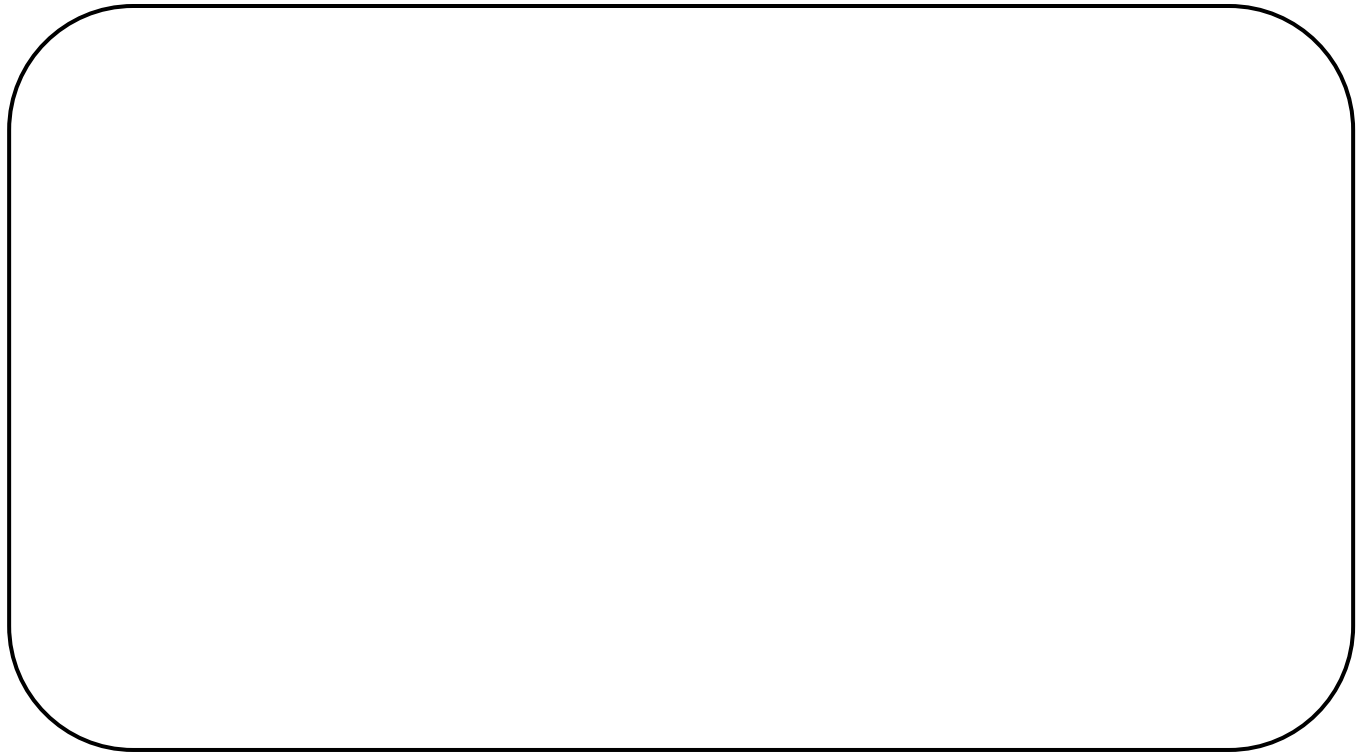
Linking

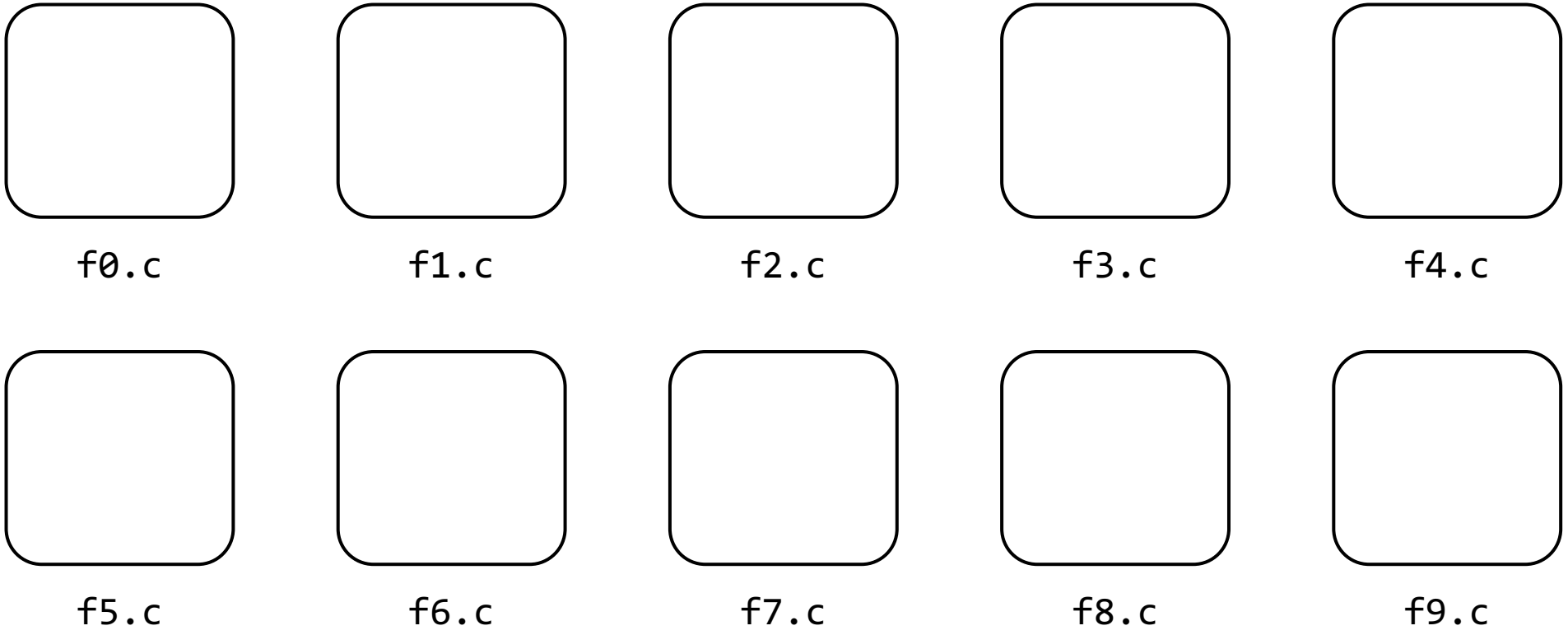
11.13回课

Linking

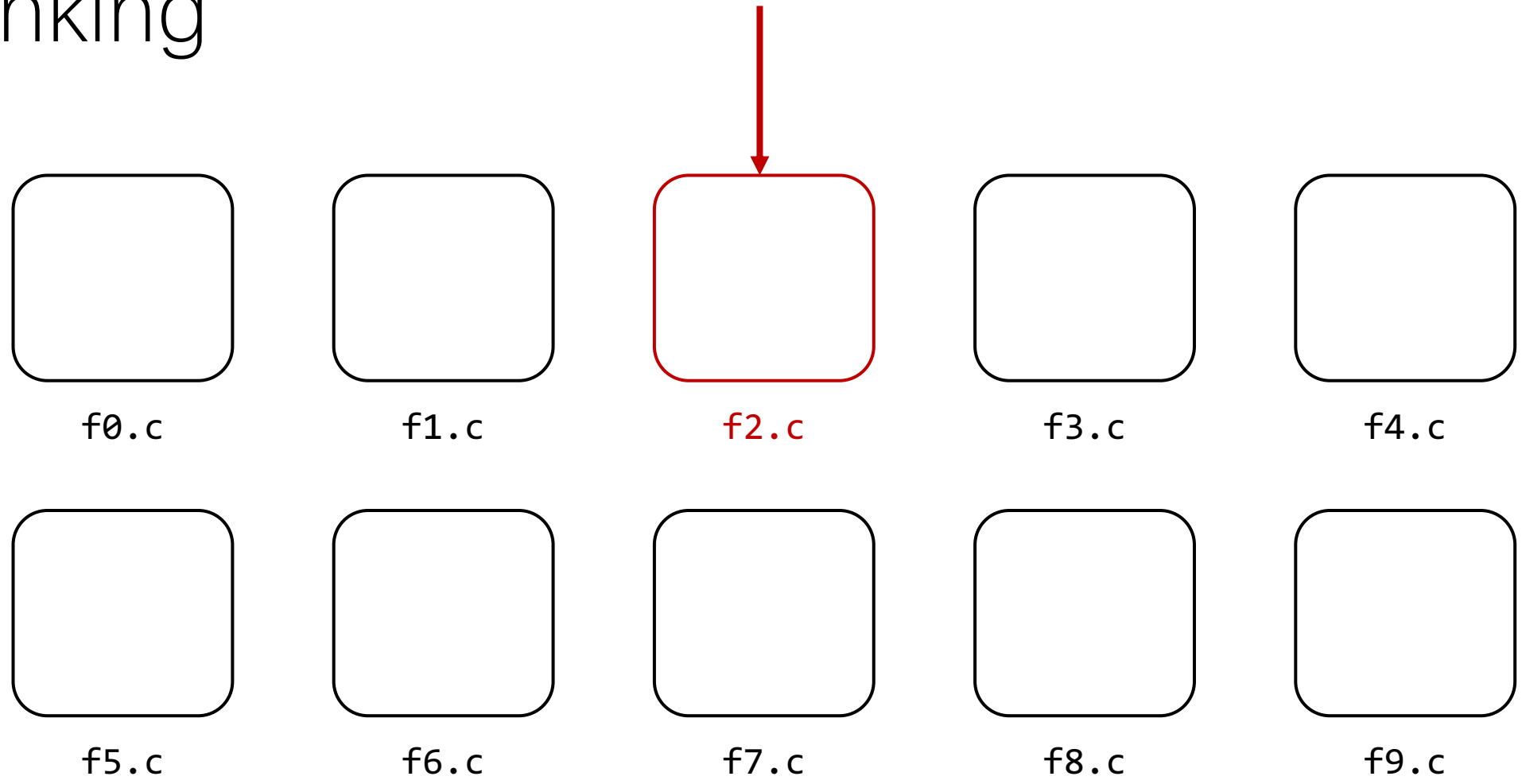


`main.c`

Linking



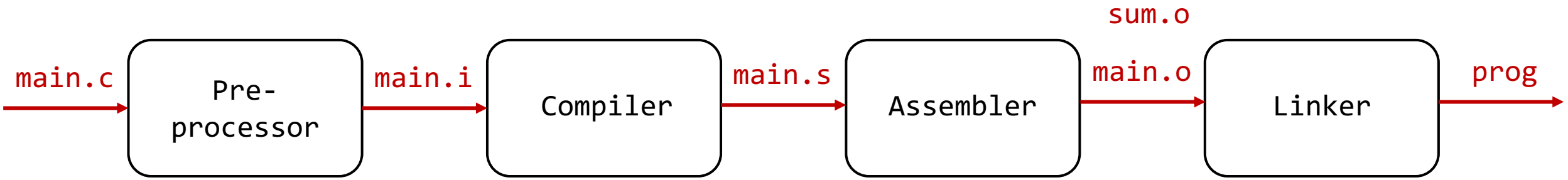
Linking



Linking

- Build Large Programs.（构造大型程序）
- Avoid dangerous programming errors.（避免一些编程错误）
- Understand how language scoping rules.（理解语言作用域规则）
- Understand other important system concepts.（理解其它重要系统概念）
- Exploit shared libraries.（更好地利用共享库）

