238P: Operating Systems

Lecture 7: Basic Architecture of a Program

Anton Burtsev
January, 2018

What is a program?

- What parts do we need to run code?

Parts needed to run a program

- Code itself
    - By convention it's called text
- Stack
    - To call functions
- Space for variables
    - Ok... this is a bit tricky
    - 3 types
        - Global, local, and heap

Space for variables (3 types)

- Global variables

```
#include <stdio.h>

char hello[] = "Hello";
int main(int ac, char **av)
{
    static char world[] = "world!";
    printf("%s %s\n", hello, world);
    return 0;
}
```

- Allocated in the program text
  - They are split in initialized (non-zero), and non-initialized (zero)

Space for variables (3 types)

- Local variables

```
#include <stdio.h>

char hello[] = "Hello";
int main(int ac, char **av)
{
    //static char world[] = "world!";
    char world[] = "world!";
    printf("%s %s\n", hello, world);
    return 0;
}
```

- Allocated on the stack
  - Remember calling conventions?

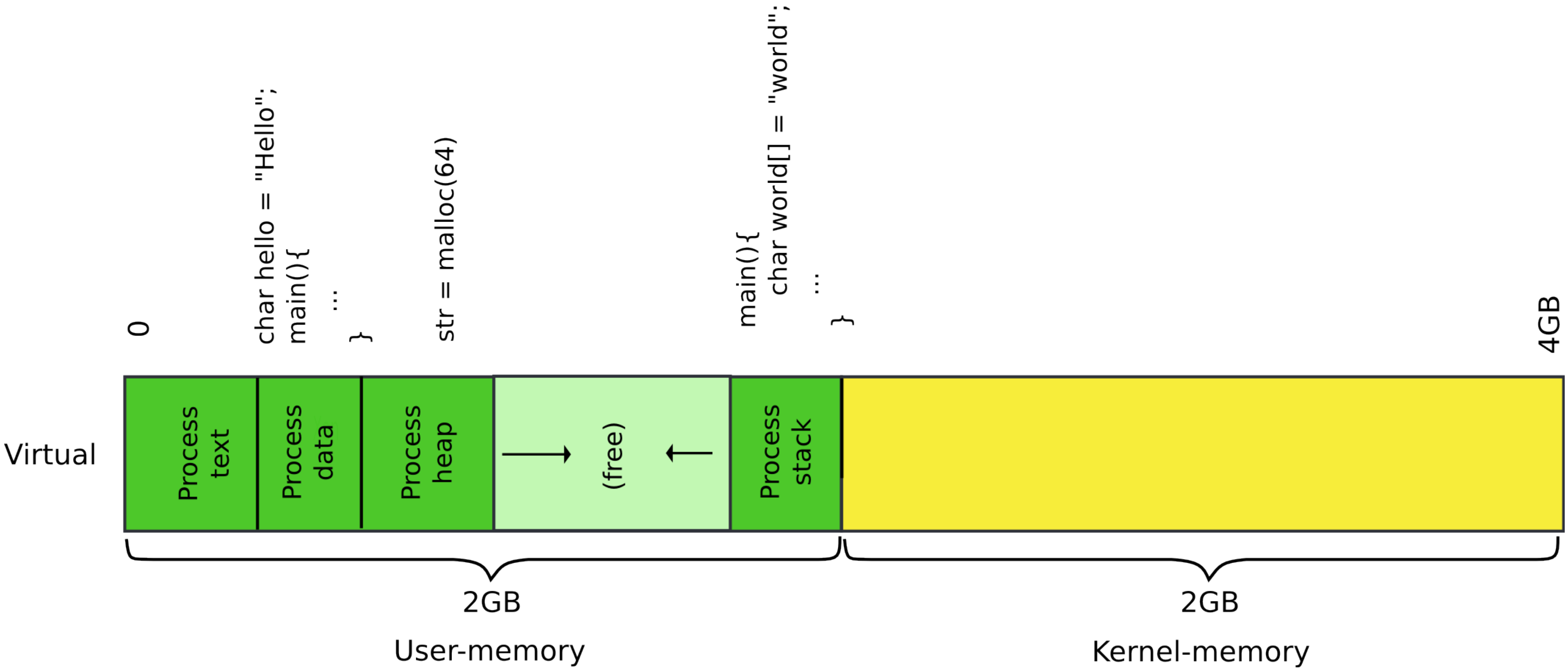
Space for variables (3 types)

