计算机系统导论

第九次小班课　11/16/2017

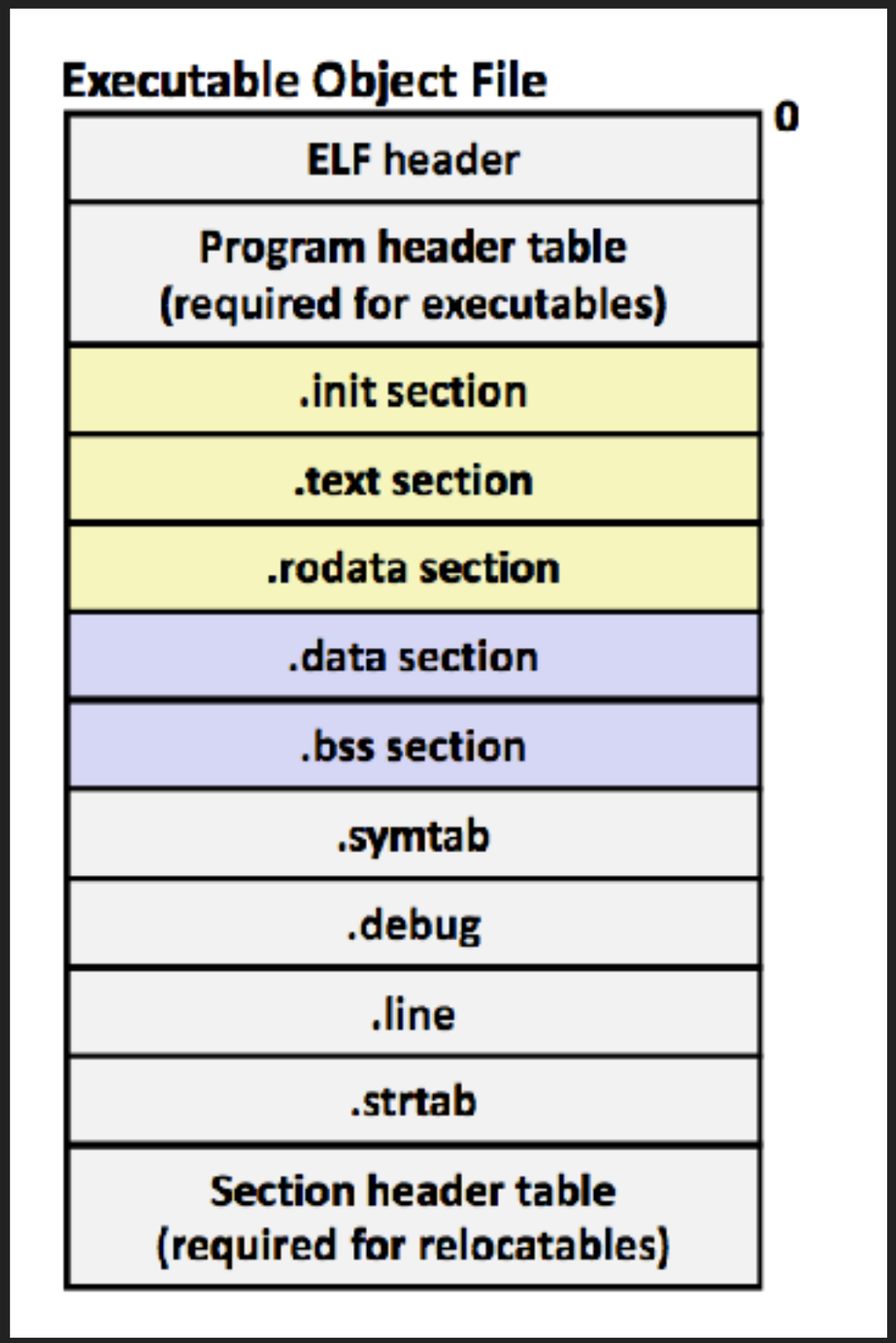
本周要点

下半学期

- 硬件 -> 软件
- 体系结构 -> 操作系统、软件
- 内容多、细节多而杂
- lab提前下手！！

