Typical Workflow of a Language Processing System

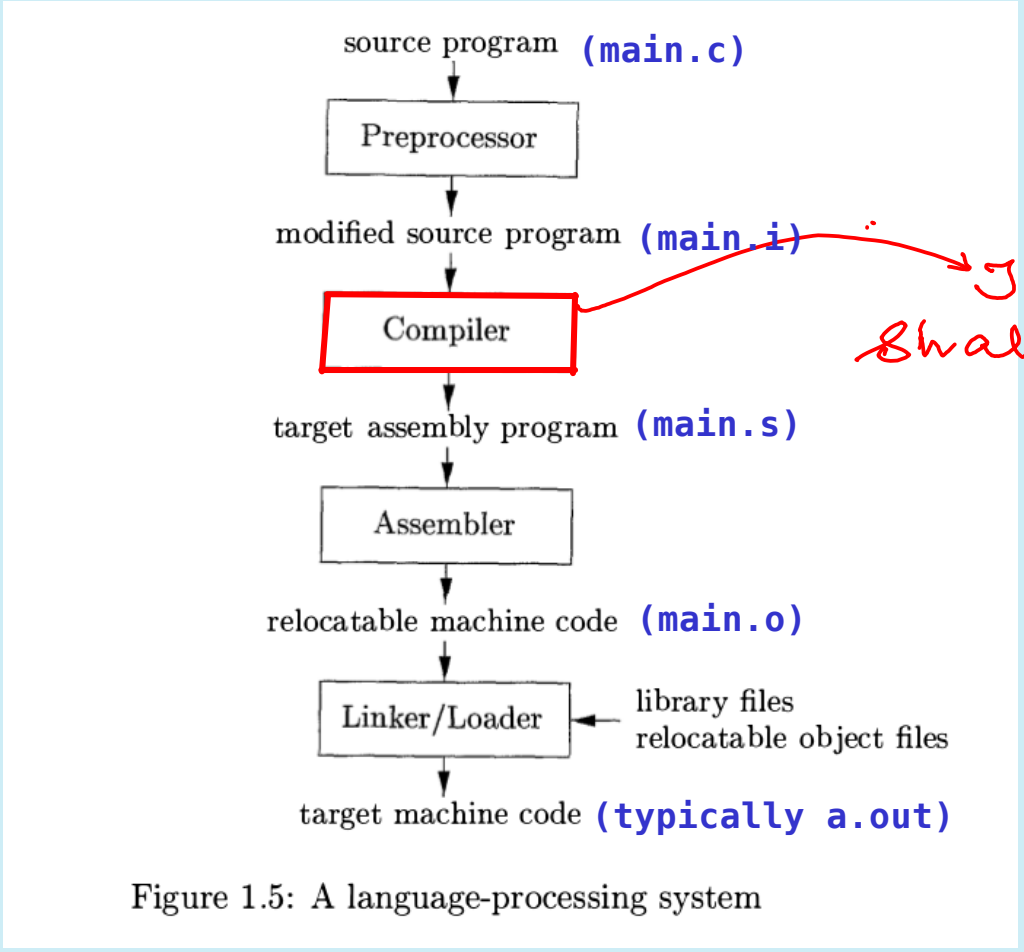


Figure 1.5: A language-processing system

This is what we shall do in the course.

Example program to illustrate compilation

main.c

```
#include <stdio.h> /*system  header files*/
extern void swap (); /*declaration*/
int buf [4] = {23, 56}; /*initialised global*/
void foo ()
{
  buf[0] = buf[1] + 1;
  buf[2] = 2;
  buf[3] = 3;
}
int main () /*definition main*/
{
  foo (); swap ();
  printf("buf[0]=%d buf[1]=%d\n", buf[0], buf[1]);
  return 0;
```