- Local variables

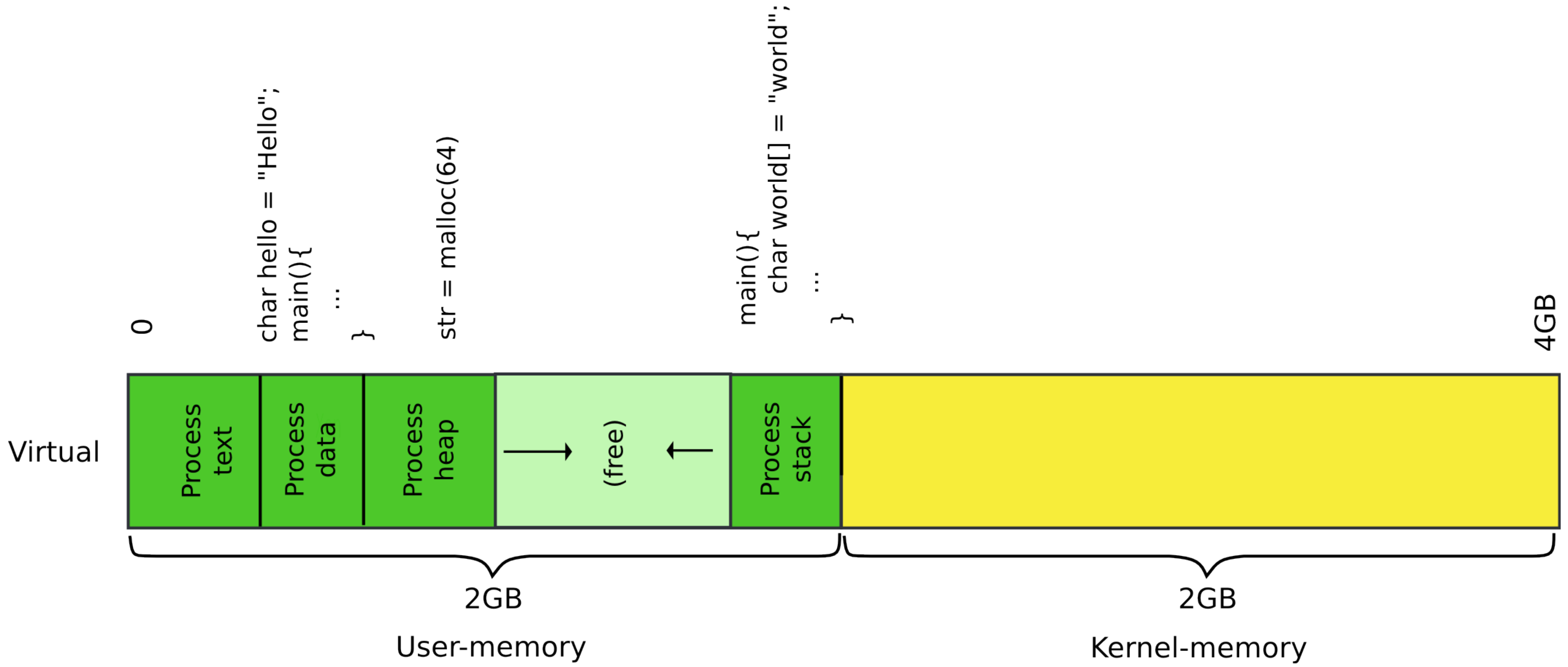
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char hello[] = "Hello";
int main(int ac, char **av)
{
    char world[] = "world!";
    char *str = malloc(64);
    memcpy(str, "beautiful", 64);
    printf("%s %s %s\n", hello, str, world);
    return 0;
}
```

- Allocated on the heap
  - Special area of memory provided by the OS from where malloc() can allocate memory

Memory layout of a process



Where do these areas come from?

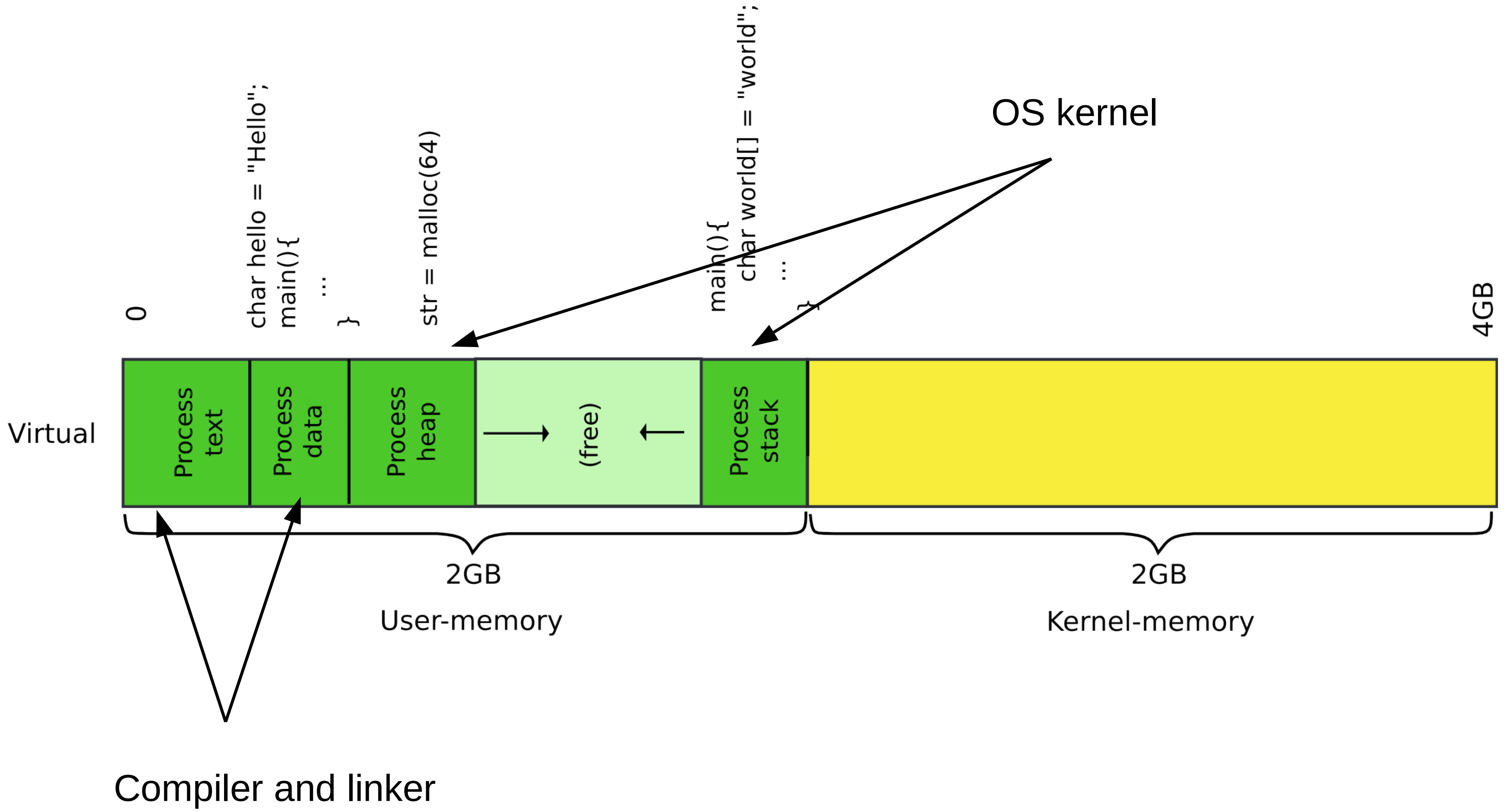


```
char hello = "Hello";
main(){
   ...
}
```

```
str = malloc(64)
```

```
main(){
  char world[] = "world";
    ...
}
```

Memory layout of a process



OS kernel

Compiler and linker

Example program

- Compute 5 + 6

```
#include <stdio.h>

int main(int ac, char **av)
{
    int a = 5, b = 6;
    return a + b;
}
```

- We build it like
  - I'm on 64 bit system, but want 32bit code, hence -m32

