上海科技大学
ShanghaiTech University

# Discussion 5 CALL

Qisheng Jiang
jiangqsh@shanghaitech.edu.cn

立志成才报国裕民

# What is CALL?

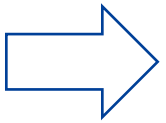


- C: Compiler
- A: Assembler
- L: Linker
- L: Loader
- Sometimes things get messed up when you're using an IDE…

# Overview

上海科技大学
ShanghaiTech University

```c
/* hello.c */

#include <stdio.h>
int main()
{
    printf("hello, world\n");
    return 0;
}
```



```
#   i   n   c   l   u   d   e
35 105 110 99 108 117 100 101
S P   <   s   t   d   i   o   .
32 60 115 116 100 105 111 46
h   >   \ n   \ n   i   n   t   S P
104 62 10 10 105 110 116 32
m   a   i   n   (   )   \ n   {
109 97 105 110 40 41 10 123
```

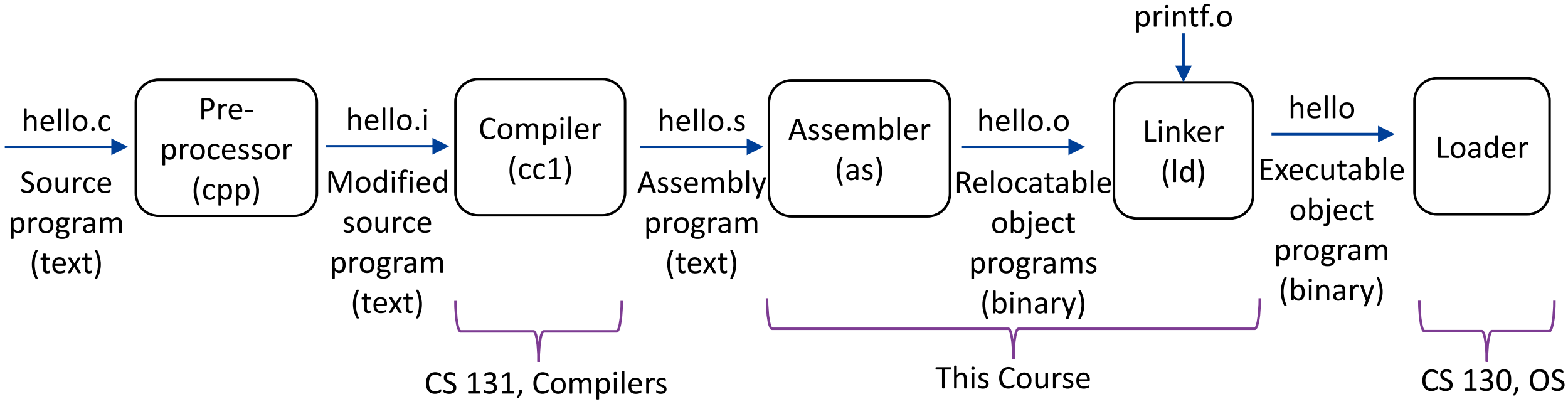
```
# compiler driver, such as gcc
# -On: optimization level

$ gcc -O2 -o hello hello.c
```

# Overview

上海科技大学
ShanghaiTech University

```c
/* hello.c */

#include <stdio.h>
int main()
{
    printf("hello, world\n");
    return 0;
}
```

```
#   i   n   c   l   u   d   e
35 105 110 99 108 117 100 101
S P   <   s   t   d   i   o   .
32 60 115 116 100 105 111 46
h   >   \ n   \ n   i   n   t   S P
104 62 10 10 105 110 116 32
m   a   i   n   (   )   \ n   {
109 97 105 110 40 41 10 123
```



# Preprocessor



- The preprocessor (cpp)
  - Modifies the original C program according to directives that begin with the '#' character.
- Such as:
  - #include
  - #define
  - …

```
# cpp source target

$ cpp hello.c hello.i
```

hello.c U | shell.sh U

hello.c

```c
/* $begin hello */
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
/* $end hello */
```

hello.i U

hello.i

```c
extern void funlockfile (FILE *__str
# 885 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 902 "/usr/include/stdio.h" 3 4

# 3 "hello.c" 2


# 4 "hello.c"
int main()
{
    printf("hello, world\n");
    return 0;
}
```

