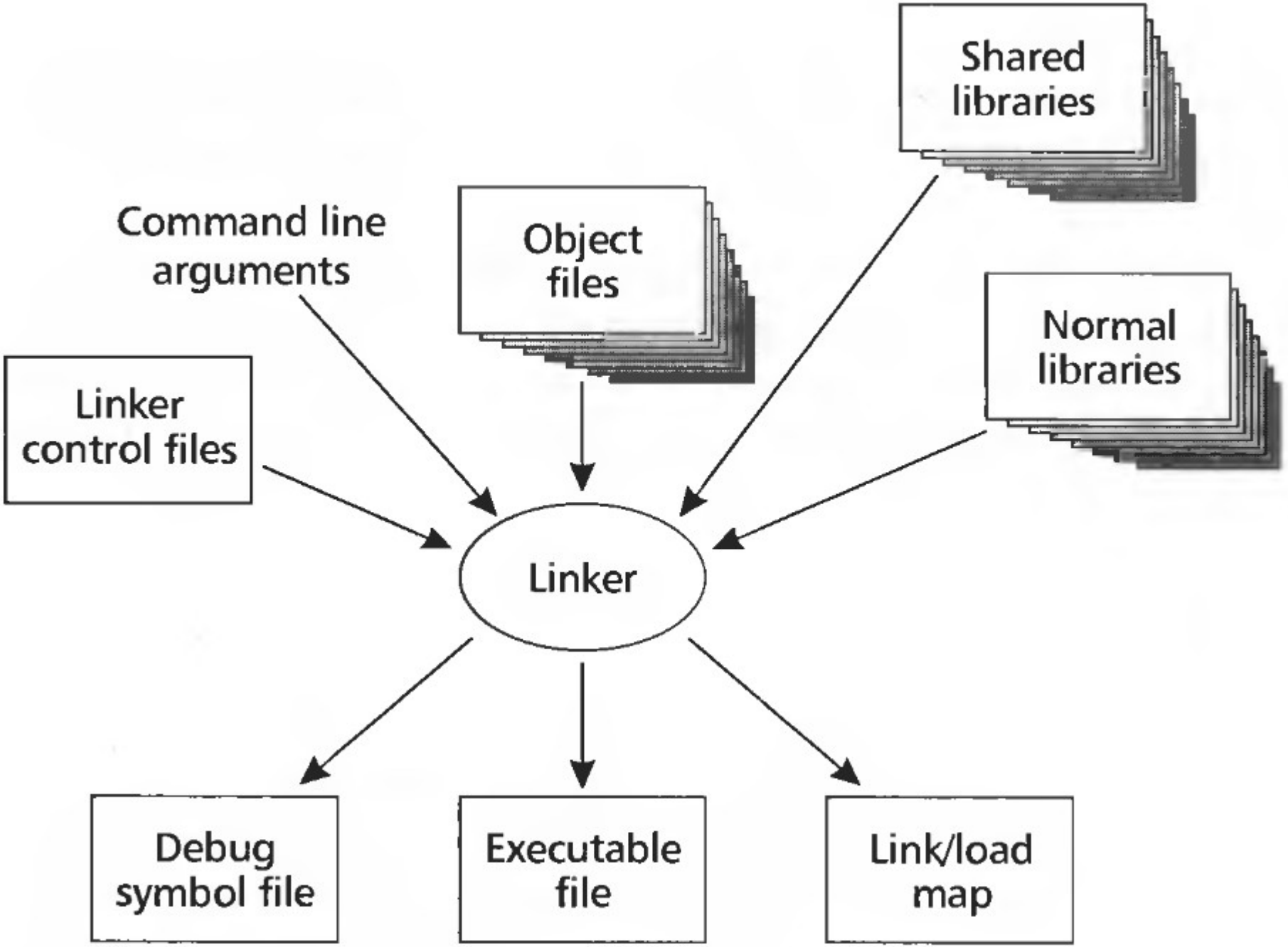
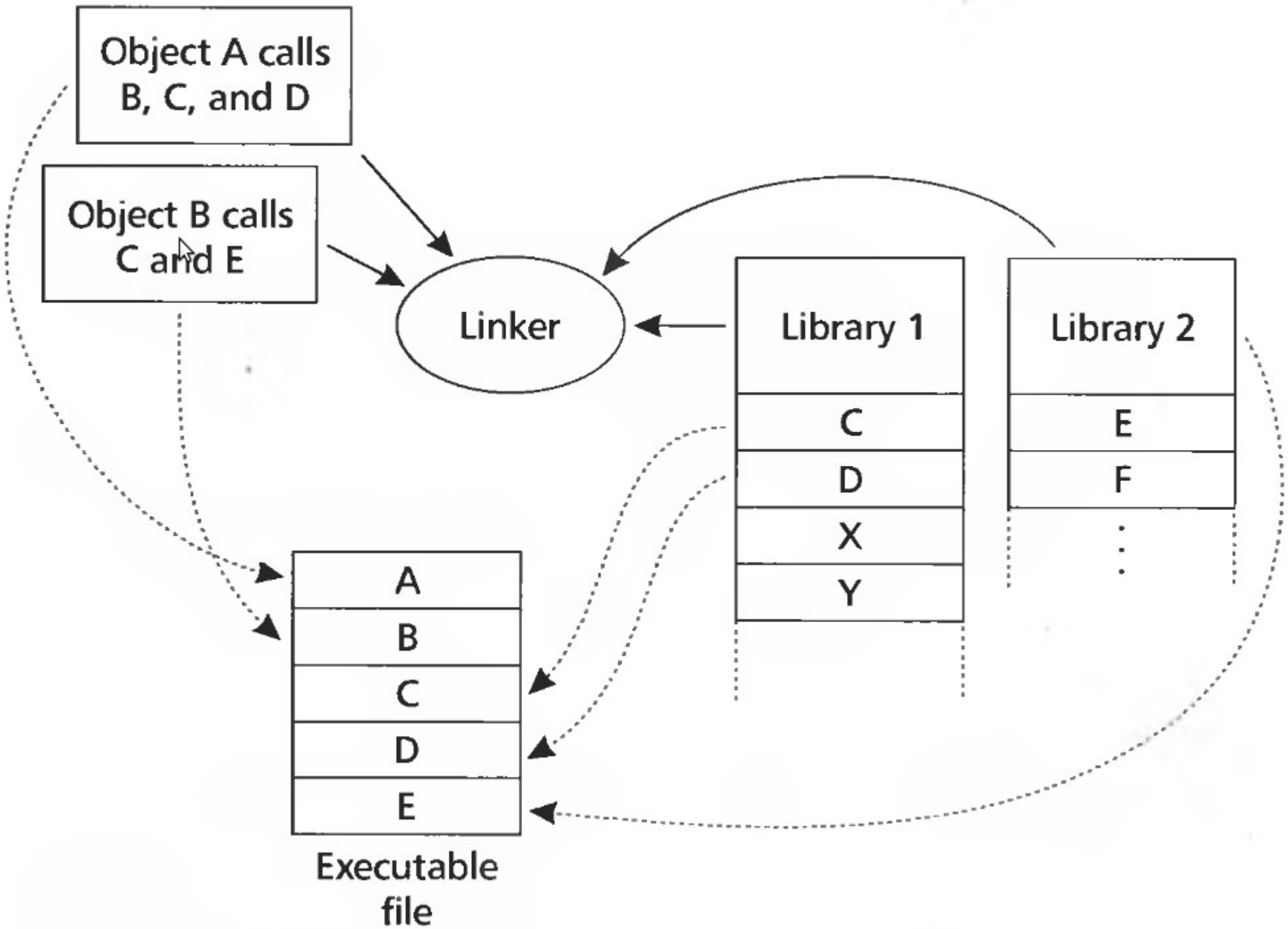
143A: Principles of Operating Systems

