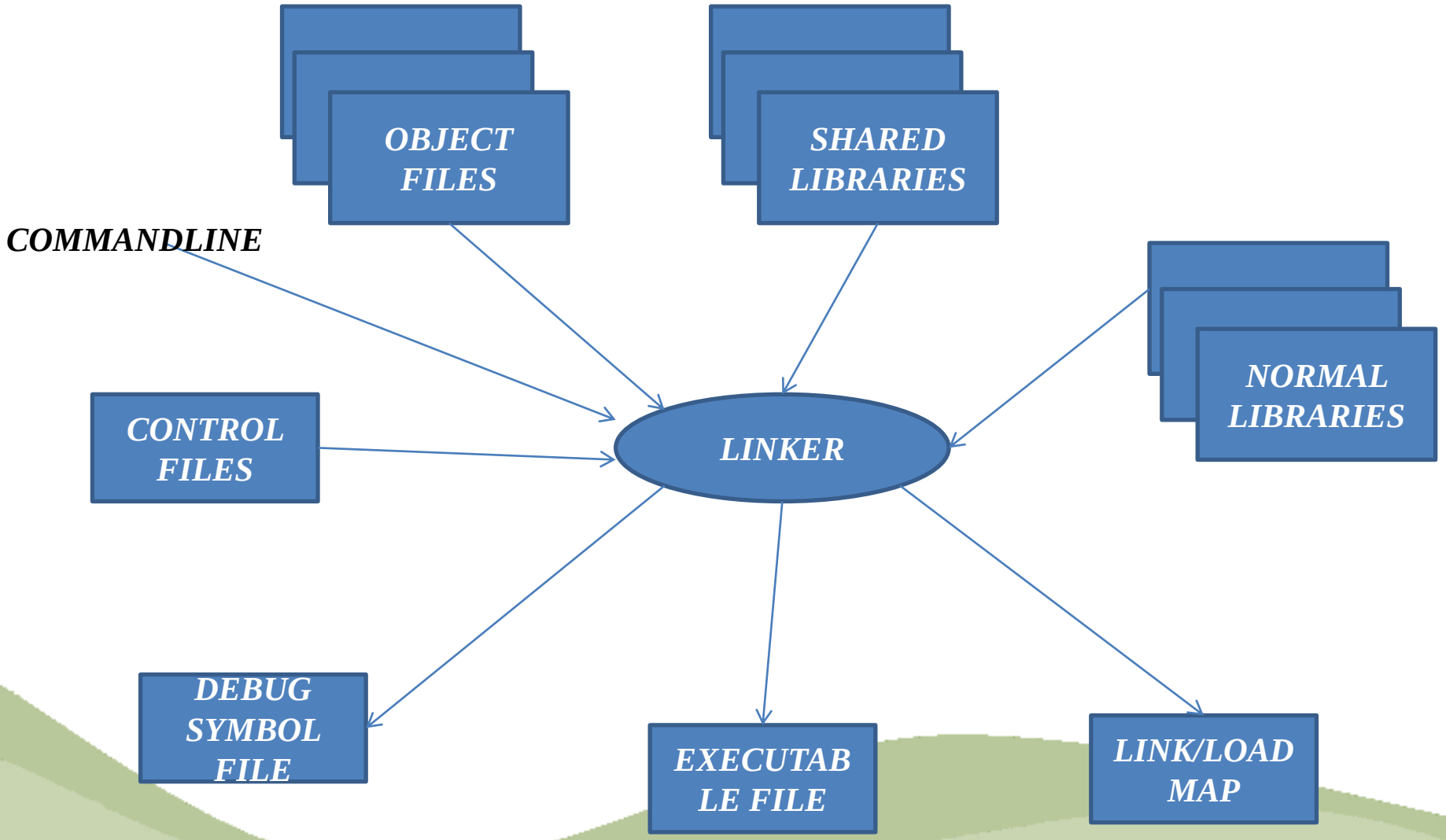
here,compiler first searches the declaration for "printf()" function,finds it in "#include<stdio.h>" and creates a (.obj) file successfully.