# Compiler



- The compiler (cc1)
  - Translates the source program in **high-level programming** language (e.g., xxx.c / xxx.i) into the **assembly-language** program (e.g., xxx.s)
- Pipeline:
  - Lexer
  - Parser
  - Semantics
  - Optimization
  - Code Generation

```c
/* main.c */

int sum(int *a, int n);

int array[2] = {1, 2};

int main()
{
    int val = sum(array, 2);
    return val;
}
```

```c
/* sum.c */

int sum(int *a, int n)
{
    int i, s = 0;
    for (i = 0; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

```
# compiler driver, such as gcc
# -On: optimization level

$ gcc -O2 -o prog main.c sum.c
```

```c
/* main.c */

int sum(int *a, int n);

int array[2] = {1, 2};

int main()
{
    int val = sum(array, 2);
    return val;
}
```

```
$ cpp main.c main.i
$ cpp sum.c sum.i

# C compiler (cc1)
# /usr/lib/gcc/x86_64-linux-gnu/11/cc1
$ cc1 -On -o main.s main.i
$ cc1 -On -o sum.s sum.i
```

```
% main.s

main:
.LFB0:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq  %rsp, %rbp
        .cfi_def_cfa_register 6
        subq  $16, %rsp
        movl  $2, %esi
        leaq  array(%rip), %rax
        movq  %rax, %rdi
        call  sum@PLT
        movl  %eax, -4(%rbp)
        movl  -4(%rbp), %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

# Assembler



- The assembler (as)
  - Translates **assembly** program (e.g., xxx.s) into **machine-language** instructions
  - Packages them in a form known as a relocatable object program
  - Stores the result in the object file (e.g., xxx.o)
- Relocatable Object Program
  - Binary encoded, human unreadable
  - ELF file in Linux, but not runnable
  - readelf / objdump