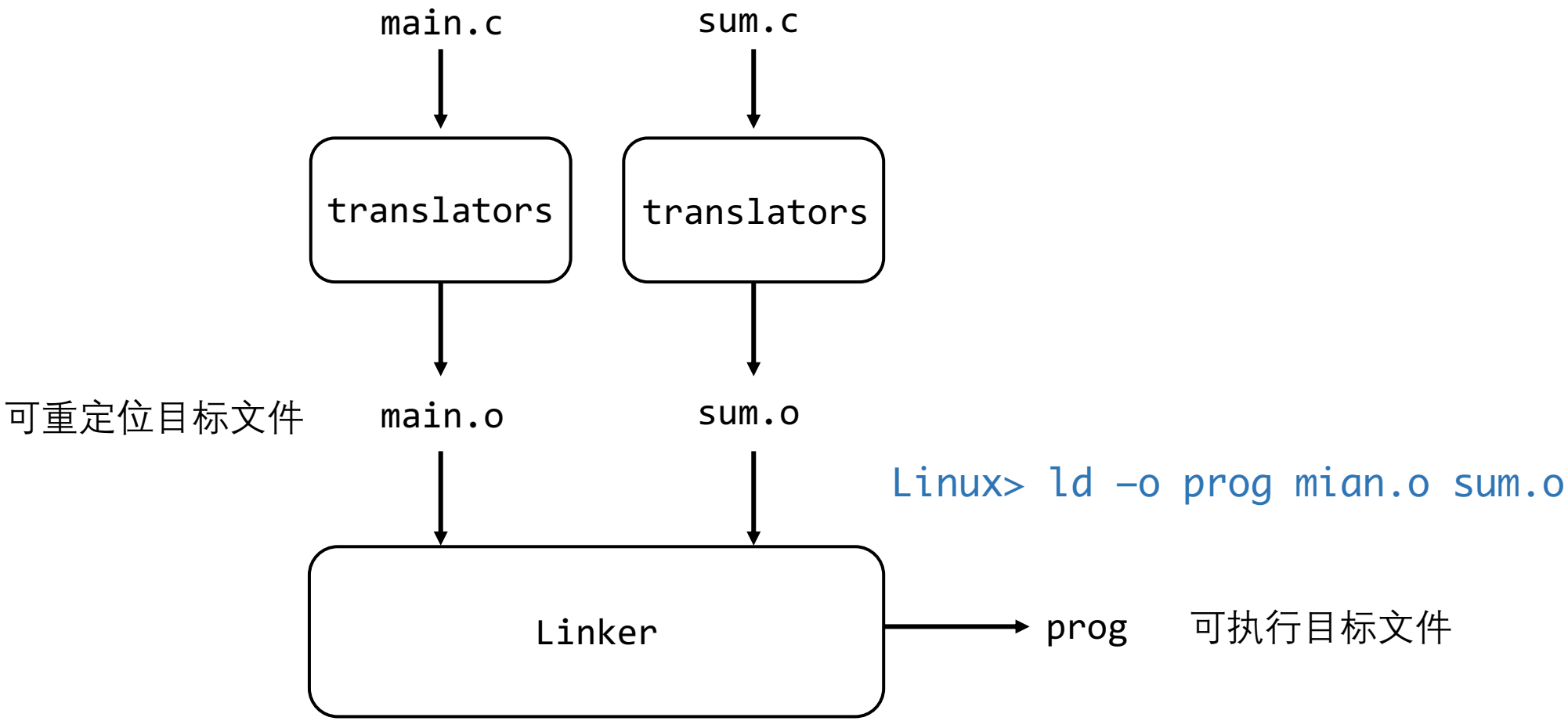
The Compilation System



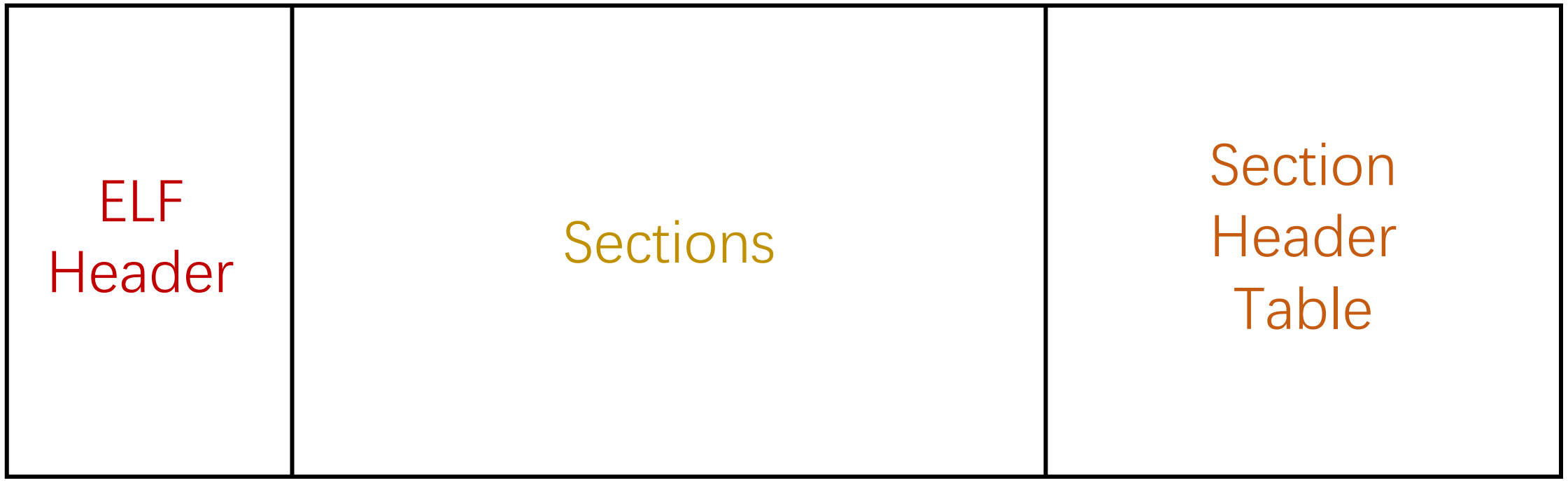

```
Linux> gcc -Og -o main.c sum.c

Linux> ./prog
```

Linking (static)



Relocatale Object Files (可重定位目标文件)



ELF : Executable Linkable Format 可执行可链接格式 (for x86-64 Linux)

ELF Header

```
ubuntu@icsdidu:~/linkdemo$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          576 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

ELF Header

```
ubuntu@icsdidu:~/linkdemo$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          576 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

7f : \DEL
45: ‘E’
4c: ‘L’
46: ‘F’
02: ELF文件类型
0x1 32位
0x2 64位
第一个01:
字节序
0x1 小端法
0x2 大端法
第二个01:
版本号 默认1