Lecture 9: Basic Architecture of a Program (part 2)

Anton Burtsev
October, 2017


Shared libraries
Command line arguments
Object files
Normal libraries
Linker control files
Linker
Debug symbol file
Executable file
Link/load map

- Input: object files (code modules)
- Each object file contains
  - A set of segments
    - Code
    - Data
  - A symbol table
    - Imported & exported symbols
- Output: executable file, library, etc.


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

Why linking?

Why linking?

- Modularity
  - Program can be written as a collection of modules
  - Can build libraries of common functions
- Efficiency
  - Code compilation
    - Change one source file, recompile it, and re-link the executable
  - Space efficiency
    - Share common code across executables
    - On disk and in memory

Two path process

- Path 1: scan input files
  - Identify boundaries of each segment
  - Collect all defined and undefined symbol information
  - Determine sizes and locations of each segment
- Path 2
  - Adjust memory addresses in code and data to reflect relocated segment addresses

Example

- Save a into b, e.g., b = a

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

- Generated code
  - a is defined in the same file at 0x1234, **b is imported**
  - Each instruction is 1 byte opcode + 4 bytes address

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

Example

- Save a into b, e.g., b = a

```
mov a, %eax
```

- 1 byte opcode

nerated code

a is defined in the same file at 0x1234, **b is imported**

Each instruction is 1 byte opcode + 4 bytes address

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

Example

- Save a into b, e.g., b = a

```
mov a, %eax
```

- 4 byte address

Generated code

- a is defined in the same file at 0x1234, **b is imported**
- Each instruction is 1 byte opcode + 4 bytes address

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

Example

- Save a into b, e.g., b = a

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

- Generated code
  - a is defined in the same file at 0x1234, **b is imported**
  - Each instruction is 1 byte opcode + 4 bytes address

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

- Assume that a is relocated by 0x10000 bytes, and b is found at 0x9a12

```
A1 34 12 01 00 mov a,%eax
A3 12 9A 00 00 mov %eax,b
```

Example

- Save a into b, e.g., b = a

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

- Generated code
  - a is defined in the same file at 0x1234, **b is imported**
  - Each instruction is 1 byte opcode + 4 bytes address

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

- Assume that a is relocated by 0x10000 bytes, and b is found at 0x9a12

```
A1 34 12 01 00 mov a,%eax
A3 12 9A 00 00 mov %eax,b
```

More realistic example

- 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 <unistd.h>
#include <string.h>
void a(char *s)
{
  write(1, s, strlen(s));
}
```

More realistic example

- 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 <unistd.h>
#include <string.h>
void a(char *s)
{
  write(1, s, strlen(s));
}
```