- A symbol table is created in (.obj) file which contains all the references to be resolved,linkers resolves them by providing respective code or resource,
  here code referred by "printf" also gets executed after successful creation of(.exe) file by linker.

- LOADERS:

  Loaders are the programs that are used to load the resources like files from secondary or main memory,i.e.
  Loader loads the referred resource or file after being Linked to referrer via a Linker during the execution of program.

LINKING PROCESS



OBJECT CODE LIBRARIES

- Object code library:

  set of object files.

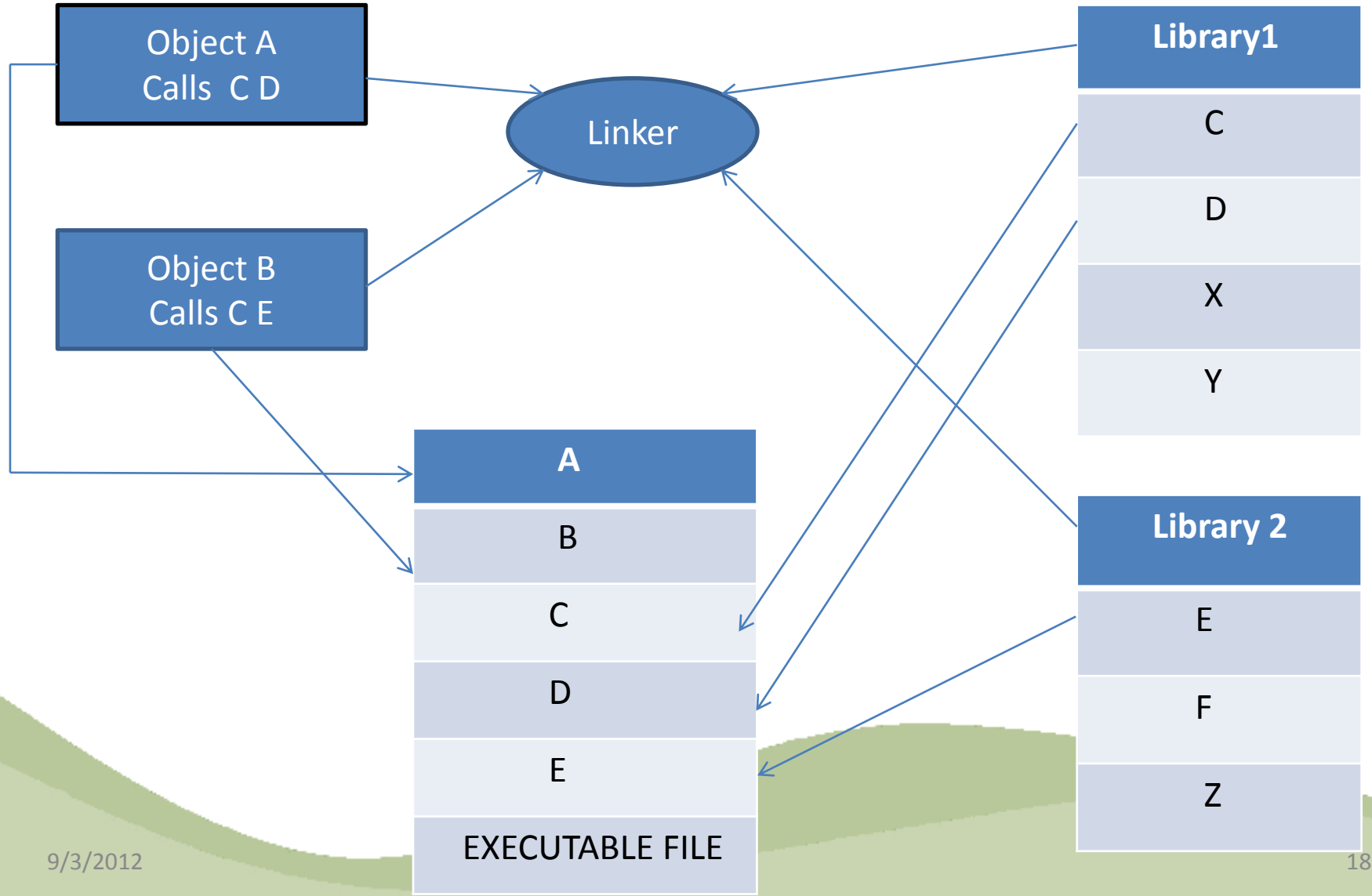
- Object File:-
  - header information (size, creation date, ...) ,object code
  - relocation information (list of places to relocate)
  - symbols: global symbols defined and symbols imported.
  - debugging information (source file, line numbers, local symbols, data structures)
- A library is little more than a set of object code files.