```
gcc -m32 hello-int.c
```

objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp      # Maintain the stack frame
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub     $0x10,%esp     # Allocate space for a and b
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp) # Initialize a = 5
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp) # Initialize b = 6
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov     -0x4(%ebp),%eax # Move b into %eax
 8048404:       8b 55 f8                mov     -0x8(%ebp),%edx # Move a into %edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

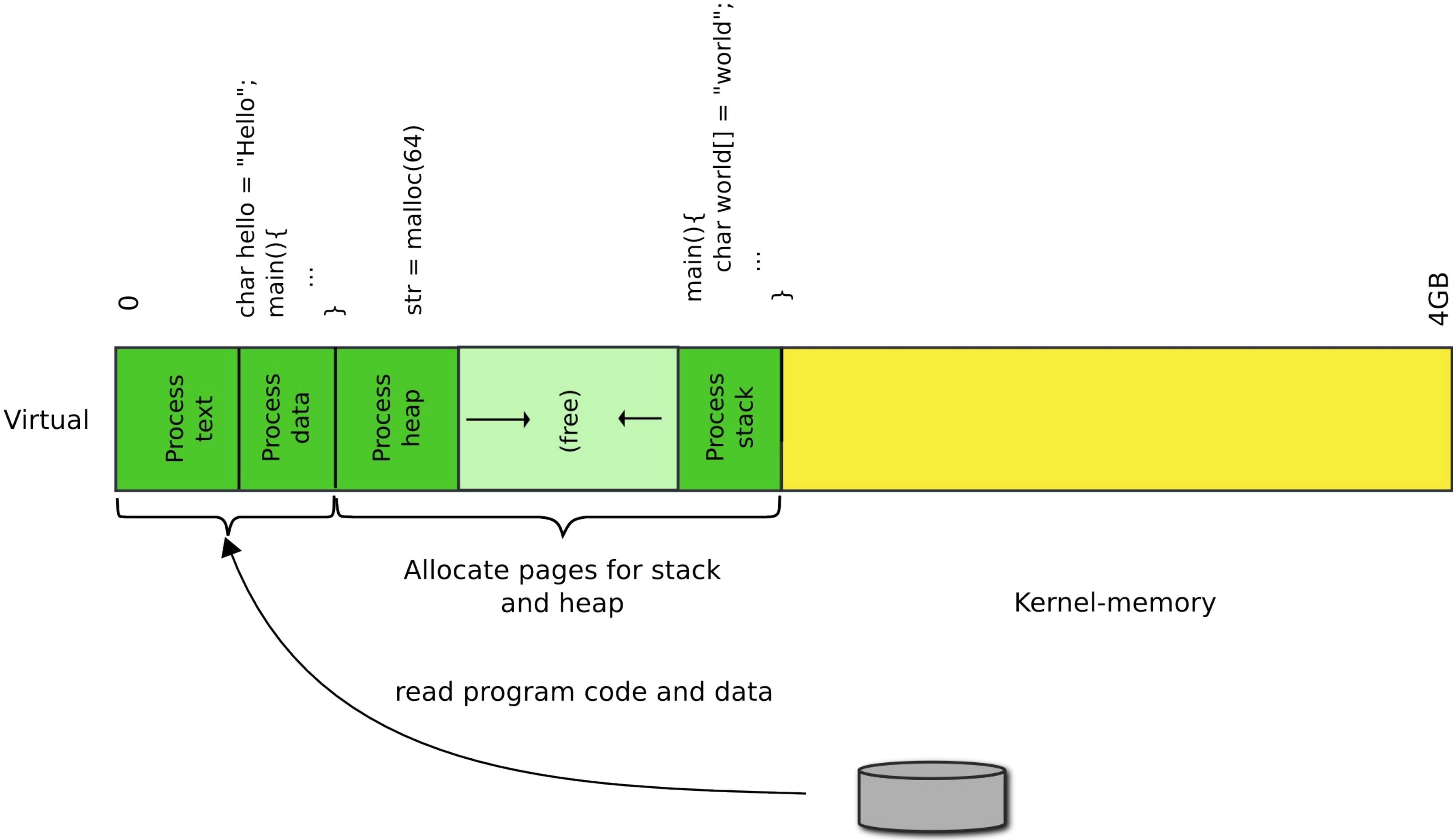
objdump -sd a.out

```
a.out:     file format elf32-i386

Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add     %edx,%eax      # a + b
 8048409:       c9                      leave
 804840a:       c3                      ret
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

objdump -sd a.out

```
a.out:     file format elf32-i386
Contents of section .text:
 80483e0 d0c9e979 ffffff90 e973ffff ff5589e5  ...y.....s...U..
 80483f0 83ec10c7 45f80500 0000c745 fc060000  ....E......E....
 8048400 008b45fc 8b55f801 d0c9c366 90669090  ..E..U.....f.f..
 8048410 555731ff 5653e805 ffffff81 c3e51b00  UW1.VS..........
 8048420 0083ec1c 8b6c2430 8db30cff ffffe861  .....l$0.......a
 8048430 feffff8d 8308ffff ff29c6c1 fe0285f6  .........)......
Contents of section .rodata:
 8048498 03000000 01000200                    ........
Contents of section .data:
 804a014 00000000 00000000                    ........
Disassembly of section .text:
…
080483ed <main>:
 80483ed:       55                      push   %ebp
 80483ee:       89 e5                   mov    %esp,%ebp
 80483f0:       83 ec 10                sub    $0x10,%esp
 80483f3:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%ebp)
 80483fa:       c7 45 fc 06 00 00 00    movl   $0x6,-0x4(%ebp)
 8048401:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048404:       8b 55 f8                mov    -0x8(%ebp),%edx
 8048407:       01 d0                   add    %edx,%eax
 8048409:       c9                      leave                 # Pop the frame ESP = EBP
 804840a:       c3                      ret                   # return
 804840b:       66 90                   xchg   %ax,%ax
 804840d:       66 90                   xchg   %ax,%ax
 804840f:       90                      nop
```