More realistic example

- 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 <unistd.h>
#include <string.h>
void a(char *s)
{
  write(1, s, strlen(s));
}
```

More realistic example

```
Sections:
 Idx Name Size     VMA      LMA      File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

More realistic example

- Two sections:
  - Text (0x10 – 16 bytes)
  - Data (16 bytes)

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

More realistic example

- Two sections:
  - Text starts at 0x0
  - Data starts at 0x10

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

More realistic example

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .t                000000 00000000 00000020 2**3
  1 .d                000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

- Code starts at 0x0

More realistic example

```
Sections:
 Idx Name Size     VMA      LMA      File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

More realistic example

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
00000000 <_main>:
  0: 55                pushl %ebp
  1: 89 e5             movl %esp,%ebp
  3: 68 10 00 00 00 pushl $0x10 # push string on the stack
    4: 32 .data
  8: e8 f3 ff ff ff call 0
    9: DISP32 _a
  d: c9                leave
  e: c3                ret
  ...
```

- First relocation entry
  - Marks pushl 0x10
  - 0x10 is beginning of the data section
  - and address of the string

More realistic example

- 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 <unistd.h>
#include <string.h>
void a(char *s)
{
  write(1, s, strlen(s));
}
```

More realistic example

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
00000000 <_main>:
  0: 55                pushl %ebp
  1: 89 e5             movl %esp,%ebp
  3: 68 10 00 00 00 pushl $0x10 # push string on the stack
    4: 32 .data
  8: e8 f3 ff ff ff call 0
    9: DISP32 _a
  d: c9                leave
  e: c3                ret
  ...
```

- First relocation entry
  - Marks pushl 0x10
  - 0x10 is beginning of the data section
  - and address of the string

More realistic example

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000010 00000000 00000000 00000020 2**3
  1 .data 00000010 00000010 00000010 00000030 2**3
Disassembly of section .text:
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
  ...
```

- Second relocation entry
  - Marks call
  - 0x0 – address is unknown

More realistic example

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 0000001c 00000000 00000000 00000020 2**2
    CONTENTS, ALLOC, LOAD, RELOC, CODE
  1 .data 00000000 0000001c 0000001c 0000003c 2**2
    CONTENTS, ALLOC, LOAD, DATA
Disassembly of section .text:
  00000000 <_a>:
  0: 55                 pushl %ebp
  1: 89 e5              movl %esp,%ebp
  3: 53                 pushl %ebx
  4: 8b 5d 08           movl 0x8(%ebp),%ebx
  7: 53                 pushl %ebx
  8: e8 f3 ff ff ff     call 0
    9: DISP32 _strlen
  d: 50                 pushl %eax
  e: 53                 pushl %ebx
  f: 6a 01              pushl $0x1
  11: e8 ea ff ff ff    call 0
    12: DISP32 _write
  16: 8d 65 fc          leal -4(%ebp),%esp
  19: 5b                popl %ebx
  1a: c9                leave
  1b: c3                ret
```

- Two sections:
  - Text (0 bytes)
  - Data (28 bytes)