ELF Header

```
ubuntu@icsdidu:~/linkdemo$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          576 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

文件类型为可重定位文件

另外还有两种文件类型：

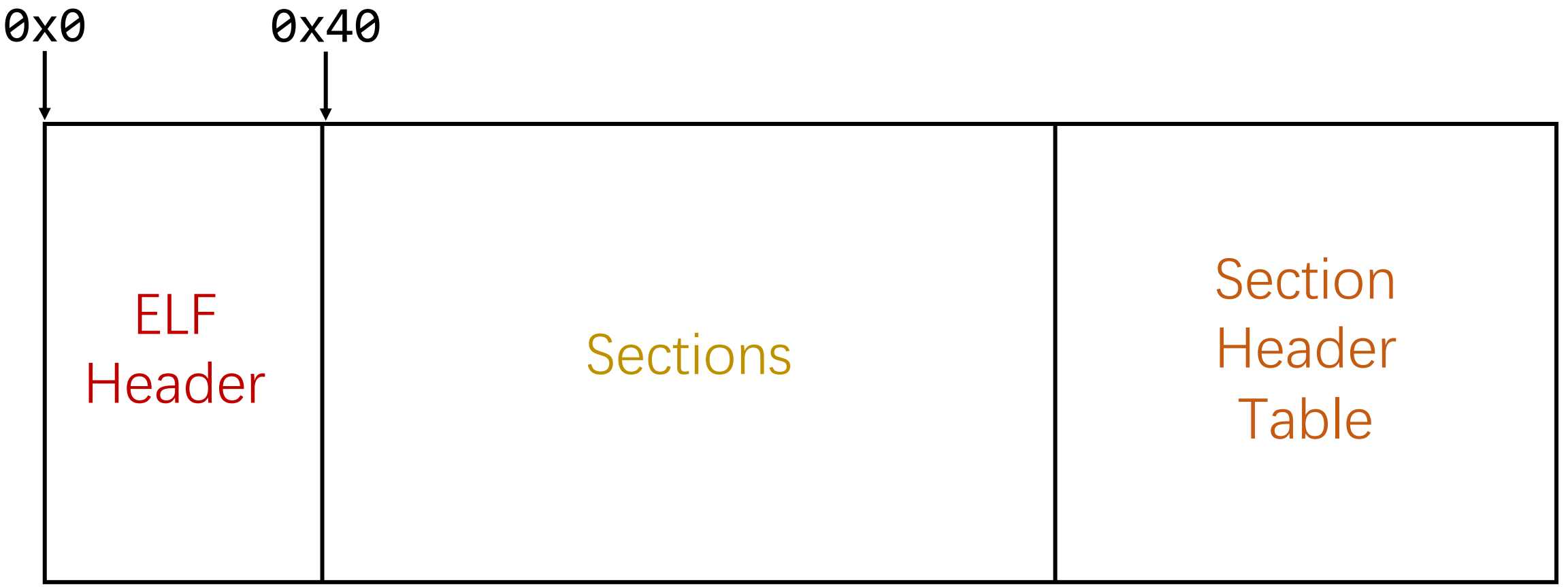
- 可执行文件
- 共享文件

ELF Header

```
ubuntu@icsdidu:~/linkdemo$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          576 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

头大小为64

Relocatale Object Files (可重定位目标文件)



ELF Header

```
ubuntu@icsdidu:~/linkdemo$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          576 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

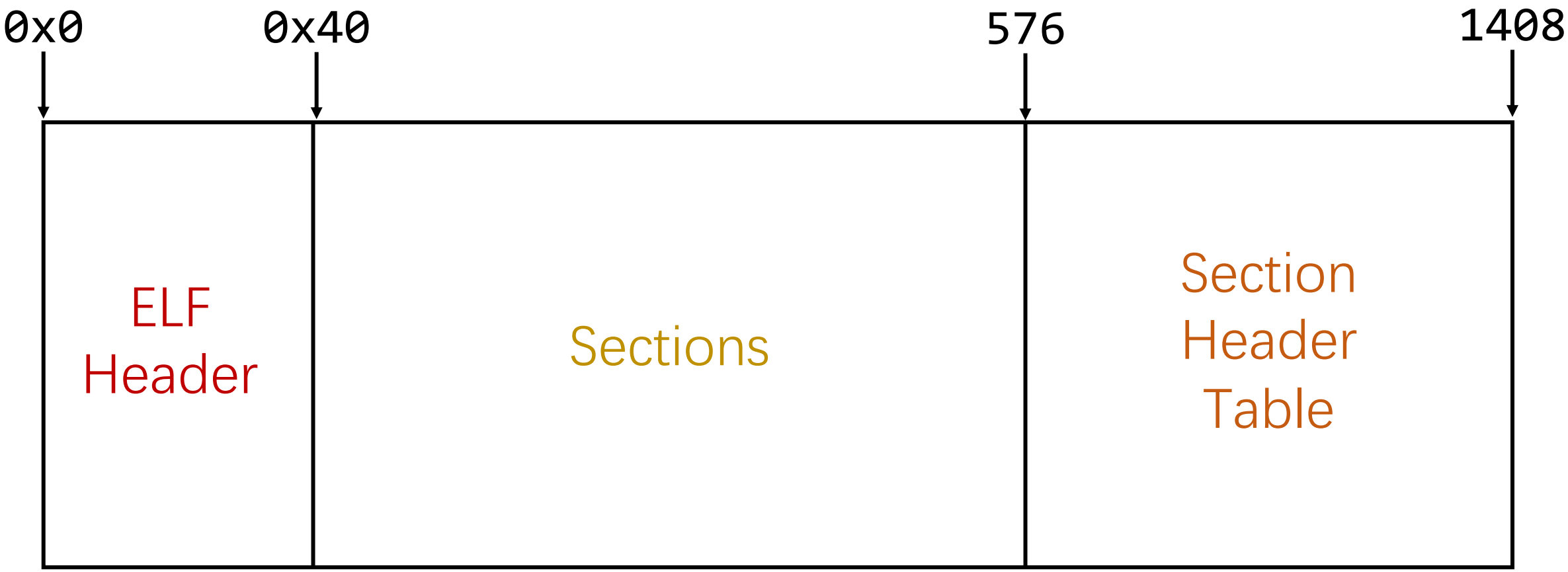
Section header table起始为576

每个section header大小为64

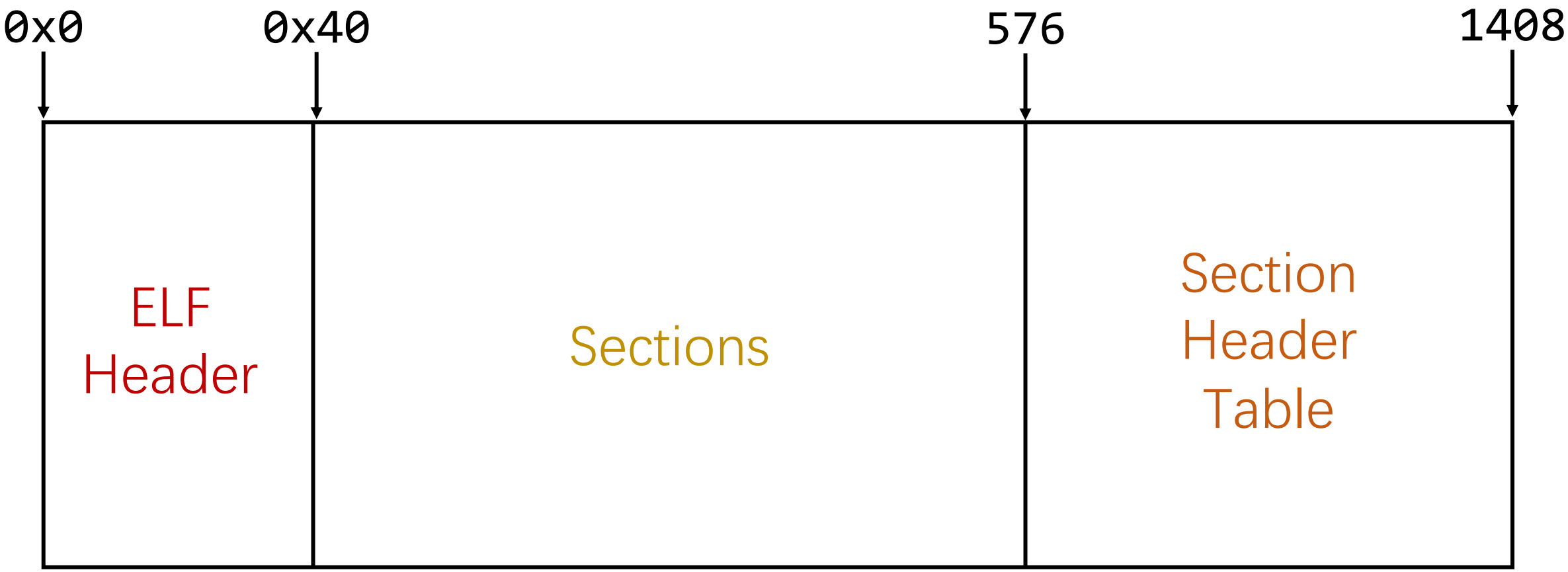
一共有13个header

64 * 13 = 832

Relocatale Object Files (可重定位目标文件)



Relocatale Object Files (可重定位目标文件)