链接

- 统一的可执行文件格式
- 不同数据、代码放在不同的节 (section) 中



链接

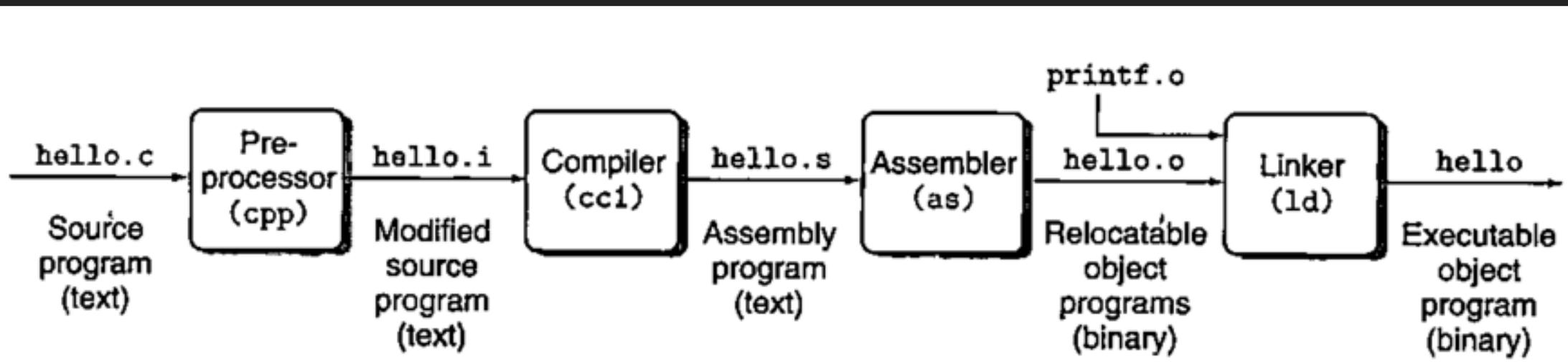
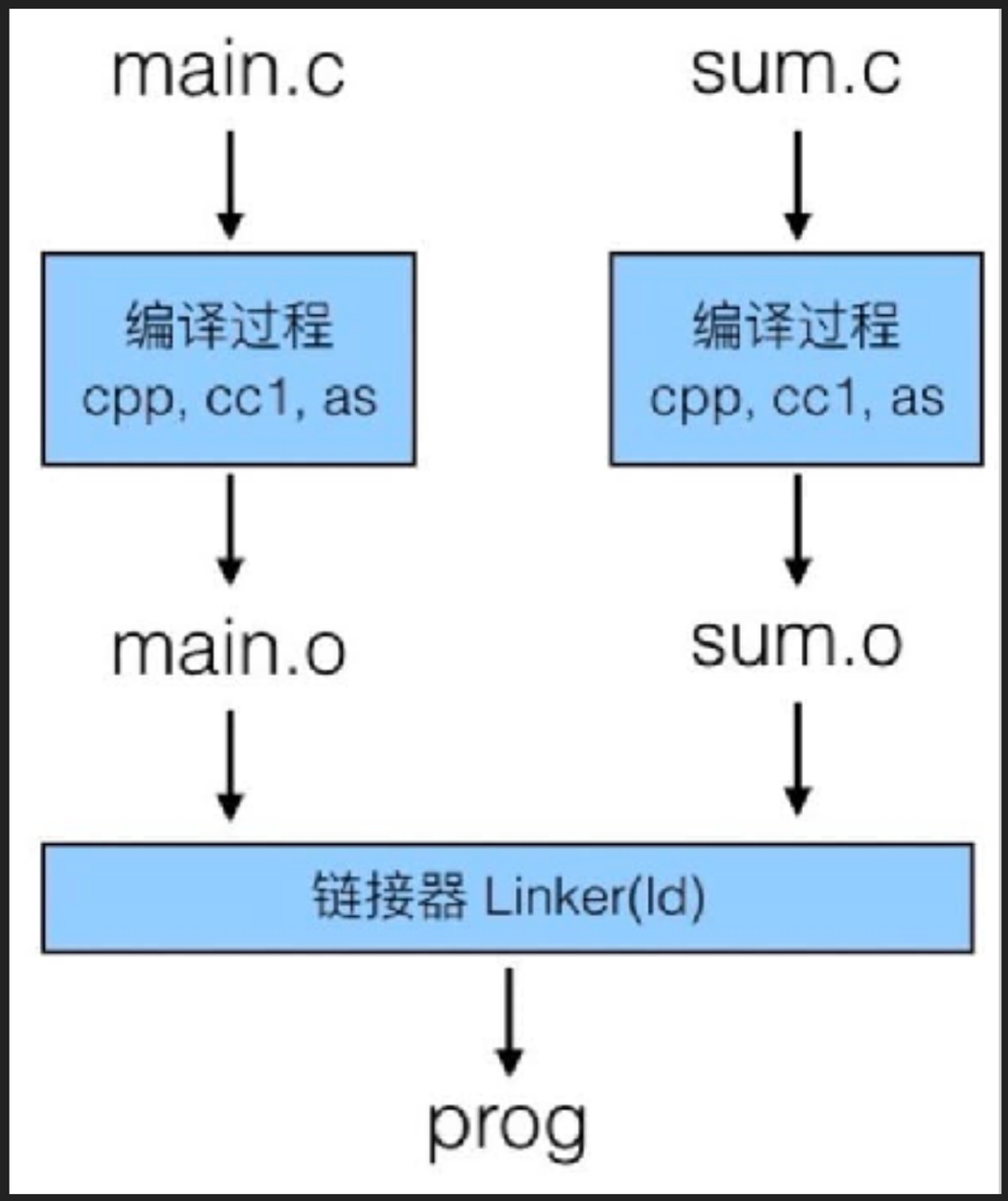