OBJECT CODE LIBRARIES

- ✓ All linkers support object code libraries in one form or another, with most also providing support for various kinds of shared libraries.

- ✓ After the linker processes all of the regular input files, if any imported names remain undefined
  - ➢ it runs through the library/libraries
  - ➢ links in any of the files in the library that export one or more undefined names.

OBJECT CODE LIBRARIES



CONTD...

- Shared libraries complicate this task a little by moving some of the work from link time to load time.

- The linker identifies the shared libraries that

resolve the undefined names in a linker run, but rather than linking anything into the program, the linker notes in the output file the names of the libraries in which the symbols were found, so that the shared library can be bound in when the program is loaded.

RELOCATION

- **Relocation** is the process of assigning load addresses to the various parts of the program, adjusting the code and data in the program to reflect the assigned addresses.

- **Relocation**, which modifies the object program so that it can be loaded at an address different from the location originally specified.

CONTD…

- The linker *relocates* these sections by
  - associating a memory location with each symbol definition
  - modifying all of the references to those symbols so that they point to this memory location.

- ✓ Relocation might happen more than once
  - linking several object files into a library
  - loading the library

RELOCATION AND CODE MODIFICATION

- The heart of a linker or loader's actions is relocation and code modification.
- When a compiler or assembler generates an object file, it generates the code using the unrelocated addresses of code and data defined within the file, and usually zeros for code and data defined elsewhere.
- As part of the linking process, the linker modifies the object code to reflect the actual addresses assigned.

EXAMPLE

- Code that moves the contents of variable a to variable b using the eax register.

```
mov    a,%eax
mov   %eax,b
```

- If a is defined in the same file at location 1234 hex and b is imported from somewhere else, the generated object code will be:

```
A1 34 12 00 00    mov a,%eax
A3 00 00 00 00    mov %eax,b
```

CONTD...

- The linker links this code so that the section in which a is located is relocated by hex 10000 bytes, and b turns out to be at hex 9A12.

- The linker modifies the code to be:

  Relocated code

  A1 34 12 01 00 mov a,%eax

  A3 12 9A 00 00 mov %eax,b

- That is, it adds 10000 to the address in the first instruction so now it refers to a's relocated address which is 11234, and it patches in the address for b. These adjustments affect instructions, but any pointers in the data part of an object file have to be adjusted as well.

Relocation and Code Modification

- Linker modifies code to reflect assigned addresses depends on hardware architecture
- On older computers with small address spaces and direct addressing there are only one or two address formats that a linker has to handle.

- Modern computers(RISC) require considerably more complex code modification because it constructs addresses by several instructions.
- No single instruction contains enough bits to hold a direct address, so the compiler and linker have to use complicated addressing tricks to handle data at arbitrary addresses.

Relocation and Code Modification

- In some cases, it's possible to calculate an address using two or three instructions, each of which contains part of the address, and use bit manipulation to combine the parts into a full address.
- In this case, the linker has to be prepared to modify each of the instructions, inserting some of the bits of the address into each instruction.

- In other cases, all of the addresses used by a routine or group of routines are placed in an array used as an "address pool", initialization code sets one of the machine registers to point to that array, and code loads pointers out of the address pool as needed using that register as a base register.

CONTD…

- The linker may have to create the array from all of the addresses used in a program, then modify instructions that so that they refer to the respective address pool entry.