Load program in memory

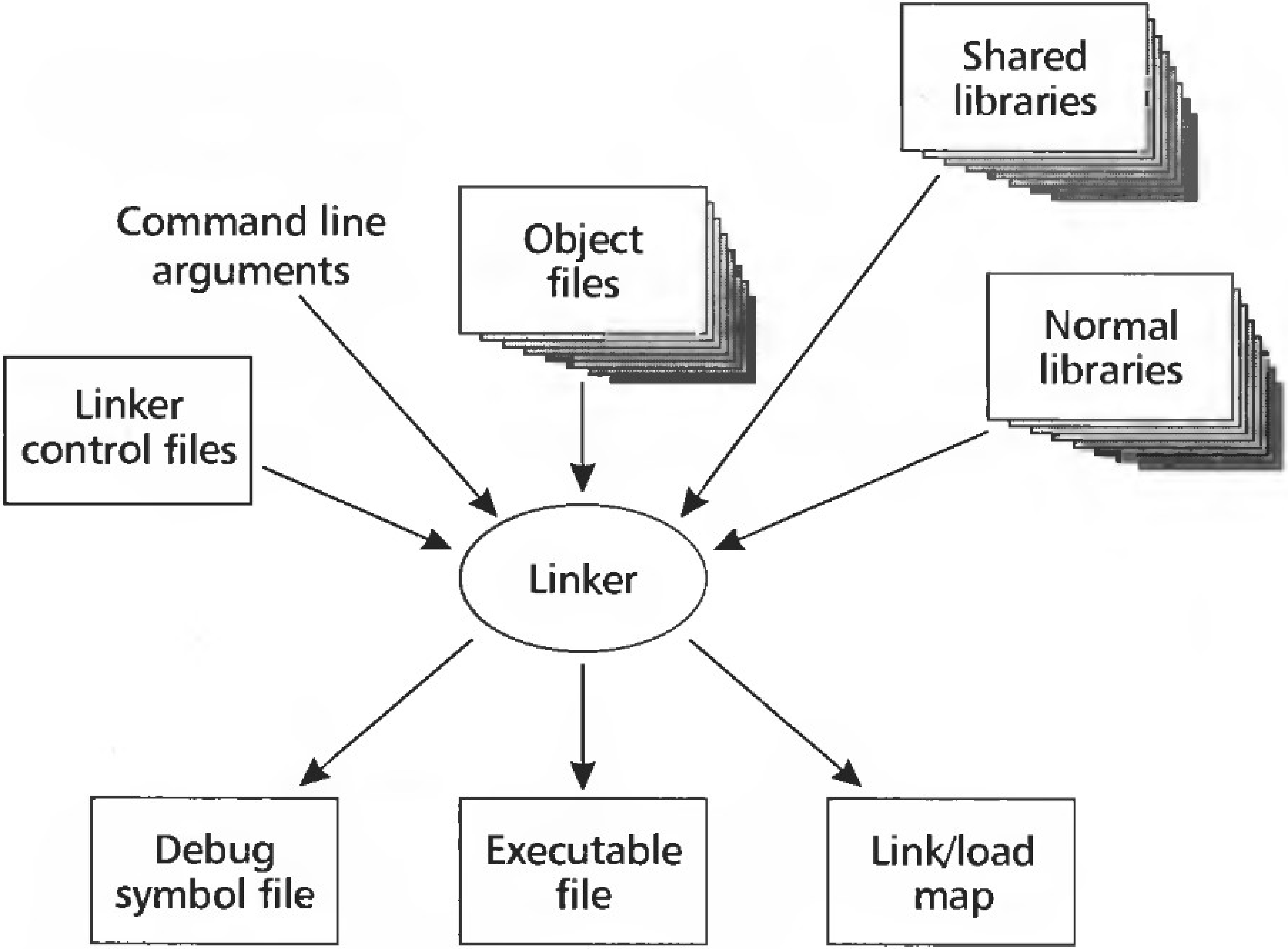


We however build programs from multiple files

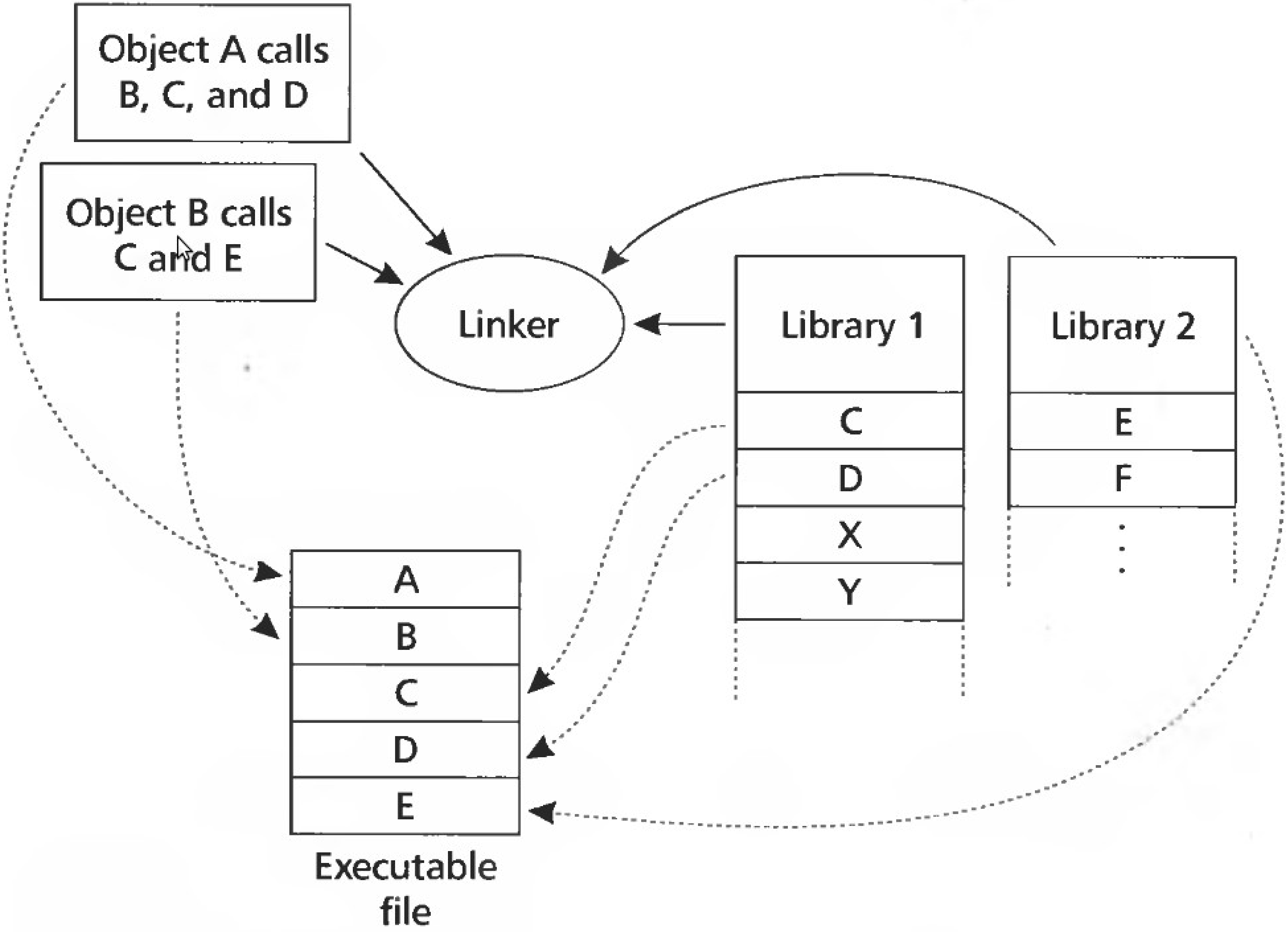
```
bootblock: bootasm.S bootmain.c
        $(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c bootmain.c
        $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c bootasm.S
        $(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
        $(OBJDUMP) -S bootblock.o > bootblock.asm
        $(OBJCOPY) -S -O binary -j .text bootblock.o bootblock
        ./sign.pl bootblock
```

Linking and loading

- Linking
  - Combining multiple code modules into a single executable
  - E.g., use standard libraries in your own code
- Loading
  - Process of getting an executable running on the machine


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

- Gene d code
  - a is d ned in the same file at 0x1234, **b is imported**
  - Each i struction 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
```

- b is imported, we don't know yet where it will be

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

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

- Source file a.c

```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
Disassem   ly 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 # 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

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

- Source file a.c

```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
  ...
```