More realistic example

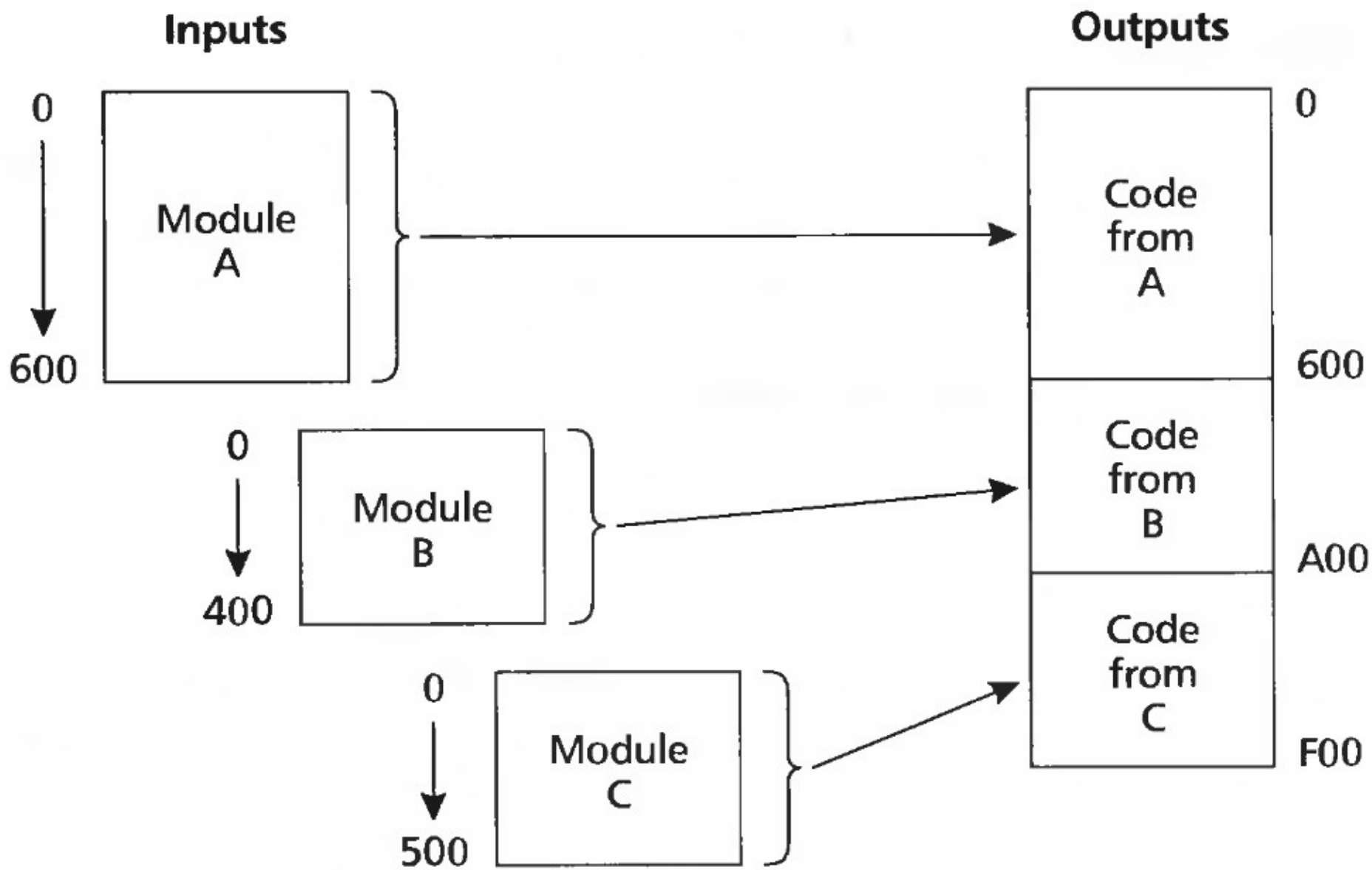
```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 0000001c 00000000 00000000 00000020 2**2
    CONTENTS, ALLOC, LOAD, RELOC, CODE
  1 .data 00000000 0000001c 0000001c 0000003c 2**2
    CONTENTS, ALLOC, LOAD, DATA
Disassembly of section .text:
  00000000 <_a>:
  0: 55                  pushl %ebp
  1: 89 e5               movl %esp,%ebp
  3: 53                  pushl %ebx
  4: 8b 5d 08            movl 0x8(%ebp),%ebx
  7: 53                  pushl %ebx
  8: e8 f3 ff ff ff      call 0
    9: DISP32 _strlen
  d: 50                  pushl %eax
  e: 53                  pushl %ebx
  f: 6a 01               pushl $0x1
  11: e8 ea ff ff ff     call 0
    12: DISP32 _write
  16: 8d 65 fc           leal -4(%ebp),%esp
  19: 5b                 popl %ebx
  1a: c9                 leave
  1b: c3                 ret
```

- Two relocation entries:
  - strlen()
  - write()