```
ubuntu@icsdidu:~/linkdemo$ wc -c main.o
1408 main.o
```

Relocatale Object Files

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

```
Linux> gcc -c main.c
```

Section Header Table

```
ubuntu@icsdidu:~/linkdemo$ readelf -S main.o
There are 14 section headers, starting at offset 0x3e0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       000000000000005f  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  000002c0
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a0
       0000000000000008  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000a8
       0000000000000008  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000a8
       000000000000000b  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000b3
       0000000000000027  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000da
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE             0000000000000000  000000e0
       0000000000000020  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS         0000000000000000  00000100
       0000000000000058  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA             0000000000000000  00000338
       0000000000000030  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB           0000000000000000  00000158
       0000000000000138  0000000000000018          12     8     8
  [12] .strtab           STRTAB           0000000000000000  00000290
       000000000000002d  0000000000000000           0     0     1
  [13] .shstrtab         STRTAB           0000000000000000  00000368
       0000000000000074  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

Offset : 偏移量（起始位置）

Size : 每个section的大小

Section Header Table

```
ubuntu@icsdidu:~/linkdemo$ readelf -S main.o
There are 14 section headers, starting at offset 0x3e0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       000000000000005f  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  000002c0
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a0
       0000000000000008  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000a8
       0000000000000008  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000a8
       000000000000000b  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000b3
       0000000000000027  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000da
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE             0000000000000000  000000e0
       0000000000000020  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS         0000000000000000  00000100
       0000000000000058  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA             0000000000000000  00000338
       0000000000000030  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB           0000000000000000  00000158
       0000000000000138  0000000000000018          12     8     8
  [12] .strtab           STRTAB           0000000000000000  00000290
       000000000000002d  0000000000000000           0     0     1
  [13] .shstrtab         STRTAB           0000000000000000  00000368
       0000000000000074  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

.text : 已编程的机器代码

.data: 已初始化的全局和静态C变量

.bss : 未初始化的全局和静态C变量
所有被初始化为0的全局或静态变量

.rodata : 只读数据

.symtab : 符号表

Section Header Table

```
ubuntu@icsdidu:~/linkdemo$ readelf -S main.o
There are 14 section headers, starting at offset 0x3e0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       000000000000005f  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  000002c0
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a0
       0000000000000008  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000a8
       0000000000000008  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000a8
       000000000000000b  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000b3
       0000000000000027  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000da
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE             0000000000000000  000000e0
       0000000000000020  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS         0000000000000000  00000100
       0000000000000058  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA             0000000000000000  00000338
       0000000000000030  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB           0000000000000000  00000158
       0000000000000138  0000000000000018          12     8     8
  [12] .strtab           STRTAB           0000000000000000  00000290
       000000000000002d  0000000000000000           0     0     1
  [13] .shstrtab         STRTAB           0000000000000000  00000368
       0000000000000074  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

注意到.bss 与 .rodata 起始位置相同

但我们之前确实定义了未初始化的全局变量 value

main.c

linkdemo > C main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Section Header Table

```
ubuntu@icsdidu:~/linkdemo$ readelf -S main.o
There are 14 section headers, starting at offset 0x3e0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       000000000000005f  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  000002c0
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a0
       0000000000000008  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000a8
       0000000000000008  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000a8
       000000000000000b  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000b3
       0000000000000027  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000da
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE             0000000000000000  000000e0
       0000000000000020  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS         0000000000000000  00000100
       0000000000000058  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA             0000000000000000  00000338
       0000000000000030  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB           0000000000000000  00000158
       0000000000000138  0000000000000018          12     8     8
  [12] .strtab           STRTAB           0000000000000000  00000290
       000000000000002d  0000000000000000           0     0     1
  [13] .shstrtab         STRTAB           0000000000000000  00000368
       0000000000000074  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

事实上，在目标文件中.bss不占据实际的空间，仅仅只是一个占位符，用来区分已经初始化和没有初始化的变量，提高空间效率.

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

打印符号表

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Name : .strtab中的字节偏移，指向符号名字符串

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

为什么没有x？

main.c

linkdemo > C main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}
```