- 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

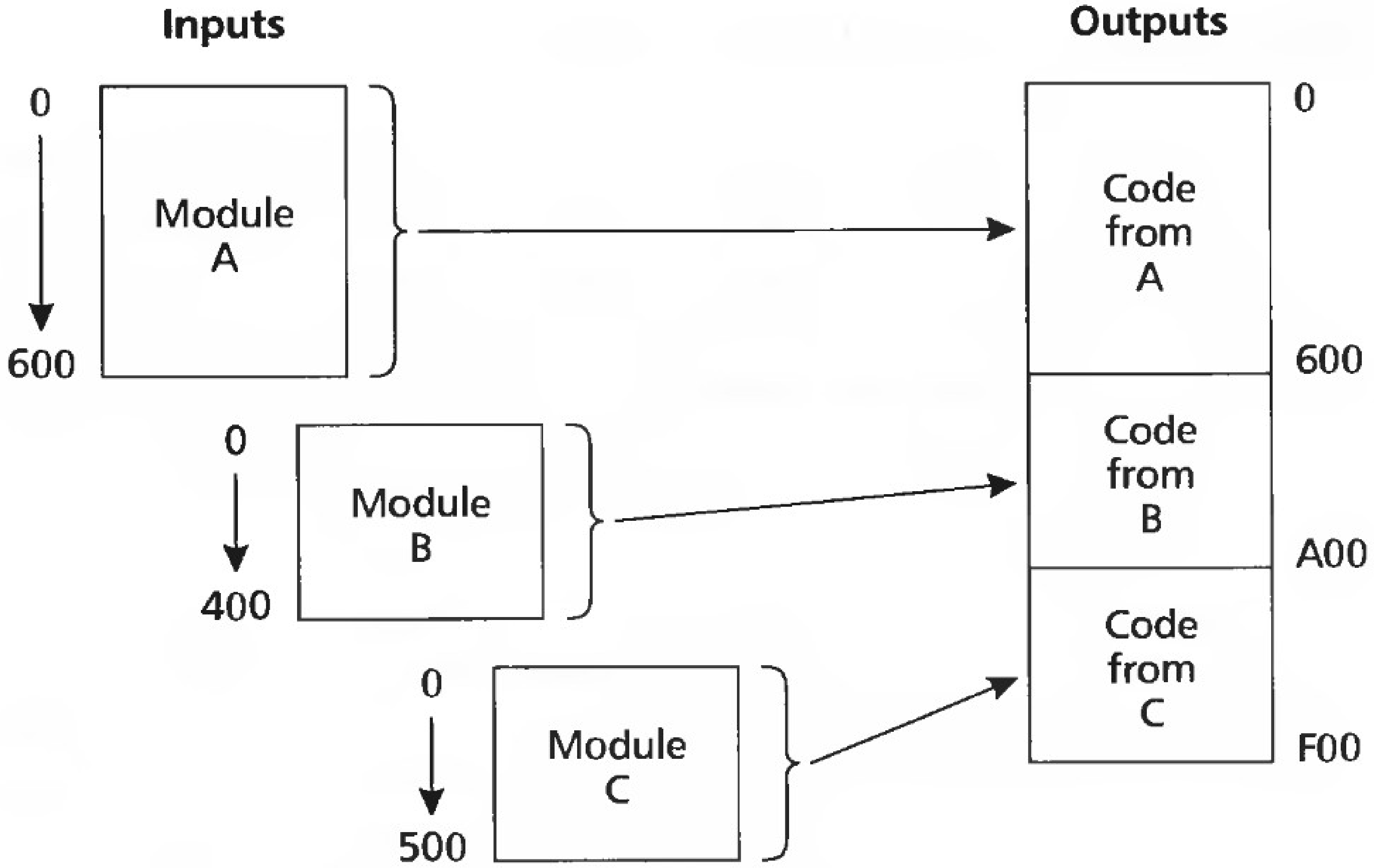
```
Sections:
 Idx Name Size     VMA      LMA      File off Algn
  0 .text 0000001c 00000000 00000000 00000020 2**2
    CONTENTS, ALLOC, LOAD, RELOC, CODE
  1 .data 00000000 0000001c 0000001c 0000003c 2**2
    CONTENTS, ALLOC, LOAD, DATA
Disassembly of section .text:
  00000000 <_a>:
  0: 55              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()

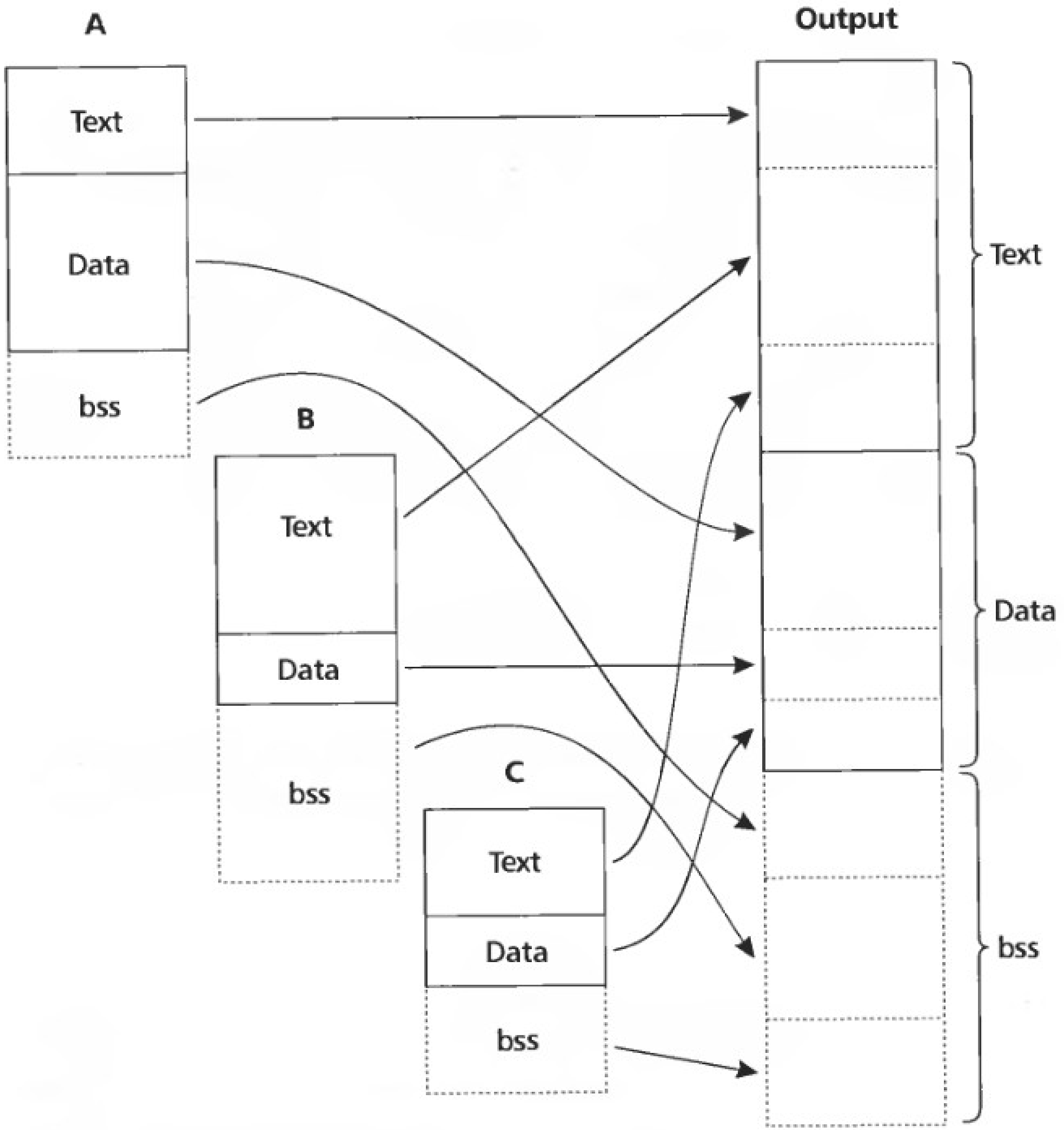
Producing an executable

- 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

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

Tasks involved

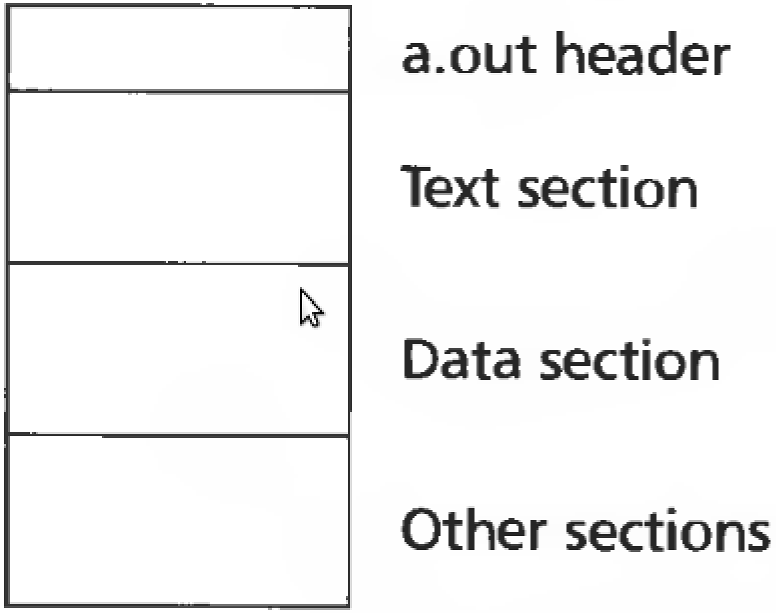
- Program loading
  - Copy a program from disk to memory so it is ready to run
    - Allocation of memory
    - Setting protection bits (e.g. read only)
- Relocation
  - Assign load address to each object file
  - Adjust the code
- Symbol resolution
  - Resolve symbols imported from other object files