- Code might be required to be position independent (PIC)
  - works regardless where library is loaded
  - same library used at different addresses by processes

- Linkers generally have to provide extra tricks to support that, separating out the parts of the program that can't be made position independent, and arranging for the two parts to communicate.

LINKER COMMAND LANGUAGES

- Every linker has some sort of command language to control the linking process.
- The linker needs the list of object files and libraries to link.
- In linkers that support multiple code and data segments, a linker command language can specify the order in which segments are to be linked.

- There are common techniques to pass commands to a linker,

Command Line:

- It is used to direct the linker to read commands from a file.

Embedded in object files:

- Linker commands to be embedded inside object files.
- This permits a compiler to pass any options needed to link an object file in the file itself.

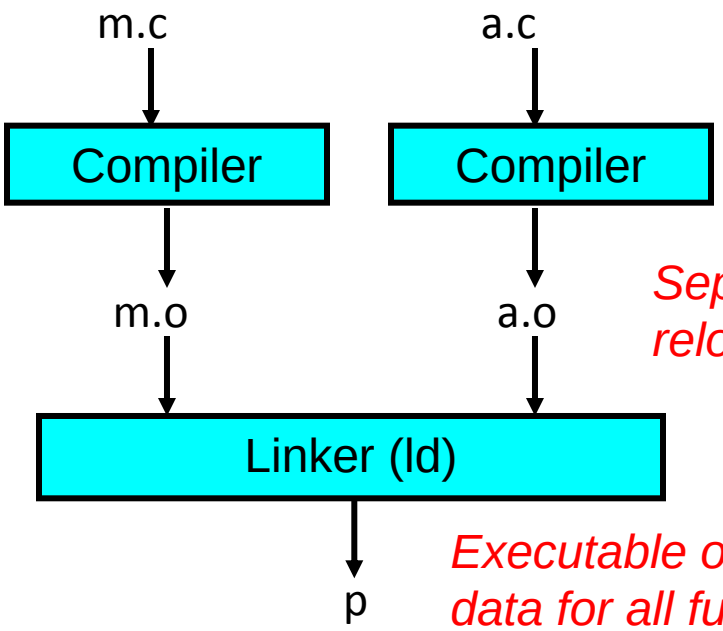
CONTD…

Separate configuration language:

- The linkers have a full fledged configuration language to control linking.
- The GNU linker, which can handle an enormous range of object file formats, machine architectures, and address space conventions

An Example from C Language

- A pair of C language source files, **m.c** with a main program that calls a routine named a, and **a.c** that contains the routine with a call to the library routines strlen and printf.
- When the source file compiles at the time the object file create.
- Each object file contain atleast one symbol table.



Separately compiled relocatable object files

Executable object file (contains code and data for all functions defined in m.c and a.c)

Source file m.c

```
extern void a(char *);
int main(int ac, char **av)
{
static char string[] =
    "Hello, world!\n";
a(string);
}
```

Source file a.c

```
#include <stdio.h>
#include <string.h>
void a(char *s)
{
printf("%d",strlen(s));
}
```

OBJECT CODE FOR M.O

Idx	Name	Size	VMA	LMA	File off	Algn
0 .	text	00000010	00000000	00000000	00000020	2**3
1 .	data	00000010	00000010	00000010	00000030	2**3

- 00000000 <_main>:
- 0: 55 pushl %ebp
- 1: 89 e5 movl %esp,%ebp
- 3: 68 10 00 00 00 pushl $0x10
- 4: 32 .data
- 8: e8 f3 ff ff ff call 0
- 9: DISP32 _a
- d: c9 leave
- e: c3 ret
- ...

- In that object file "text" segment containing the read only program code, and "data" segment containing the string.

There are two relocation entries,

- That marks the pushl instruction that puts the address of the string on the stack in preparation for the call to a.
- one that marks the call instruction that transfers control to a.

- The symbol table exports the definition of _main, imports _a.
- The object file of the subprogram a.c also contain data and text segments.

Two relocation entries,

- mark the calls to strlen and printf, and the symbol table exports _a and imports _strlen and _printf.