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

局部变量在运行时被栈和寄存器管理，链接器对此类符号不感兴趣

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

名称修饰，防止静态变量的名字冲突

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Value : 符号地址（对于可重定位文件是所属section内的相对地址，对于可执行文件是一个绝对的运行时地址）

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Size : 条目的大小

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Type : 条目的类型

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Bind：条目的属性是本地的还是全局的

Symbols

- Global Symbols（全局符号） 非静态的C函数和全局变量
- External Symbols（外部符号） 其它模块中定义的非静态C函数和全局变量
- Local Symbols（局部符号） 静态C函数和静态变量

Symbols

- Global Symbols（全局符号） 非静态的C函数和全局变量
- External Symbols（外部符号） 其它模块中定义的非静态C函数和全局变量
- Local Symbols（局部符号） 静态C函数和静态变量

用Static属性来保护你的变量和函数是很好的习惯

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

Ndx : 所属section的索引

```
ubuntu@icsdidu:~/linkdem
There are 14 section hea

Section Headers:
  [Nr] Name
       Size
  [ 0]
       0000000000000000
  [ 1] .text
       000000000000005f
  [ 2] .rela.text
       0000000000000078
  [ 3] .data
       0000000000000008
  [ 4] .bss
       0000000000000008
  [ 5] .rodata
       000000000000000b
  [ 6] .comment
       0000000000000027
  [ 7] .note.GNU-stack
       0000000000000000
  [ 8] .note.gnu.pr[...]
       0000000000000020
  [ 9] .eh_frame
       0000000000000058
  [10] .rela.eh_frame
       0000000000000030
  [11] .symtab
       0000000000000138
  [12] .strtab
       000000000000002d
  [13] .shstrtab
       0000000000000074
Key to Flags:
  W (write), A (alloc),
  L (link order), O (ext
  C (compressed), x (unk
  D (mbind), l (large),
```

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

三个特殊的伪section：ABS (不该被重定位)
UND(未定义)
COMMON(未初始化的全局变量)

```
ubuntu@icsdidu:~/linkdem
There are 14 section hea

Section Headers:
  [Nr] Name
       Size
  [ 0]
       0000000000000000
  [ 1] .text
       000000000000005f
  [ 2] .rela.text
       0000000000000078
  [ 3] .data
       0000000000000008
  [ 4] .bss
       0000000000000008
  [ 5] .rodata
       000000000000000b
  [ 6] .comment
       0000000000000027
  [ 7] .note.GNU-stack
       0000000000000000
  [ 8] .note.gnu.pr[...]
       0000000000000020
  [ 9] .eh_frame
       0000000000000058
  [10] .rela.eh_frame
       0000000000000030
  [11] .symtab
       0000000000000138
  [12] .strtab
       000000000000002d
  [13] .shstrtab
       0000000000000074
Key to Flags:
  W (write), A (alloc),
  L (link order), O (ext
  C (compressed), x (unk
  D (mbind), l (large),
```

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

.bss : 未初始化的静态变量，以及初始化为0的全局或静态变量

COMMON : 未初始化的全局变量

注：这个规则只是一个惯例，具体行为取决于编译器，比如上图value作为未初始化的全局变量被分配给了.bss

符号解析

对那些定义和引用在相同模块的的符号的引用，其解析是简单明了的

而对于一个不是在当前模块定义的符号，链接器就会假设其是在某个其它模块中定义的，并生成一个UDN条目交给链接器处理

如果链接器在所有输入的模块中都找不到这个被引用符号的定义，就输出错误信息并终止

解析多重定义的全局符号

- Strong symbols (强符号)：

  函数和已初始化的全局变量

- Weak symbols (弱符号)：

  未初始化的全局变量

Rule 1

不允许多个同名强符号

Rule 2

强弱同名选强

Rule 3

多弱同名任意选

Rule 1

不允许多个同名强符号

Rule 2

强弱同名选强

Rule 3

多弱同名任意选

Rule2和Rule3会导致一些不易察觉的运行时错误

课本P473给出了一个很好的例子

Rule 1

不允许多个同名强符号

Rule 2

强弱同名选强

Rule 3

多弱同名任意选

Rule2和Rule3会导致一些不易察觉的运行时错误

课本P473给出了一个很好的例子

由于Rule2和Rule3的存在，编译器遇到一个弱全局符号时不知道其他模块是否也定义了同名的符号，它无法预测使用哪一个定义，所以将它分配给COMMON，将选择权留给链接器.

而如果这个全局符号被初始化过，它就是一个强全局符号，编译器会自信地将它分配给.bss或者.data

.symtab

```
ubuntu@icsdidu:~/linkdemo$ readelf -s main.o

Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    3 a.1
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 b.0
     8: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 count
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    4 value
    10: 0000000000000000    43 FUNC    GLOBAL DEFAULT    1 func
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    12: 000000000000002b    52 FUNC    GLOBAL DEFAULT    1 main
```

main.c

linkdemo > main.c

```c
#include <stdio.h>

int count = 10;
int value;

void func(int sum){
    printf("sum is:%d\n",sum);
}

int main(){
    static int a = 1;
    static int b = 0;
    int x = 1;
    func(a + b + x);
    return 0;
}

```

Linking with Static Libraries

Why?