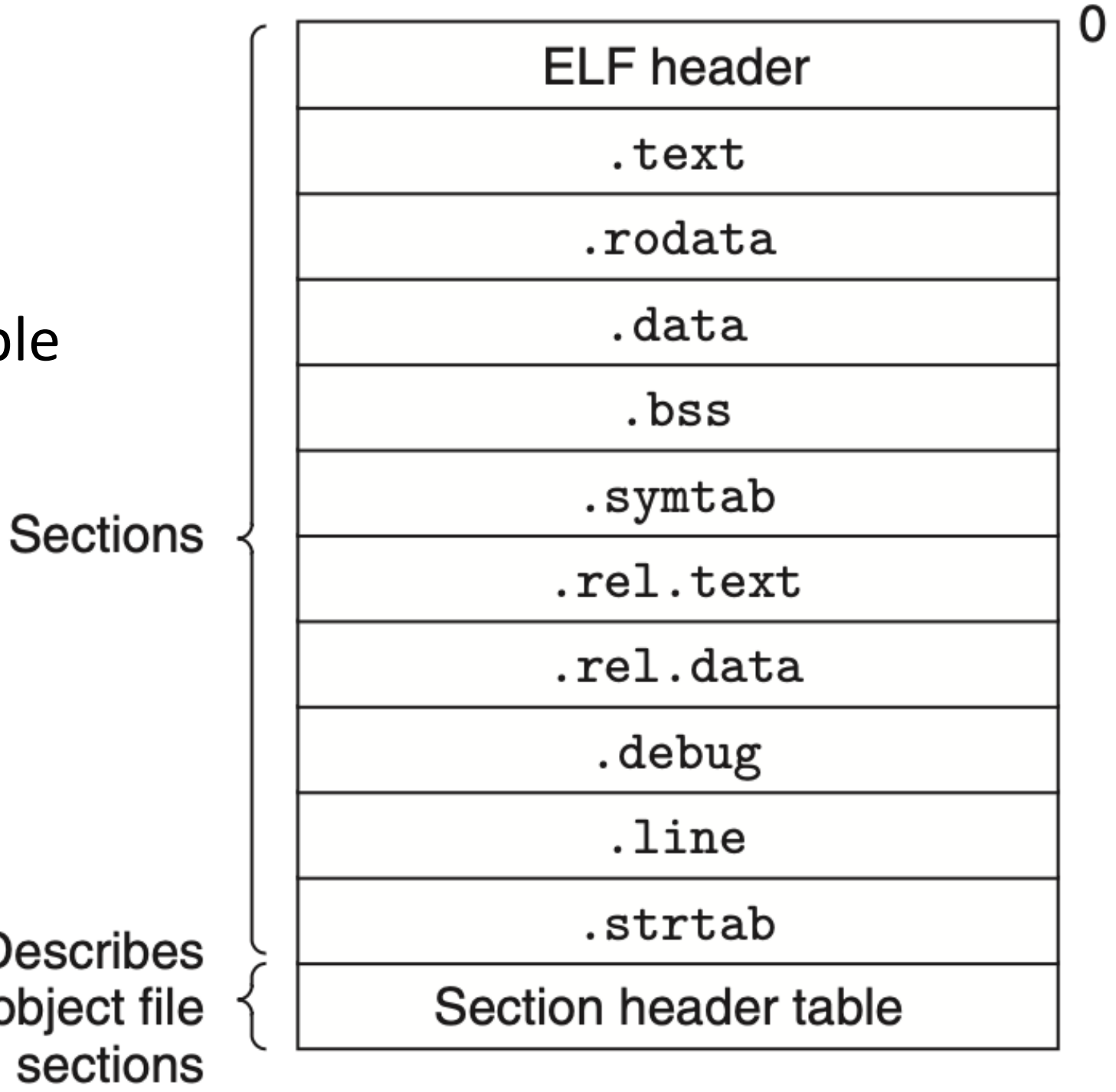
# Object Files



- Type
  - Relocatable object file.
    - Contains binary code and data in a form that can be combined with other relocatable object files at compile time to create an executable object file.
  - Executable object file.
    - Contains binary code and data in a form that can be copied directly into memory and executed.
  - Shared object file.
    - A special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run time.

# Object Files

- ELF
  - Executable and Linkable Format



# Object Files



- .text: The machine **code** of the compiled program.
- .data: Initialized global and static C **variables**.
- .bss: Uninitialized global and static C **variables**, along with any global or static variables that are initialized to zero.
- .symtab: A **symbol table** with information about functions and global variables that are defined and referenced in the program.
- .rel.text: A list of **locations** in the .text section that will need to be **modified** when the **linker** combines this object file with others.
  - Call an **external function** or reference a **global variable**
- .rel.data: **Relocation** information for any global **variables** that are referenced or defined by the module.

```
# Relocatable object program: main.o
$ as -o main.o main.s
# Disassemble
$ objdump -s -d main.o

Disassembly of section .text:
0000000000000000 <main>:
   0: 55                          push %rbp
   1: 48 89 e5                    mov  %rsp,%rbp
   4: 48 83 ec 10                 sub  $0x10,%rsp
   8: be 02 00 00 00              mov  $0x2,%esi
   d: 48 8d 05 00 00 00 00        lea  0x0(%rip),%rax #14<main+0x14>
  14: 48 89 c7                    mov  %rax,%rdi
  17: e8 00 00 00 00              call 1c <main+0x1c>
  1c: 89 45 fc                    mov  %eax,-0x4(%rbp)
  1f: 8b 45 fc                    mov  -0x4(%rbp),%eax
  22: c9                          leave
  23: c3                          ret
```