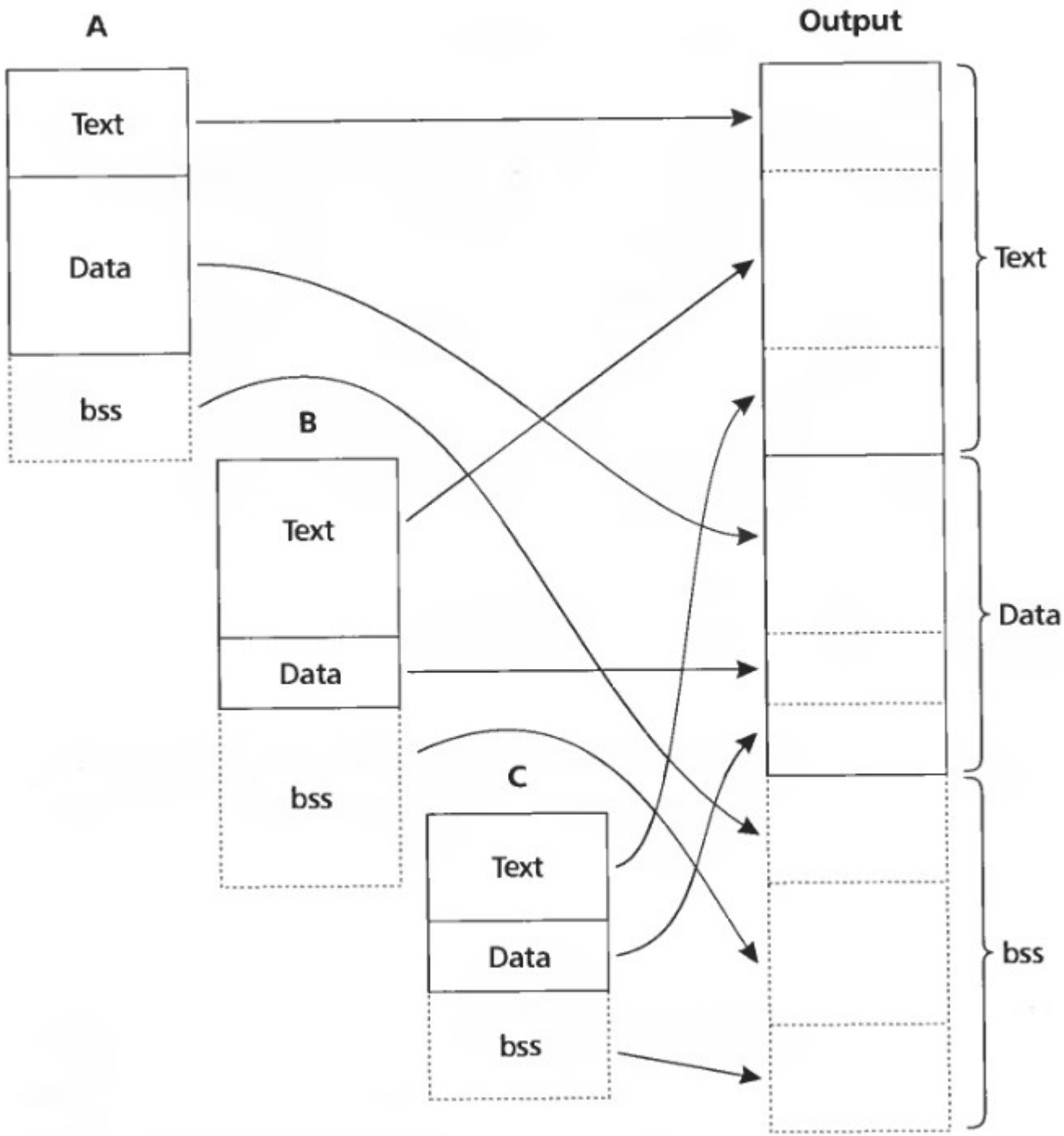
Producing an executable

- Combine corresponding segments from each object file
  - Combined text segment
  - Combined data segment
- Pad each segment to 4KB to match the page size

Multiple object files



Merging segments



```
Sections:
 Idx Name Size     VMA      LMA      File off Algn
  0 .text 00000fe0 00001020 00001020 00000020 2**3
  1 .data 00001000 00002000 00002000 00001000 2**3
  2 .bss  00000000 00003000 00003000 00000000 2**3
Disassembly of section .text:
00001020 <start-c>:
  ...
  1092: e8 0d 00 00 00 call 10a4 <_main>
  ...
000010a4 <_main>:

  10a7: 68 24 20 00 00 pushl $0x2024
  10ac: e8 03 00 00 00 call 10b4 <_a>
  ...
000010b4 <_a>:

  10bc: e8 37 00 00 00 call 10f8 <_strlen>
  ...
  10c3: 6a 01 pushl $0x1
  10c5: e8 a2 00 00 00 call 116c <_write>
  ...
000010f8 <_strlen>:
  ...
0000116c <_write>:
  ...
```

Linked executable

```
Sections:
 Idx Name Size      VMA       LMA       File off Algn
  0 .text 00000fe0 00001020 00001020 00000020 2**3
  1 .data 00001000 00002000 00002000 00001000 2**3
  2 .bss  00000000 00003000 00003000 00000000 2**3
Disassembly of section .text:
00001020 <start-c>:
  ...
  1092: e8 0d 00 00 00 call 10a4 <_main>
  ...
000010a4 <_main>:

  10a7: 68 24 20 00 00 pushl $0x2024
  10ac: e8 03 00 00 00 call 10b4 <_a>
  ...
000010b4 <_a>:

  10bc: e8 37 00 00 00 call 10f8 <_strlen>
  ...
  10c3: 6a 01 pushl $0x1
  10c5: e8 a2 00 00 00 call 116c <_write>
  ...
000010f8 <_strlen>:
  ...
0000116c <_write>:
  ...
```

- Relative to EIP address
  - Hence 3

Linked executable

Tasks involved

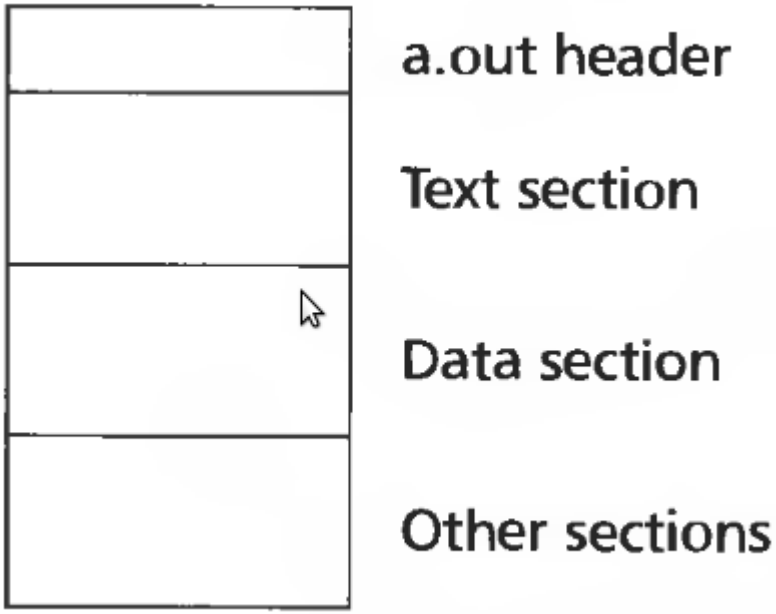
- Program loading
  - Copy a program from disk to memory so it is ready to run
    - Allocation of memory
    - Setting protection bits (e.g. read only)
- Relocation
  - Assign load address to each object file
  - Adjust the code
- Symbol resolution
  - Resolve symbols imported from other object files