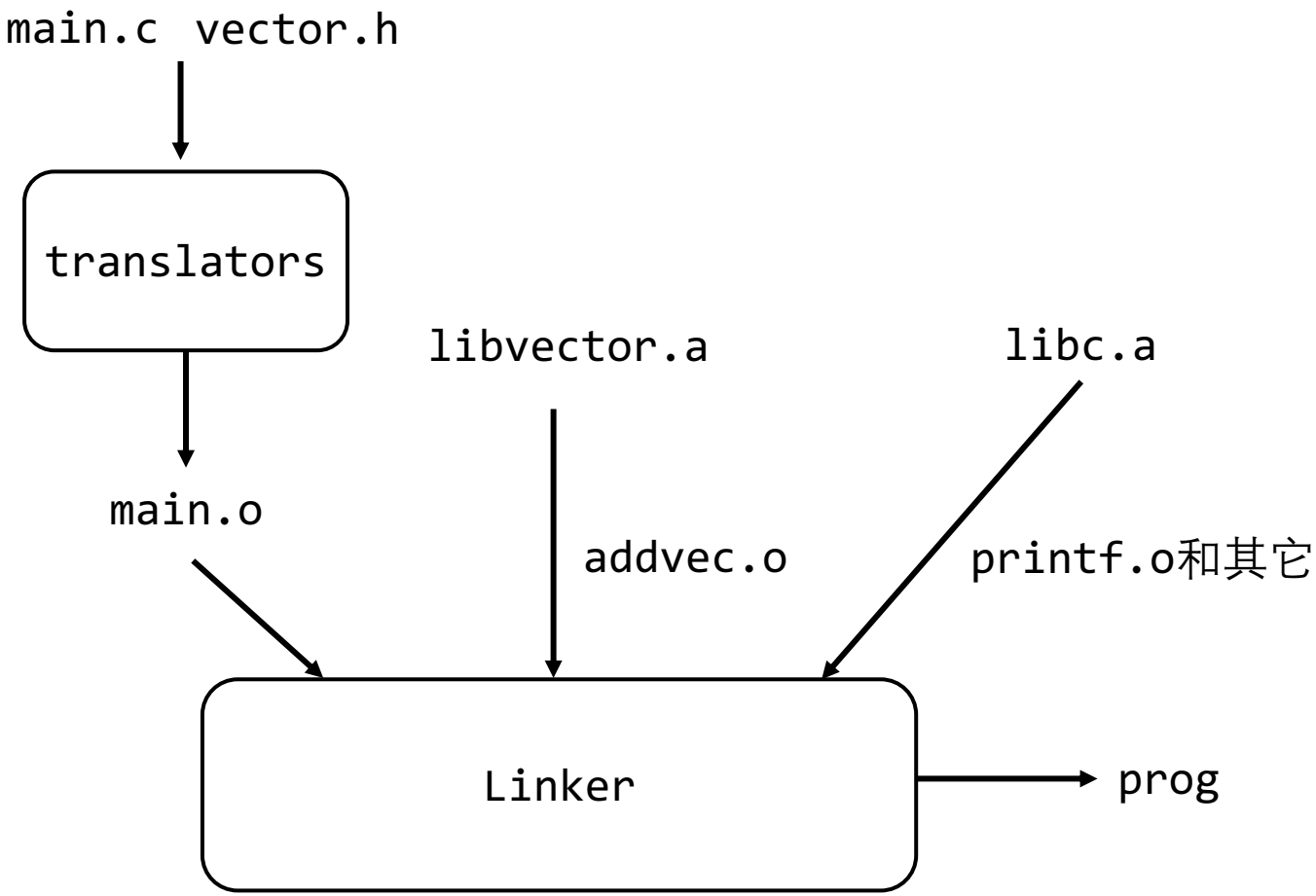
见课本475页

Linking with Static Libraries

在Linux系统中，静态库以一种称为*存档*的特殊文件格式(.a)存放在磁盘中。

存档文件是一组连接起来的可重定位目标文件的集合，有一个头部来描述每个成员目标文件的大小和位置

Linking with Static Libraries



Static Linking

链接器如何使用静态库来解析引用？

Static Linking

```
Linux> gcc foo.c libx.a libx.y
       ------------------->
```

Static Linking

```
Linux> gcc foo.c libx.a libx.y
       ------------->
```

由于是按顺序进行链接，所以输入的顺序很重要，必要时还需要重复库

Relocation

- Relocating sections and symbol definitions
  重定位节和符号定义

- Relocating symbol references with sections
  重定位符号引用

Relocation

- Relocating sections and symbol definitions
  重定位节和符号定义

链接器将所有相同类型的section合并成同一个section
然后将运行时内存地址赋给新的section和定义的每个符号
这一步结束后每条指令和全局变量都有唯一的运行时内存地址了
但此时代码节和数据节对符号的引用的地址还是旧的，所以就需要重定位符号引用

Relocation

- Relocating symbol references with sections
  重定位符号引用

Relocation

- Relocating symbol references with sections
  重定位符号引用

汇编器在处理一个对最终位置未知的引用时会生成一个重定位条目，它告诉链接器在合并时如何修改这个引用

重定位条目的结构如下：

```
typedef struct {
    long offset;          //被修改引用的节偏移
    long type:32,         //修改引用的模式
         symbol:32;       //引用应该指向的符号
    long addend;          //偏移调整
}ELf64_Rela;
```

type = R_X86_64_PC32表示相对地址的引用，R_X86_64_32表示绝对地址的引用

直接理解有点抽象，我们结合一个例子来看

这里以书上对call指令引用sum的重定向为例

```
0000000000000000<main>:
   0:    48 83 ec 08              sub  $0x8,%rsp
   4:    be 02 00 00 00           mov  $0x2,%esi
   9:    bf 00 00 00 00           mov  $0x0,%edi
   e:    e8 00 00 00 00           call 13 <main+0x13>

   ...

  13:    48 83 c4 08              add  $0x8,%rsp
  17:    c3                       ret
```

main.o的反汇编代码如图所示

```
0000000000000000<main>:
   0:    48 83 ec 08              sub  $0x8,%rsp
   4:    be 02 00 00 00           mov  $0x2,%esi
   9:    bf 00 00 00 00           mov  $0x0,%edi
   e:    e8 00 00 00 00           call 13 <main+0x13>
  13:    48 83 c4 08              add  $0x8,%rsp
  17:    c3                       ret
```

这里call函数想调用的sum函数，但sum的运行时位置还没有确定，所以call函数姑且让call调用+0
即下一条指令+0，也就是0x13

```
0000000000000000<main>:
   0:    48 83 ec 08             sub  $0x8,%rsp
   4:    be 02 00 00 00          mov  $0x2,%esi
   9:    bf 00 00 00 00          mov  $0x0,%edi
   e:    e8 00 00 00 00          call 13 <main+0x13>
  13:    48 83 c4 08             add  $0x8,%rsp
  17:    c3                      ret
```