# Linker



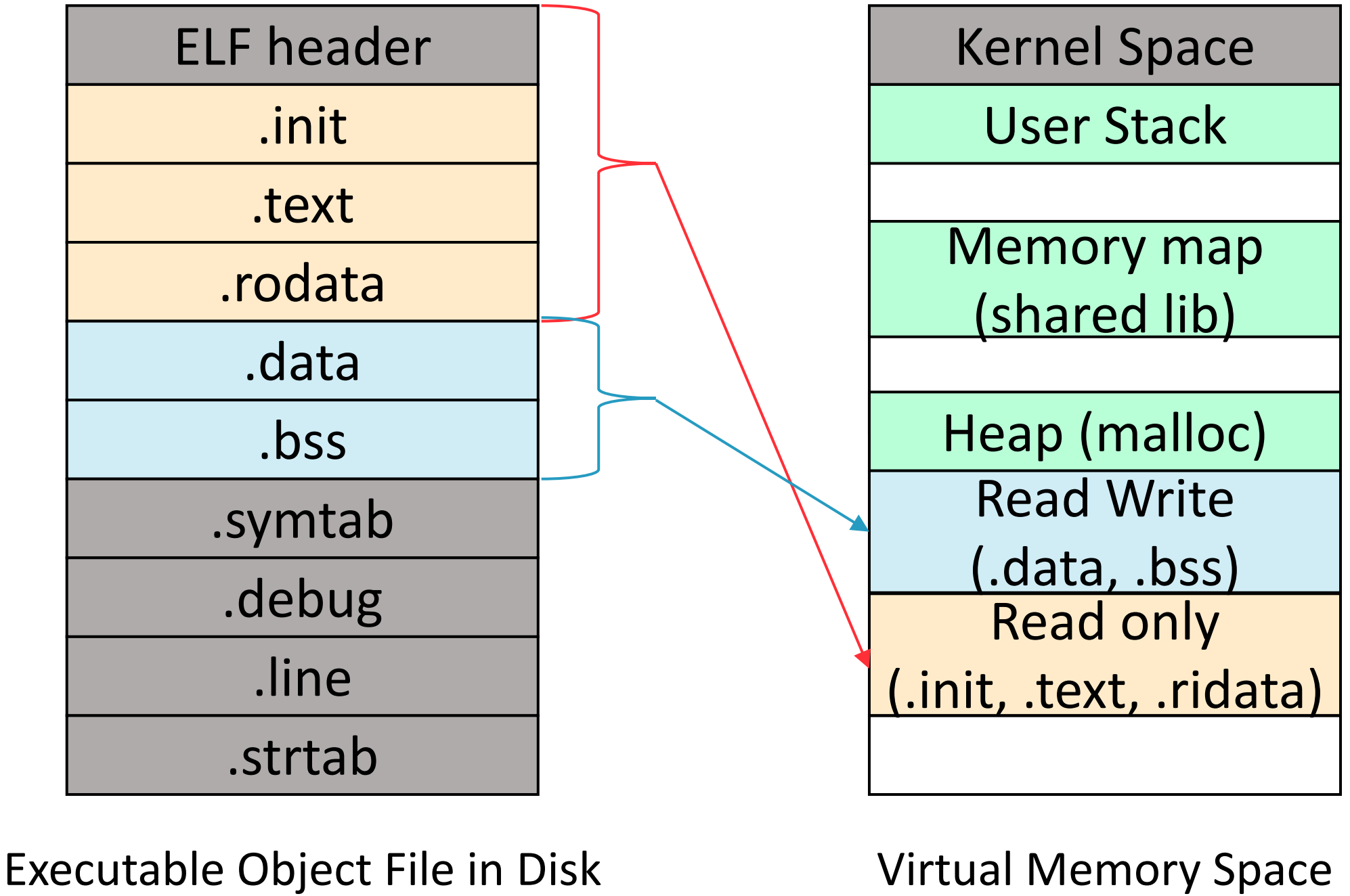
- The linker (ld)
  - Combines necessary object files (e.g., main.o and sum.o)
  - Creates the binary **executable object file** (e.g., a.out)
- Static link
  - Concatenate .text and .data section of each .o file
  - Resolve references
    - Symbol resolution
    - Relocation

```
$ ld -o a.out main.o sum.o
$ objdump -s -d a.out > out.dump

0000000000401000 <main>:
  401000: 55                              push   %rbp
  401001: 48 89 e5                        mov    %rsp,%rbp
  401004: 48 83 ec 10                     sub    $0x10,%rsp
  401008: be 02 00 00 00                  mov    $0x2,%esi
  40100d: 48 8d 05 ec 2f 00 00            lea    0x2fec(%rip),%rax
# 404000 <array>
  401014: 48 89 c7                        mov    %rax,%rdi
  401017: e8 08 00 00 00                  call   401024 <sum>
  40101c: 89 45 fc                        mov    %eax,-0x4(%rbp)
  40101f: 8b 45 fc                        mov    -0x4(%rbp),%eax
  401022: c9                              leave
  401023: c3                              ret

0000000000401024 <sum>:
```