Figure 1.3 The compilation system.

链接


main.c
sum.c
编译过程
cpp, cc1, as
编译过程
cpp, cc1, as
main.o
sum.o
链接器 Linker(ld)
prog

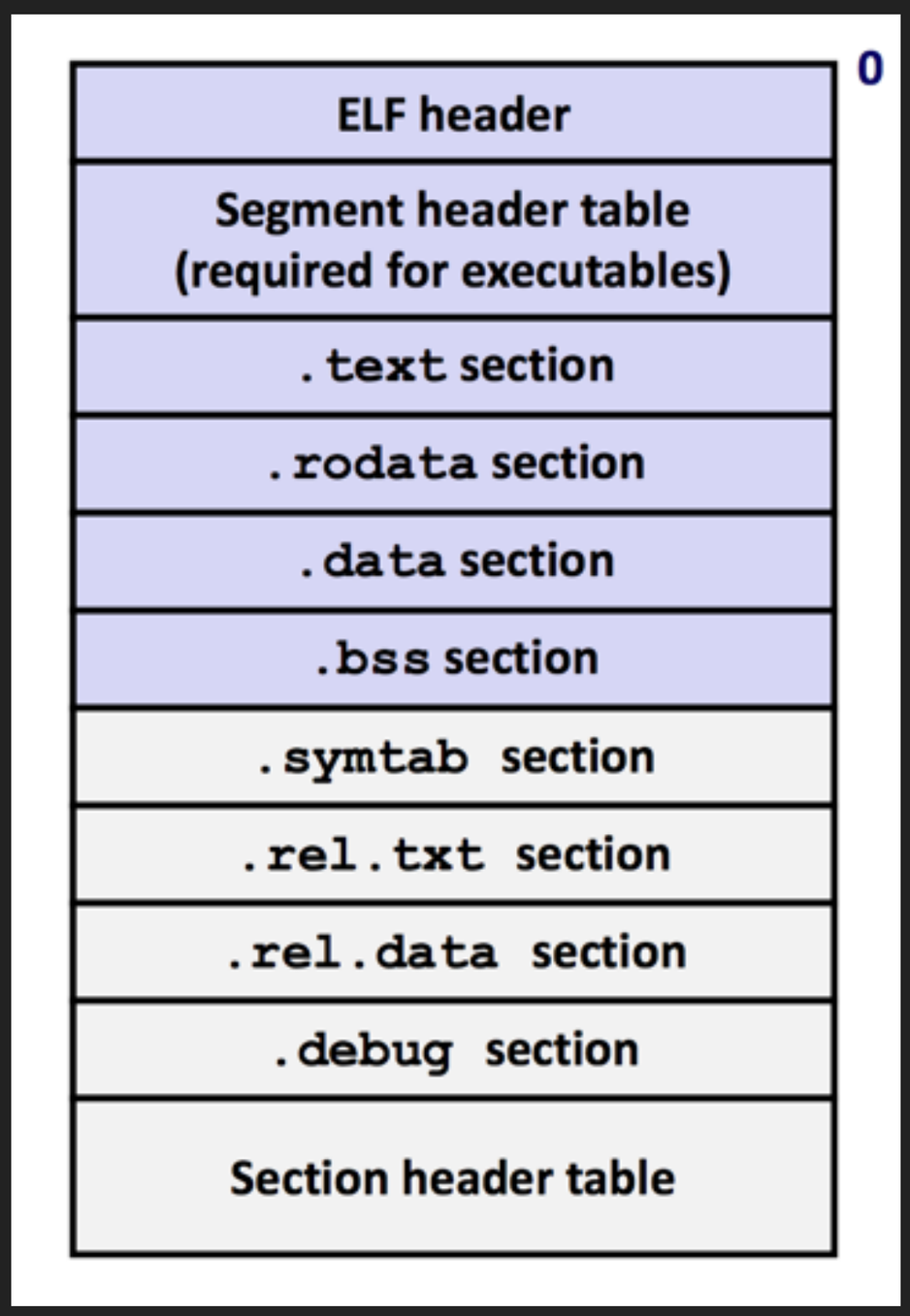
链接

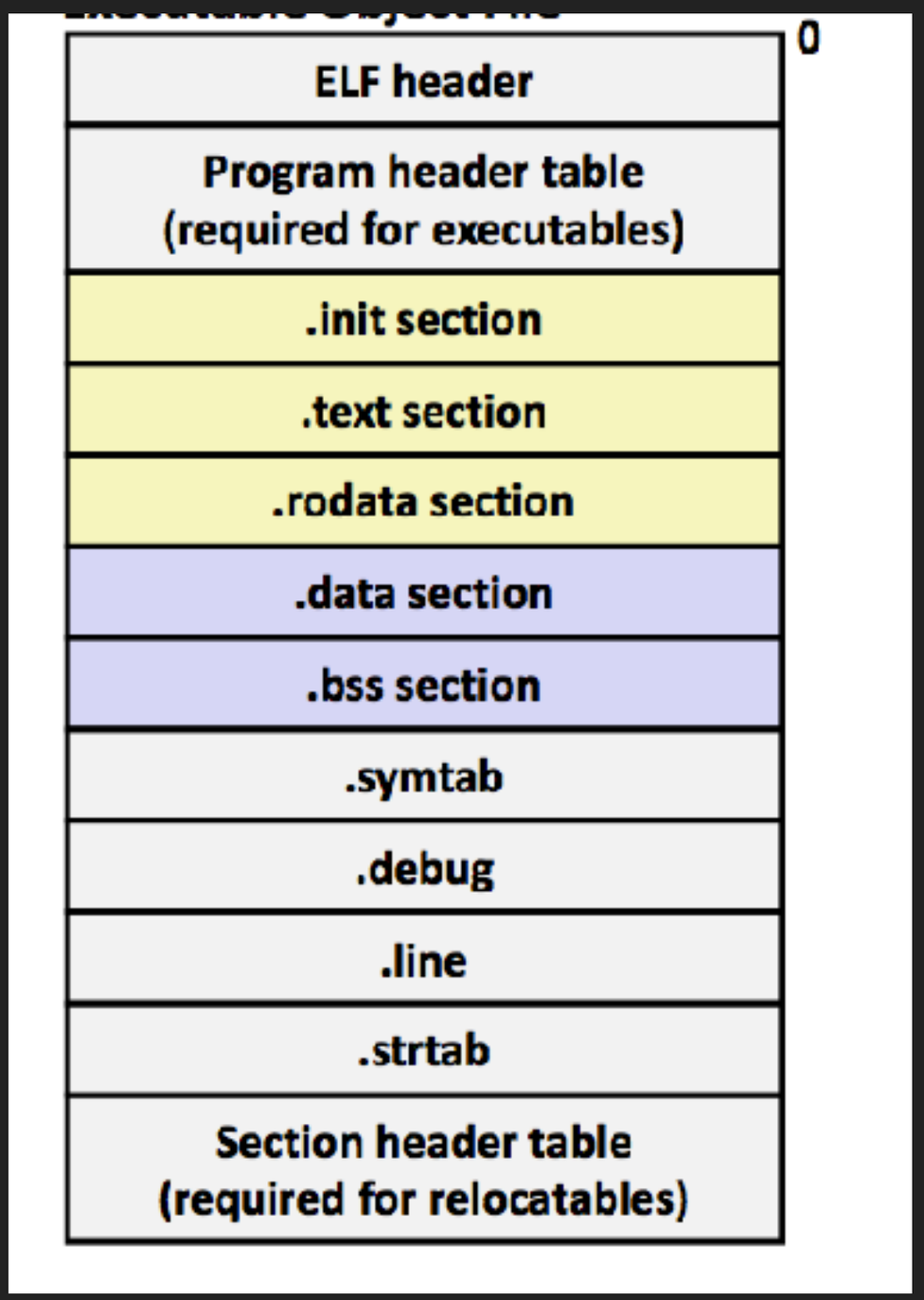
为什么要单独的链接器？让gcc在编译时就把完整的二进制代码组合在一起，是否可行?