Object files

Object files

- Conceptually: five kinds of information
  - Header: code size, name of the source file, creation date
  - Object code: binary instruction and data generated by the compiler
  - Relocation information: list of places in the object code that need to be patched
  - Symbols: global symbols defined by this module
    - Symbols to be imported from other modules
  - Debugging information: source file and file number information, local symbols, data structure description

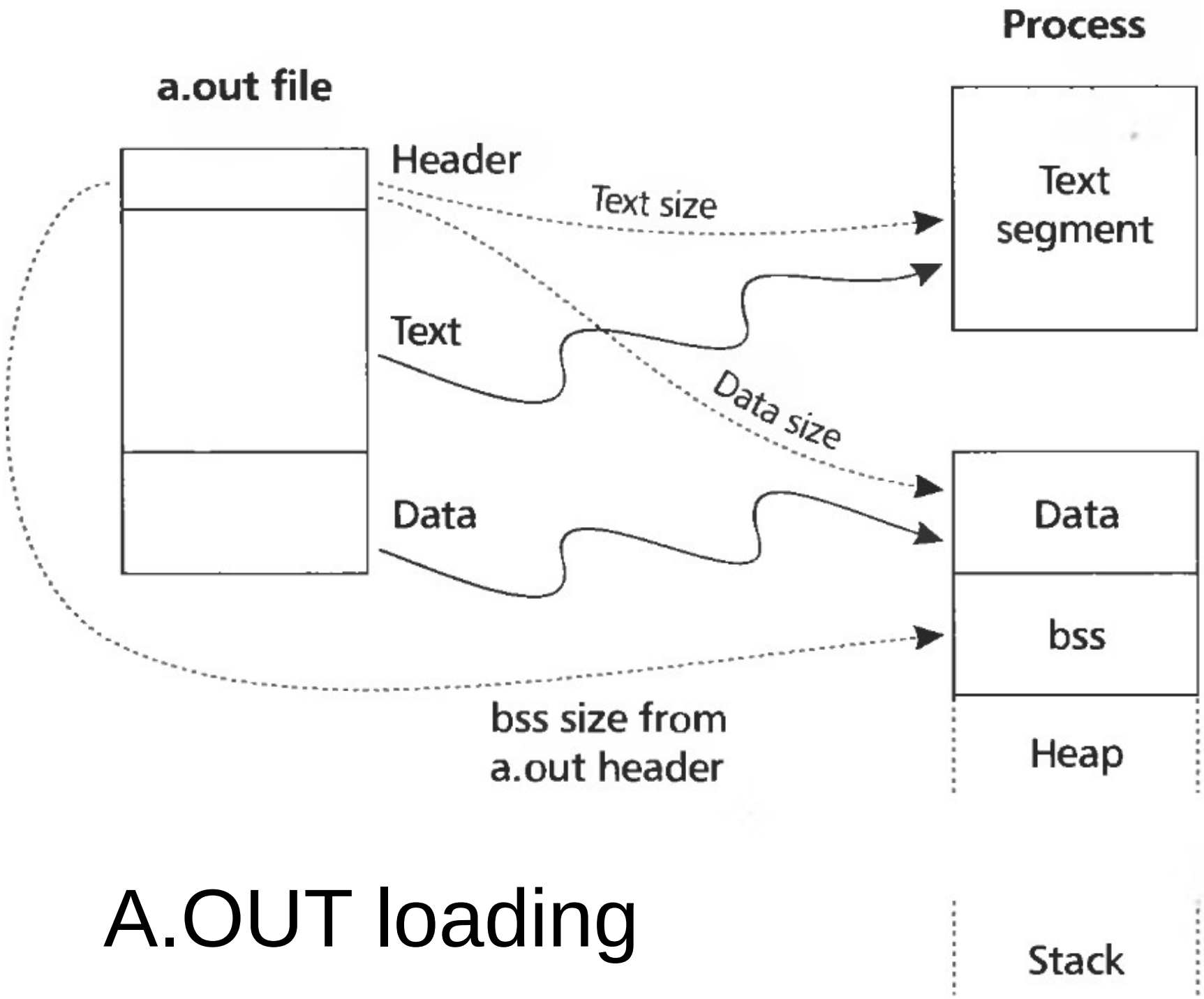
Example: UNIX A.OUT



- Small header
- Text section
  - Executable code
- Data section
  - Initial values for static data

- A.OUT header

```
int a_magic;  // magic number
int a_text;   // text segment size
int a_data;   // initialized data size
int a_bss;    // uninitialized data size
int a_syms;   // symbol table size
int a_entry;  // entry point
int a_trsize; // text relocation size
int a_drsize; // data relocation size
```



A.OUT loading

A.OUT loading

- Read the header to get segment sizes
- Check if there is a shareable code segment for this file
  - If not, create one,
  - Map into the address space,
  - Read segment from a file into the address space
- Create a private data segment
  - Large enough for data and BSS
  - Read data segment, zero out the BSS segment
- Create and map stack segment
  - Place arguments from the command line on the stack
- Jump to the entry point

Types of object files

- Relocatable object files (.o)
- Static libraries (.a)
- Shared libraries (.so)
- Executable files

- We looked at A.OUT, but Unix has a general format capable to hold any of these files

ELF

Elf header

- Magic number, type (.o, exec, .so), machine, byte ordering, etc.