与此同时汇编器生成了一个重定位条目r 存放在.rel.text中
其结构如下：
r.offset = 0xf
r.symbol = sum
r.type = R_X86_64_PC32
r.Addend = -4

```
0000000000000000<main>:
   0:    48 83 ec 08             sub  $0x8,%rsp
   4:    be 02 00 00 00          mov  $0x2,%esi
   9:    bf 00 00 00 00          mov  $0x0,%edi
   e:    e8 00 00 00 00          call 13 <main+0x13>
  13:    48 83 c4 08             add  $0x8,%rsp
  17:    c3                      ret
```

与此同时汇编器生成了一个重定位条目r 存放在.rel.text中
其结构如下：
r.offset = 0xf
r.symbol = sum
r.type = R_X86_64_PC32
r.addend = -4

其中r.offset的位置恰好就是错误的0x00，我们的工作就是让修改r.offset让call调用r.symbol

```
0000000000000000<main>:
   0:    48 83 ec 08             sub  $0x8,%rsp
   4:    be 02 00 00 00          mov  $0x2,%esi
   9:    bf 00 00 00 00          mov  $0x0,%edi
   e:    e8 00 00 00 00          call 13 <main+0x13>
  13:    48 83 c4 08             add  $0x8,%rsp
  17:    c3                      ret
```

我们已经经过了重定位的第一步
因此，.text(main)和sum的运行时地址已经知道了，记为ADDR(.text)和ADDR(r.symbol)

要修改的0x00此时位于
refaddr=ADDR(main)+r.offset

故：*redaddr = ADDR(r.symbol) – refaddr + r.addend

这里r.addend修正的其实就是refaddr(0x00)与下一条指令(0x48)间隔的4个字节

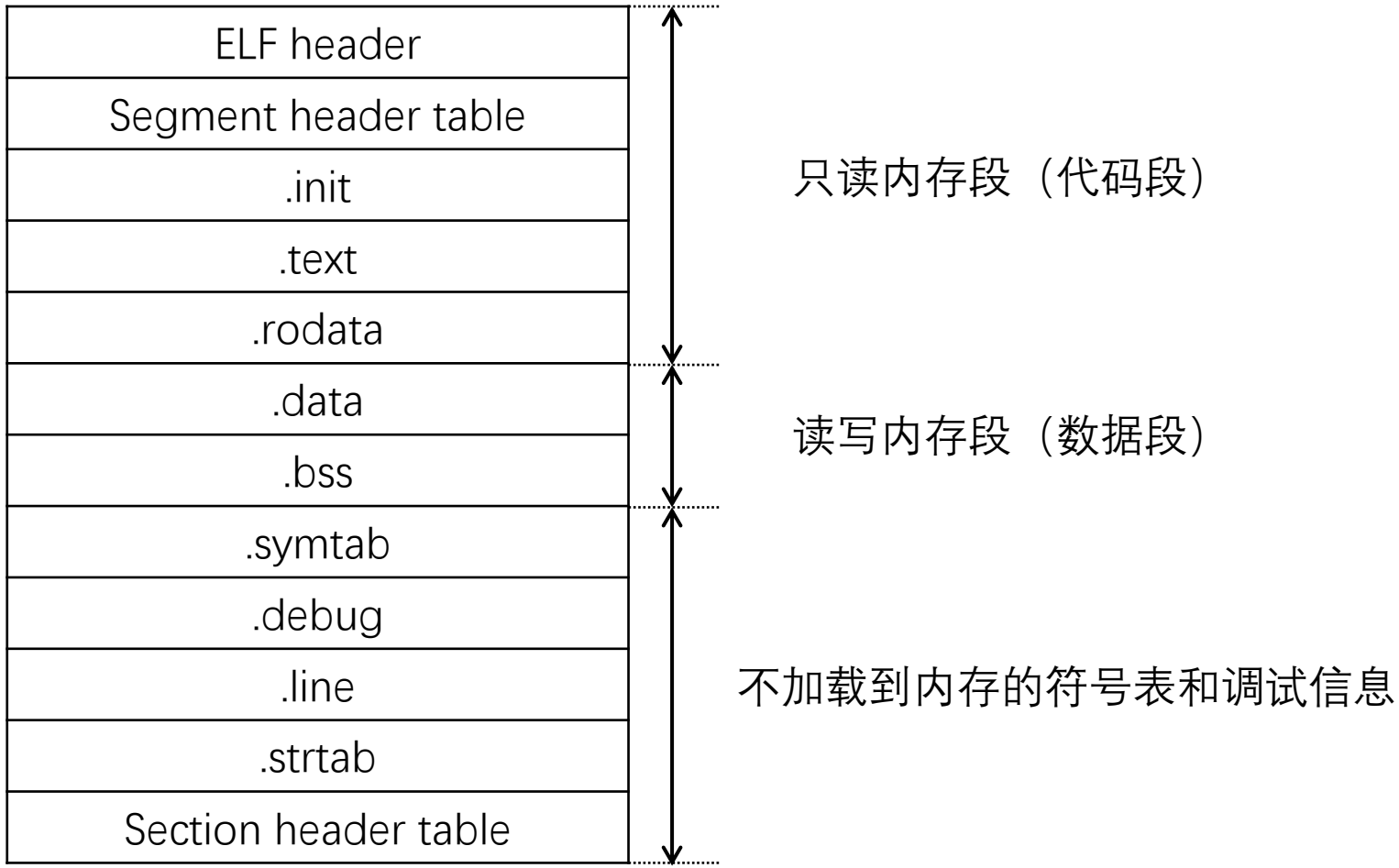
Executable Object Files

ELF header
Segment header table
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
Section header table

- Segment header table段头部表将连续的文件节映射到运行时内存

- 因为可执行文件是完全链接的，所以它不再需要rel节

Executable Object Files



Significant Disadvantages of Static Linking

- Need to be maintained and updated periodically
  静态库需要定期维护或更新

- Almost every C programs use standard I/O functions
  几乎每个C函数都使用标准I/O函数（浪费内存资源）

Dynamic Linking

基本思路：

在生成可执行文件(prog)时，做一些简单的链接。在程序开始执行之前，在程序加载到内存的过程中动态的完成链接过程.

动态库不会被复制到prog，而是直接重定位到某个内存段，然后重定位prog中所有由动态库定义的符号的引用

Dynamic Linking

实现过程：

加载器在加载prog时，会发现其包含一个.interp 的节，它包含了动态链接器(ld-linux.so)的路径名称，加载器会加载和运行这个动态链接器，动态链接器会执行上述的重定位完成链接，再将控制传递给应用程序.