链接

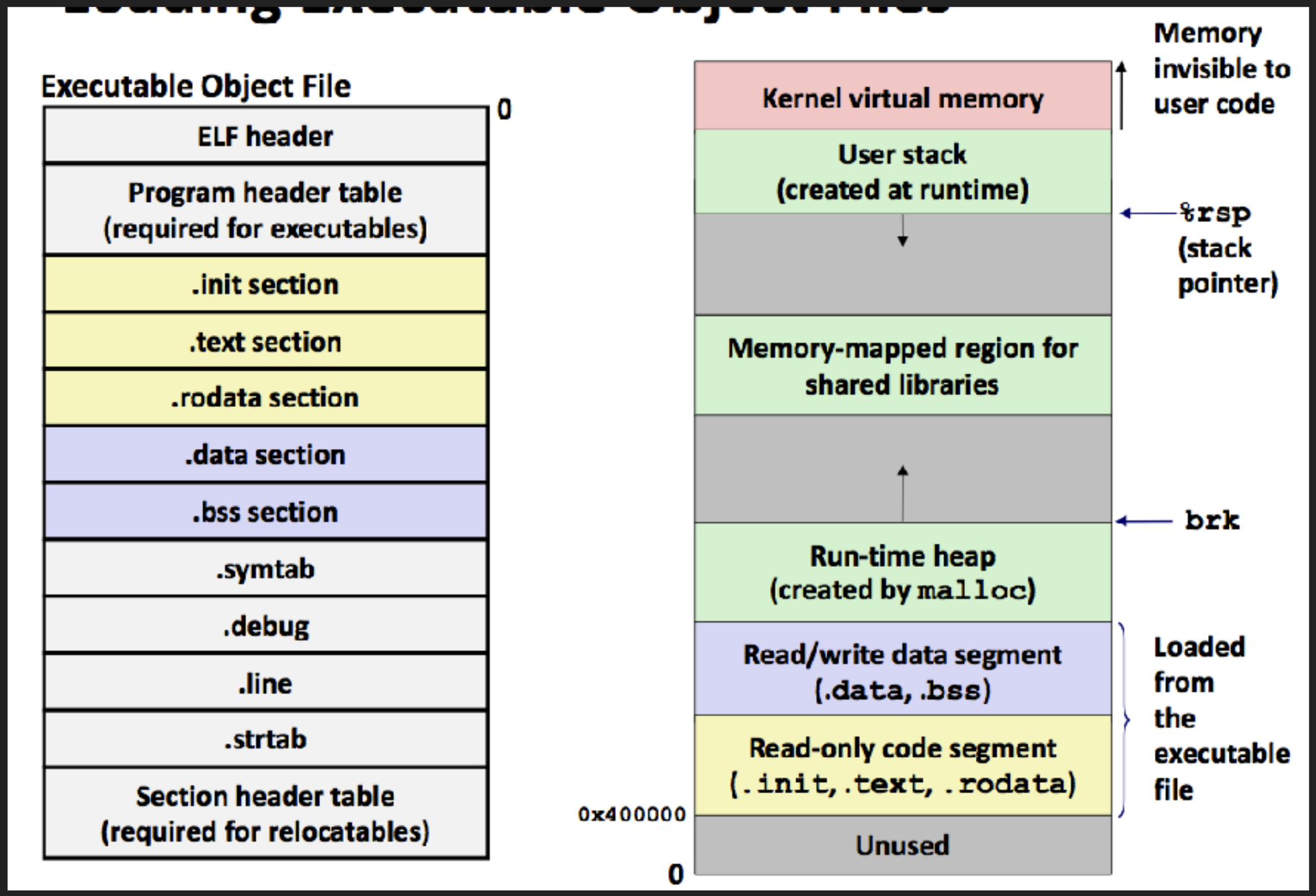
- 模块化：我们可以把程序分散到不同的小的源代码中，而不是一个巨大的类中。这样带来的好处是可以复用常见的功能/库，如 libc、libm等
- 高效率：改动代码时只需要重新编译改动的文件，其他不受影响。而常用的函数和功能可以封装成库，提供给程序进行调用（节省空间）