An example from C language

m.c

```
int e=7;

int main() {
  int r = a();
  exit(0);
}
```

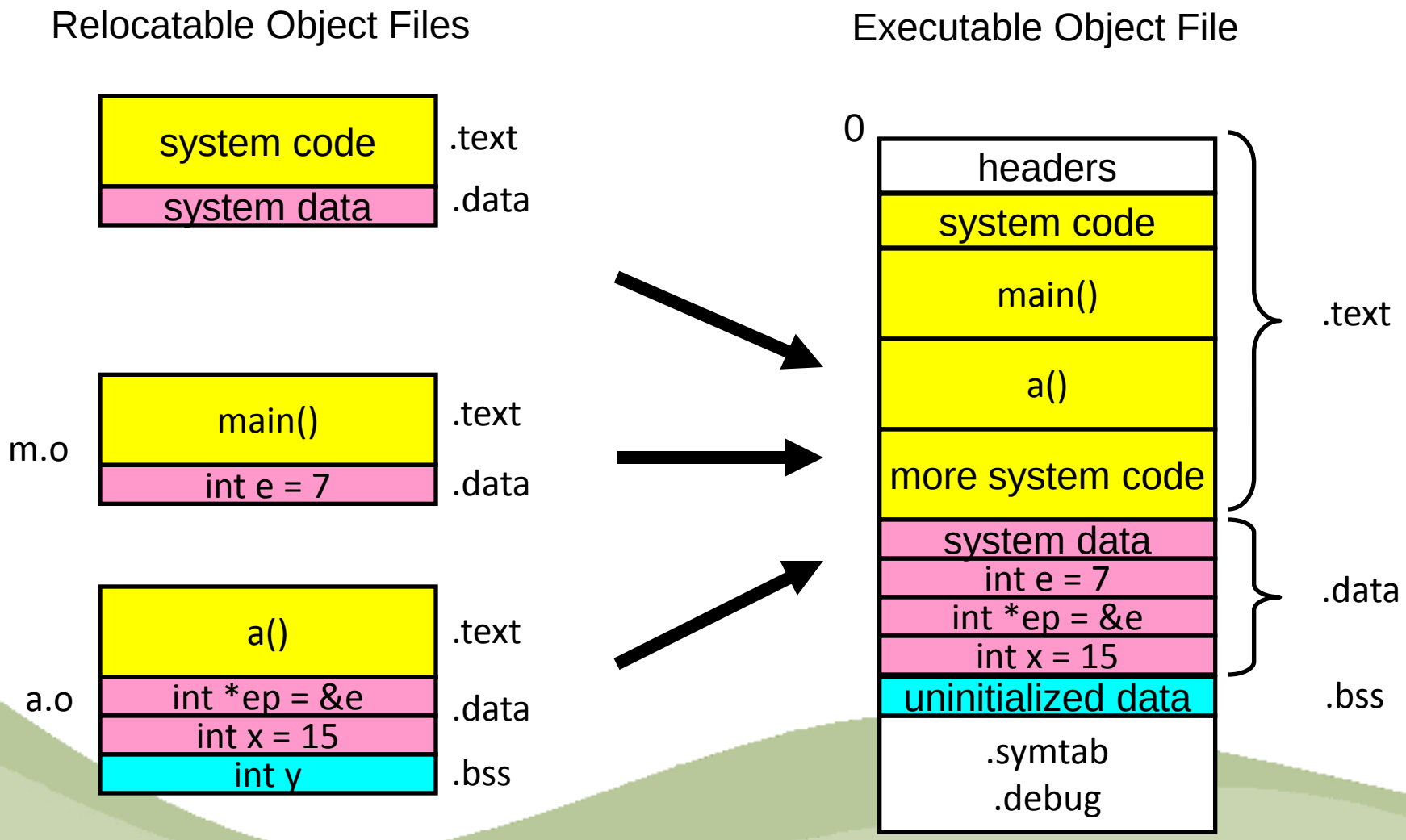
a.c

```
extern int e;

int *ep=&e;
int x=15;
int y;

int a() {
  return *ep+x+y;
}
```

Merging Relocatable Object Files into an Executable Object File



THANK YOU ….