Segment header table

- Page size, virtual addresses memory segments (sections), segment sizes.

`.text` section

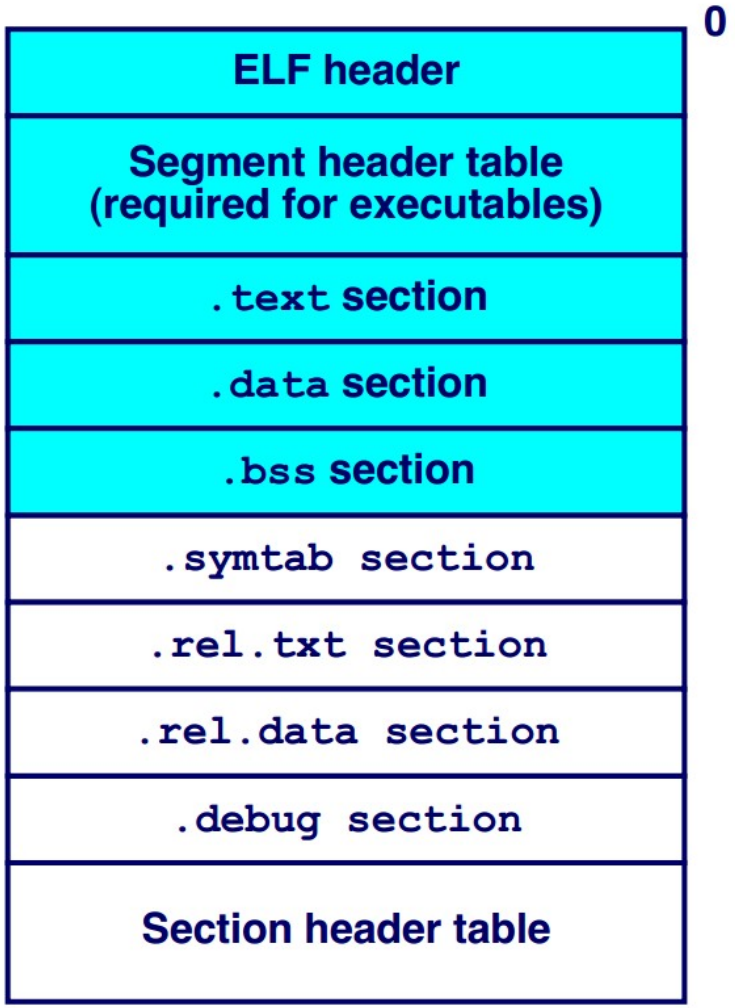
- Code

`.data` section

- Initialized global variables

`.bss` section

- Uninitialized global variables
- "Block Started by Symbol"
- "Better Save Space"
- Has section header but occupies no space



ELF (continued)

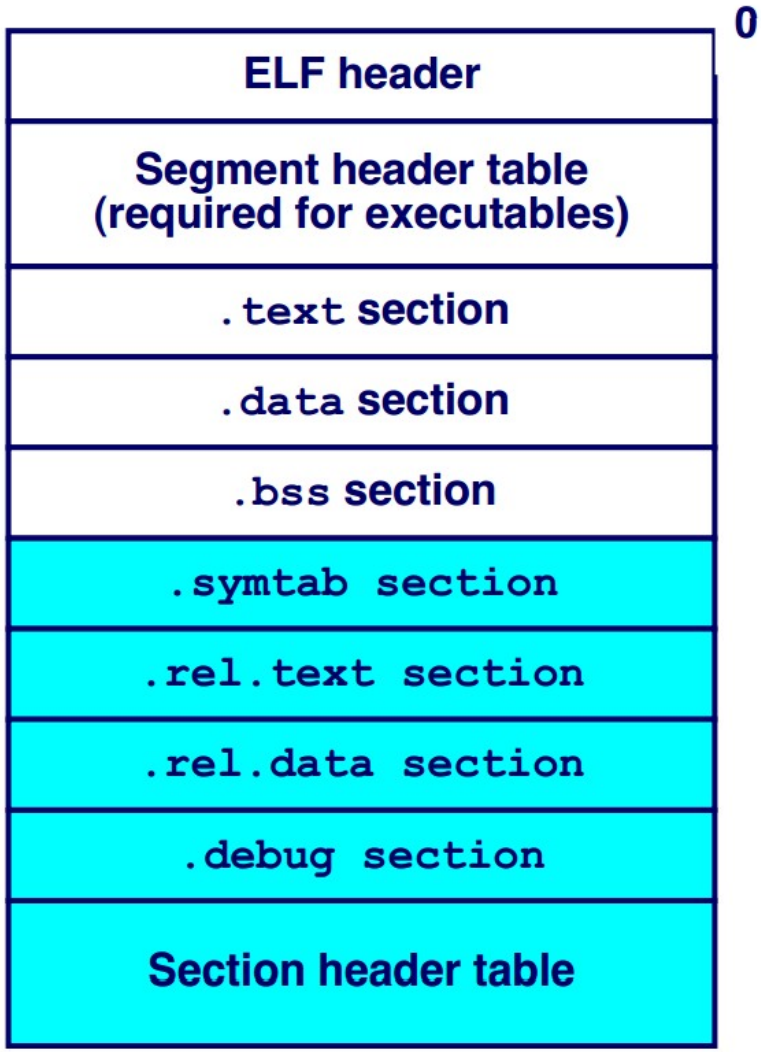
`.symtab` section
- Symbol table
- Procedure and static variable names
- Section names and locations

`.rel.text` section
- Relocation info for `.text` section
- Addresses of instructions that will need to be modified in the executable
- Instructions for modifying.