ELF

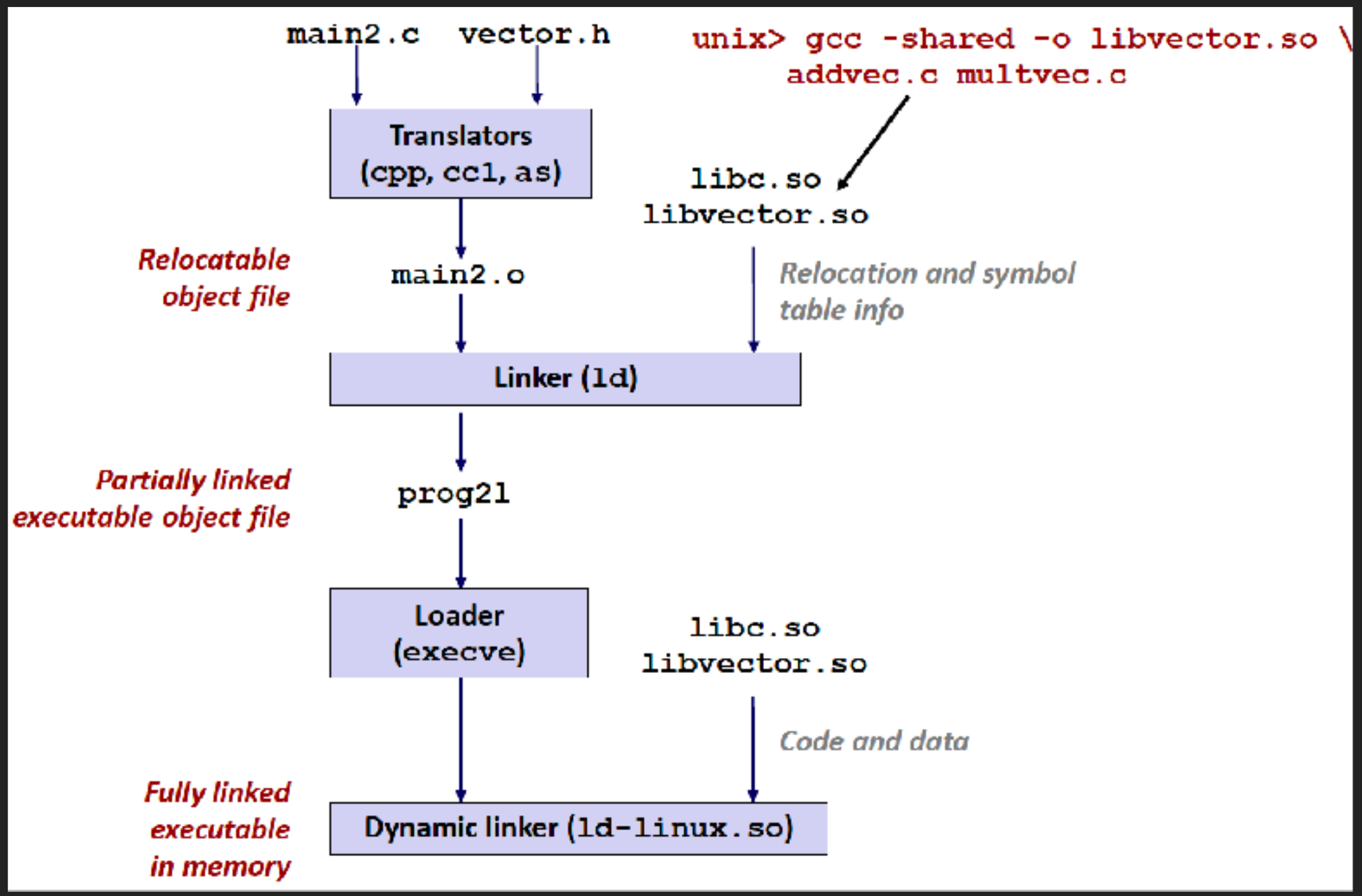

0
ELF header
Segment header table
(required for executables)
. text section
. rodata section
. data section
. bss section
. symtab section
. rel . txt section
. rel . data section
. debug section
Section header table


0
ELF header
Program header table
(required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table
(required for relocatables)

加载


Executable Object File
0
ELF header
Program header table
(required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table
(required for relocatables)
Memory
invisible to
user code
Kernel virtual memory
User stack
(created at runtime)
%rsp
(stack
pointer)
Memory-mapped region for
shared libraries
brk
Run-time heap
(created by malloc)
Read/write data segment
(.data, .bss)
Read-only code segment
(.init, .text, .rodata)
Loaded
from
the
executable
file
0x400000
Unused
0

动态链接


main2.c
vector.h
Translators
(cpp, cc1, as)
Relocatable
object file
main2.o
unix> gcc -shared -o libvector.so \
addvec.c multvec.c
libc.so