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

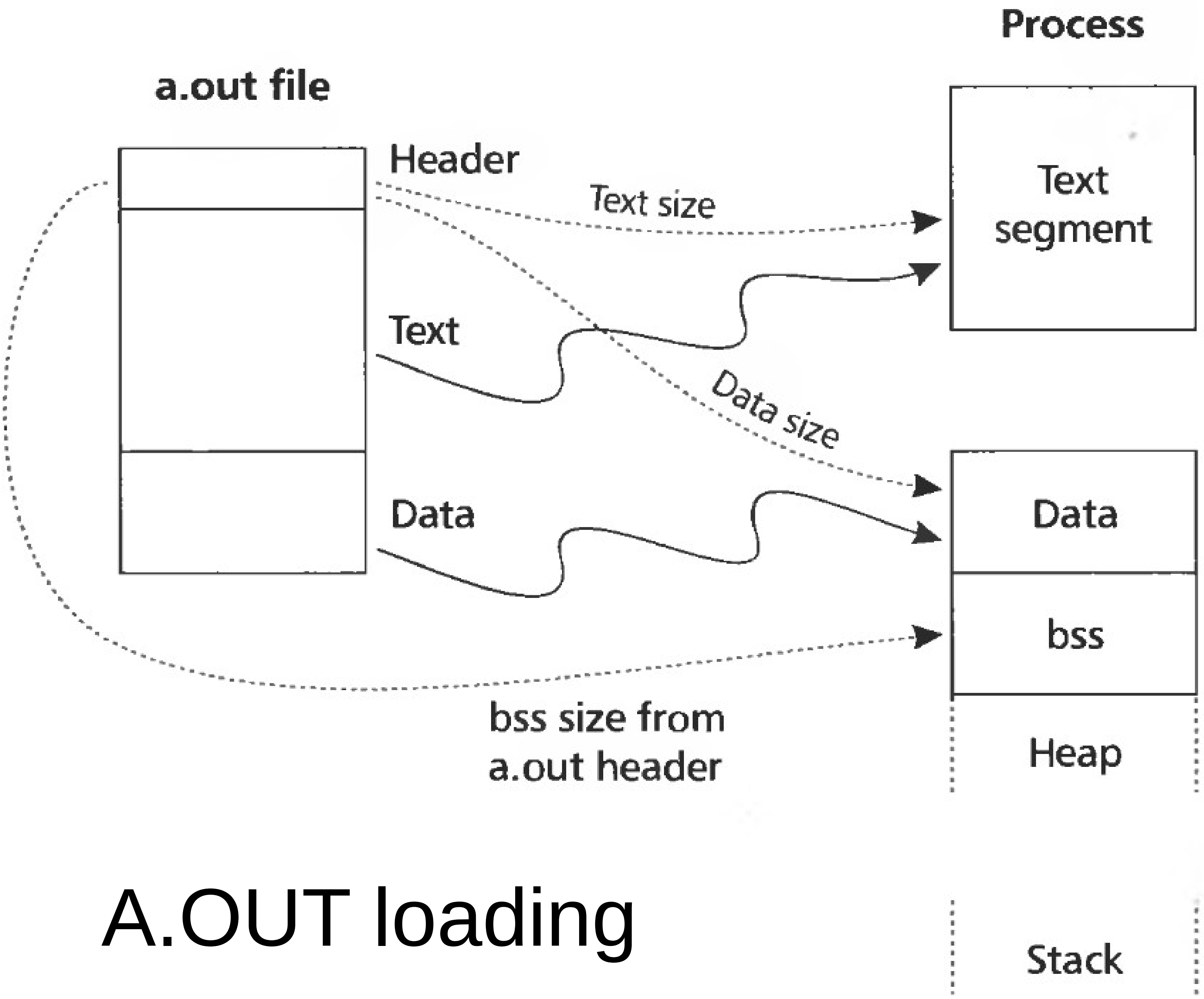
Example: UNIX A.OUT



- 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

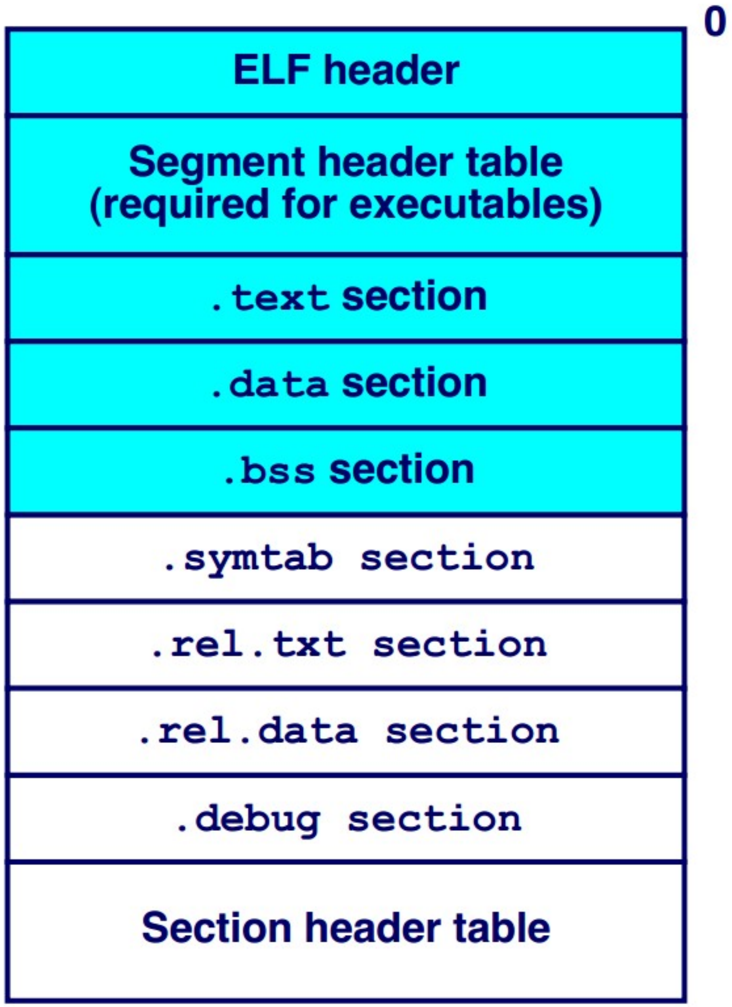
- Code

`.data` section

- Initialized global variables

`.bss` section

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



0

ELF header

Segment header table
(required for executables)

`.text` section

`.data` section

`.bss` section

`.symtab section`

`.rel.txt section`

`.rel.data section`

`.debug section`

Section header table

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

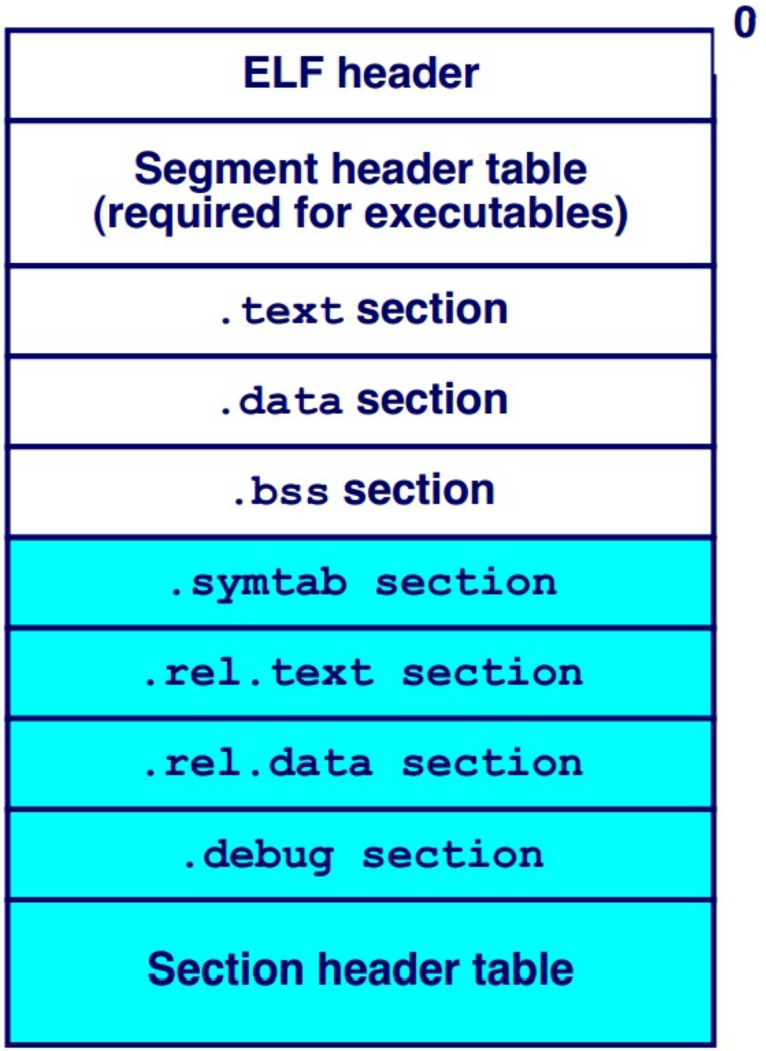
- Relocation info for `.data` section
- Addresses of pointer data that will need to be modified in the merged executable

`.debug` section

- Info for symbolic debugging (`gcc -g`)

Section header table

- Offsets and sizes of each section



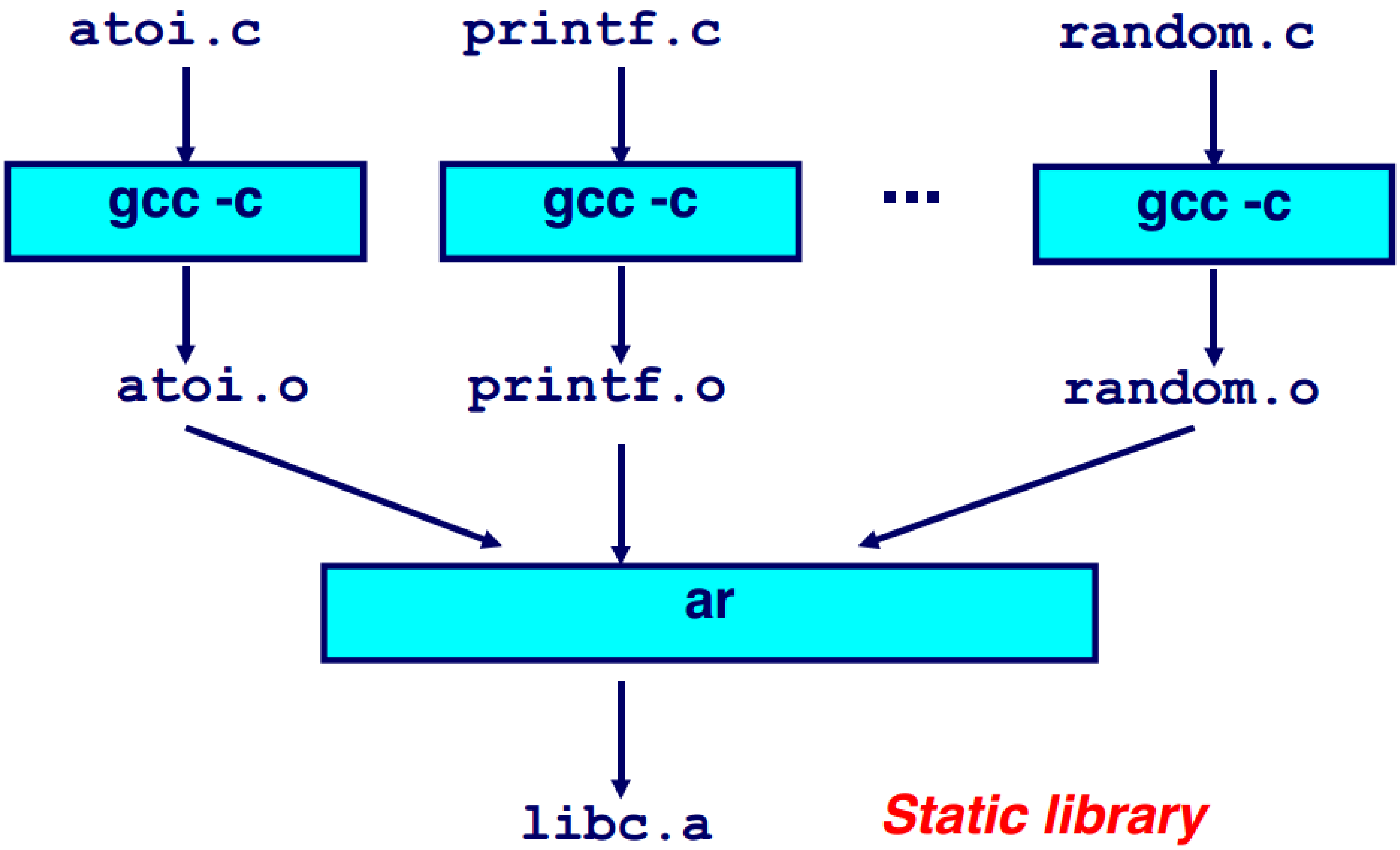
Static libraries

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

Shared libraries
(.so or .dll)

Motivation

- 1000 programs in a typical UNIX system
- 1000 copies of printf

- **How big is printf actually?**

Motivation

- 1000 programs in a typical UNIX system
- 1000 copies of printf
  - **Printf is a large function**
  - **Handles conversion of multiple types to strings**
  - **5-10K**
- This means 5-10MB of disk is wasted on printf
- Runtime memory costs are
  - 10K x number of running programs

Shared libraries

- Motivation
  - Share code of a library across all processes
    - E.g. libc is linked by all processes in the system
  - Code section should remain identical
    - To be shared read-only
  - What if library is loaded at different addresses?
    - Remember it needs to be relocated

Position independent code

Position independent code (PIC)

- Main idea:
  - Generate code in such a way that it can work no matter where it is located in the address space
  - Share code across all address spaces

Thank you!

Function pointers