# Loader



- The loader
  - Copies the code and data in the executable file into memory
  - Transfers control to the beginning of the program
- Resource allocation
  - Spawn new thread
  - Page table (address space)
- Program state initialization
  - Pass in `argc`, `argv` onto the stack
  - Map ELF file to into memory
  - Clear interrupt
  - Setup registers (Stack pointer)
  - Jump to program entry (Not necessarily main)


ELF header
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
Kernel Space
User Stack
Memory map
(shared lib)
Heap (malloc)
Read Write
(.data, .bss)
Read only
(.init, .text, .ridata)
Executable Object File in Disk
Virtual Memory Space

# Linking with Static Libraries



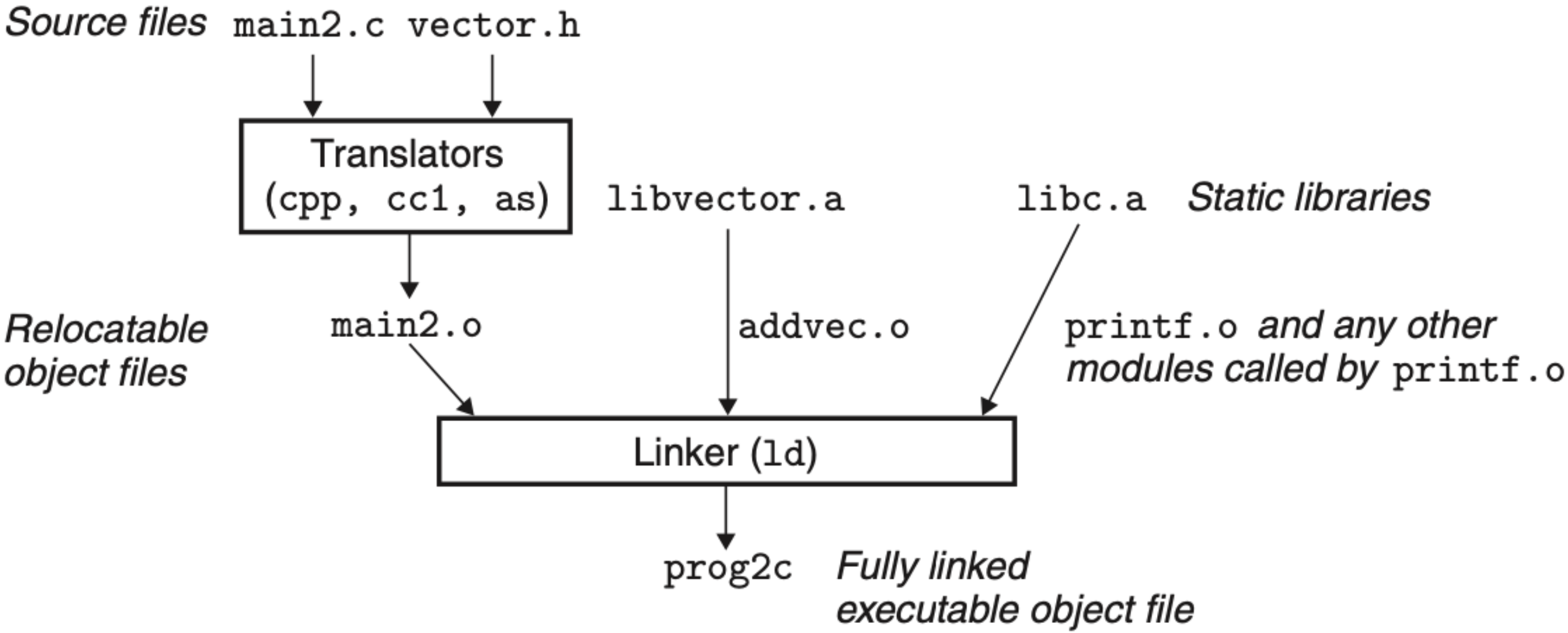
- So far,
  - linker reads a collection of relocatable object files and links them together into an output executable file
- C defines an extensive collection of standard functions
  - Compiler recognize calls to the standard functions and to generate the appropriate code directly.
  - Put all of the standard C functions in a single relocatable object module (i.e., libc.o)
    - $ gcc main.c /usr/lib/libc.o
    - waste disk and memory space
  - Create a separate relocatable file for each standard function.
    - $ gcc main.c /usr/lib/printf.o /usr/lib/scanf.o . . .
    - error prone and time consuming