libvector.so
Relocation and symbol
table info
Linker (ld)
Partially linked
executable object file
prog21
Loader
(execve)
libc.so
libvector.so
Code and data
Fully linked
executable
in memory
Dynamic linker (ld-linux.so)

必考1 符号解析

- 全局符号、外部符号、局部符号
- 强符号、弱符号
- 符号存放在哪一节
- static
- 伪节

必考2 重定位

- 两种基本的重定位类型
- 计算题
- 重定位PC相对引用
- 重定位绝对引用

补充题

```c
#include "../inc/Fork.h"

int main()
{
    Fork();
    Fork() && Fork() || Fork();
    Fork();

    while (1);
}
```

最后共有多少个进程?

```c
#include "../inc/Fork.h"

int main(int argc, char** argv)
{
    int n = atoi(argv[1]);
    for (int i = 0; i < n; i++)
        Fork();
    printf("蛤\n");
}
```

最后共有多少个蛤?

```c
#include "../inc/Fork.h"

int main(int argc, char** argv)
{
    int n = atoi(argv[1]);
    for (int i = 0; i < n; i++)
        Fork(), printf("蛤\n");
}
```

最后共有多少个蛤?

```c
#include "../inc/Fork.h"

int main(int argc, char** argv)
{
    int n = atoi(argv[1]);
    for (int i = 0; i < n; i++)
        printf("蛤"), Fork();
}
```

最后共有多少个蛤?

本周要点

最后共有多少个蛤?

本周要点

Fork有什么缺点?