```
#include <stdio.h>

void func_a(void){
    printf("func_a\n");
    return;
}

void func_b(void) {
    printf("func_b\n");
    return;
}

int main(int ac, char **av)
{
    void (*fp)(void);

    fp = func_b;
    fp();
    return;
}
```

Function pointers

```
08048432 <func_b>:
 8048432:       55                      push   %ebp
 8048433:       89 e5                   mov    %esp,%ebp
 8048435:       83 ec 18                sub    $0x18,%esp
 8048438:       c7 04 24 07 85 04 08    movl   $0x8048507,(%esp)
 804843f:       e8 ac fe ff ff          call   80482f0 <puts@plt>
 8048444:       90                      nop
 8048445:       c9                      leave
 8048446:       c3                      ret

08048447 <main>:
 8048447:       55                      push   %ebp
 8048448:       89 e5                   mov    %esp,%ebp
 804844a:       83 e4 f0                and    $0xfffffff0,%esp
 804844d:       83 ec 10                sub    $0x10,%esp
                                        # Load pointer to func_p on the stack
 8048450:       c7 44 24 0c 32 84 04    movl    $0x8048432,0xc(%esp)
 8048457:       08
 8048458:       8b 44 24 0c             mov    0xc(%esp),%eax
 804845c:       ff d0                   call   *%eax
 804845e:       90                      nop
 804845f:       c9                      leave
 8048460:       c3                      ret
```

Function pointers

```
08048432 <func_b>:
 8048432:       55                      push   %ebp
 8048433:       89 e5                   mov    %esp,%ebp
 8048435:       83 ec 18                sub    $0x18,%esp
 8048438:       c7 04 24 07 85 04 08    movl   $0x8048507,(%esp)
 804843f:       e8 ac fe ff ff          call   80482f0 <puts@plt>
 8048444:       90                      nop
 8048445:       c9                      leave
 8048446:       c3                      ret

08048447 <main>:
 8048447:       55                      push   %ebp
 8048448:       89 e5                   mov    %esp,%ebp
 804844a:       83 e4 f0                and    $0xfffffff0,%esp
 804844d:       83 ec 10                sub    $0x10,%esp
                                        # Load pointer to func_p on the stack
 8048450:       c7 44 24 0c 32 84 04    movl   $0x8048432,0xc(%esp)
 8048457:       08

                                        # Move func_b into %eax
 8048458:       8b 44 24 0c             mov     0xc(%esp),%eax
 804845c:       ff d0                   call    *%eax  # Call %eax
 804845e:       90                      nop
 804845f:       c9                      leave
 8048460:       c3                      ret
```

nm a.out

```
0804a01c B __bss_start
0804a01c b completed.6591
0804a014 D __data_start
0804a014 W data_start
….
0804a01c D _edata
0804a020 B _end
08048484 T _fini
...
08048294 T _init
...
080483ed T main
…
080482f0 T _start
...
```