`.rel.data` section
- Relocation info for `.data` section
- Addresses of pointer data that will need to be modified in the merged executable

`.debug` section
- Info for symbolic debugging (`gcc -g`)

Section header table
- Offsets and sizes of each section

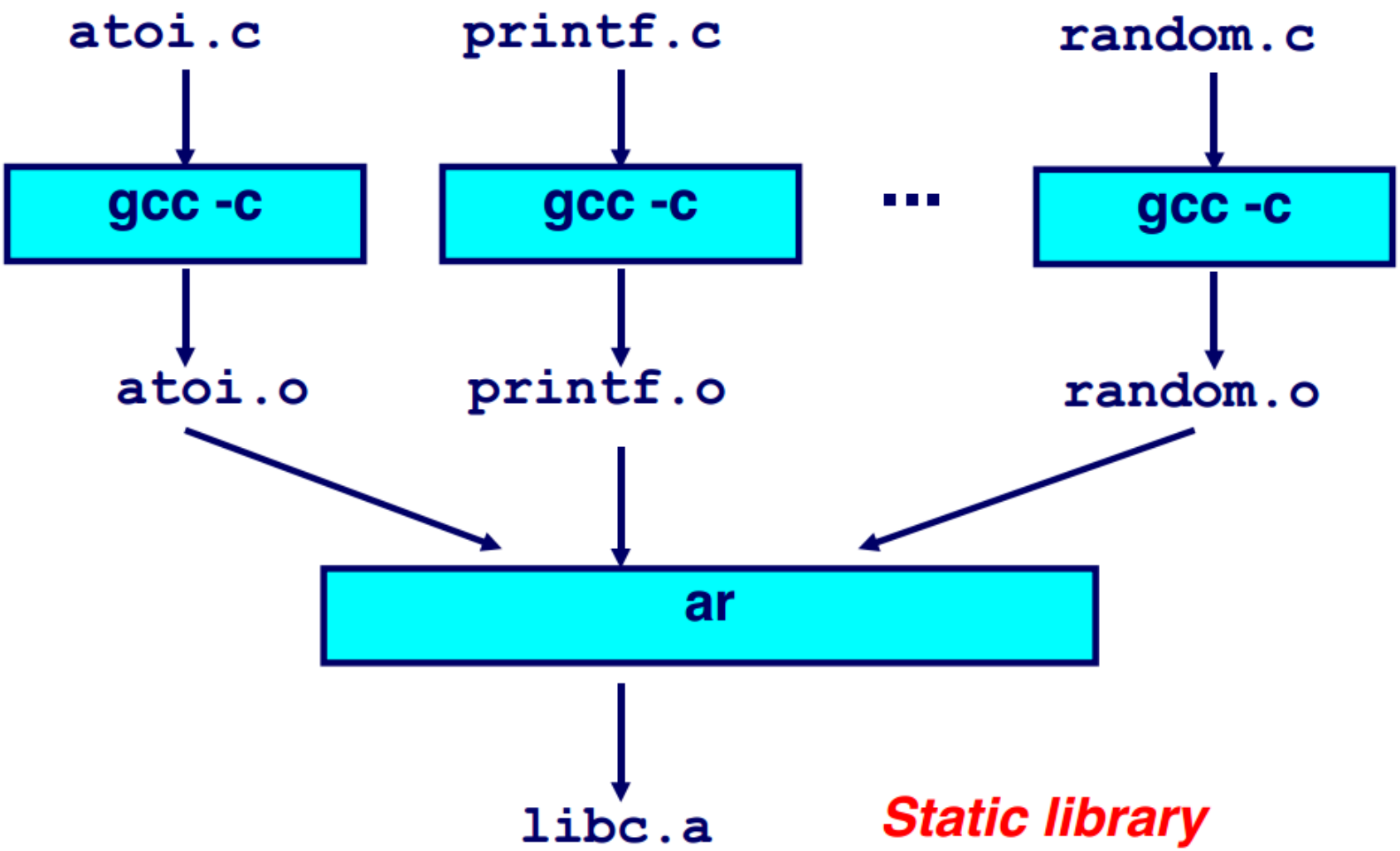


Static libraries

Libraries

- Conceptually a library is
  - Collection of object files
- UNIX uses an archive format
    - Remember the **ar** tool
  - Can support collections of any objects
  - Rarely used for anything instead of libraries

Creating a static library



Searching libraries

- First linker path needs resolve symbol names into function locations
- To improve the search library formats add a directory
  - Map names to member positions

Thank you!