# Linking with Static Libraries



- Static library
  - Related functions
    - Be compiled into separate object modules
    - Be packaged in a single static library file
  - Application programs
    - can then use any of the functions defined in the library by specifying a single filename on the command line.
    - $ gcc main.c /usr/lib/libm.a /usr/lib/libc.a
  - At link time, the linker will only copy the object modules that are referenced by the program
    - Reduce the size of the executable on disk and in memory
    - Programmer only needs to include the names of a few library files

```
# compile each
gcc -c addvec.c multvec.c
# static library: libvector.a
ar rcs libvector.a addvec.o multvec.o
# compile and link
gcc -c main2.c
gcc -static -o prog2c main2.o ./libvector.a

$ ll -h
-rw-rw-r-- 1 jiang jiang 101 2月 28 09:45 vector.h
-rw-rw-r-- 1 jiang jiang 200 2月 27 16:29 addvec.c
-rw-rw-r-- 1 jiang jiang 1.5K 2月 28 09:42 addvec.o
-rw-rw-r-- 1 jiang jiang 208 2月 27 16:29 multvec.c
-rw-rw-r-- 1 jiang jiang 1.5K 2月 28 09:42 multvec.o
-rw-rw-r-- 1 jiang jiang 3.2K 2月 28 09:42 libvector.a
-rw-rw-r-- 1 jiang jiang 235 2月 27 16:29 main2.c
-rw-rw-r-- 1 jiang jiang 1.8K 2月 28 09:45 main2.o
-rwxrwxr-x 1 jiang jiang 880K 2月 28 09:45 prog2c*
```

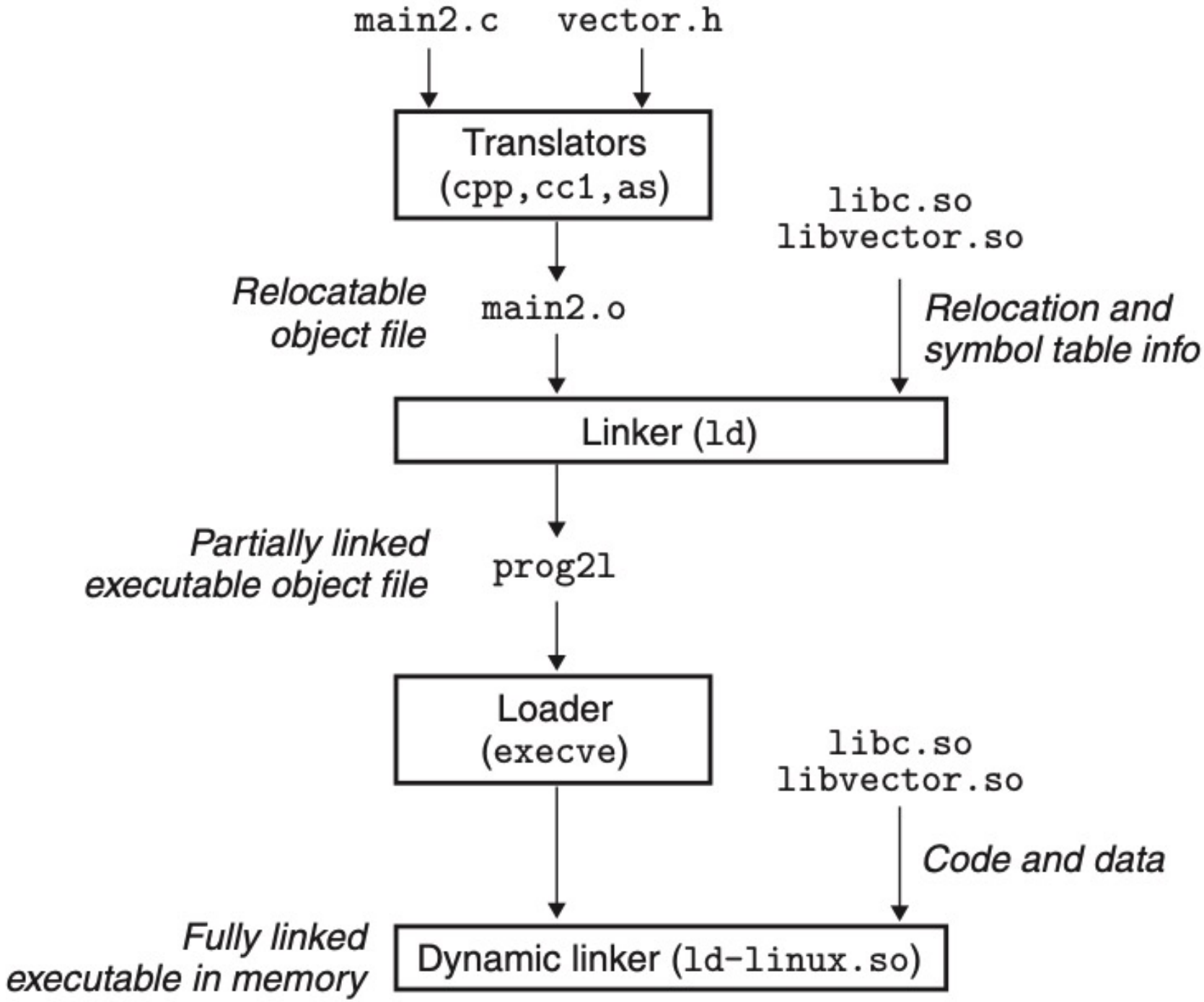

Source files
main2.c vector.h
Translators
(cpp, cc1, as)
libvector.a
libc.a
Static libraries
Relocatable
object files
main2.o
addvec.o
printf.o and any other
modules called by printf.o
Linker (ld)
prog2c
Fully linked
executable object file

# Dynamic Linking with Shared Libraries



- Disadvantages of static libraries
  - If static libraries are updated periodically
    - Application programmers have to explicitly relink their programs against the updated library
  - Waste system resources for frequently used functions
    - such as, printf and scanf
- Shared library
  - Object module
    - can be loaded at an arbitrary memory address and linked with a program in memory at either run time or load time
    - Dynamic linking
    - Dynamic linker

```
# shared libary: libvector.so
# -fpic: generate position-independent code
# -shared: create a shared object file
gcc -shared -fpic -o libvector.so addvec.c multvec.c

# compile and link
gcc -o prog2l main2.c ./libvector.so

$ ll -h
-rw-rw-r-- 1 jiang jiang 3.2K 2月 28 09:42 libvector.a
-rwxrwxr-x 1 jiang jiang 15K 2月 28 10:03 libvector.so*
-rwxrwxr-x 1 jiang jiang 880K 2月 28 09:45 prog2c*
-rwxrwxr-x 1 jiang jiang 16K 2月 28 10:03 prog2l*
```


main2.c
vector.h
Translators
(cpp,cc1,as)
libc.so
libvector.so
Relocatable
object file
main2.o
Relocation and
symbol table info
Linker (ld)
Partially linked
executable object file
prog2l
Loader
(execve)
libc.so
libvector.so
Code and data
Fully linked
executable in memory
Dynamic linker (ld-linux.so)

# Ref



- Book
  - Computer Systems: A Programmers Perspective (3rd)
    - 深入理解计算机系统
  - 程序员的自我修养 —— 链接、装载与库
- Course
  - 南京大学，袁春风：计算机系统基础
    - 程序的表示、转换与链接