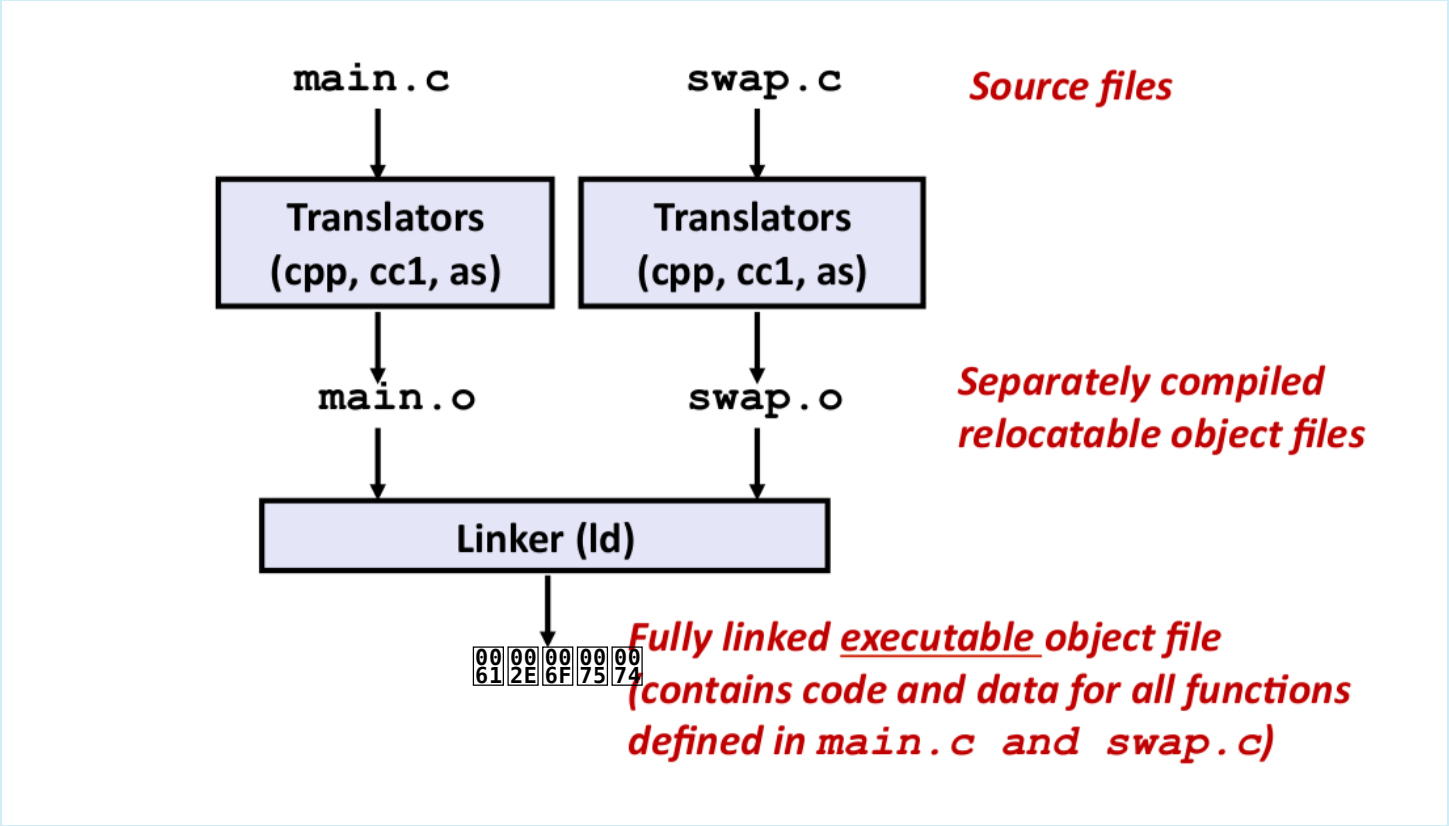
swap.c

```
#define one 1
extern int buf[];/*declaration buf*/
int *bufp0 = &buf[0];/*initialized global*/
static int *bufp1; /*uninitialized global*/

void swap () /* definition swap */
{
  int temp; /* local */
  bufp1 = &buf[one];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

- Preprocesor directives
- Declarations of function and data -
- Definitions of function and data
  - global data
    - unintialized
    - initialized
  - local data
  - read-only data

Separate compilation



Regular compilation:
gcc main.c swap.c -o main   -- produces the executable main

Most of us live with this for our entire lives.

Optimizations:

gcc -O2 main.c swap.c

O, O1 - Reduces code size and execution time without significant increase
in compilation time
O2 - Nearly all optimizations. Does not do loop unrolling and function inlining.
O3 - ALL optimizations
O0 - (or no O switch) - NO optimizations. Useful while debugging. Optimizations may interfere with debugging.
Os - Optimize for size

Read the descriptions of the optimizations from:

http://gcc.gnu.org/onlinedocs/gcc-3.1.1/gcc/Optimize-Options.htmlOptimizations:

```
Just preprocessing:   https://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html
gcc -E main.c swap.c
cpp -dN -dI main.c



Producing assembly output
gcc -S main.c swap.c          -- produces main.s and swap.s

Producing object files:
gcc -c main.c swap.c          -- produces the object files main.o
                                 and swap.o
```

Producing assembly output

```
gcc -S main.c swap.c          -- produces main.s and swap.s
```

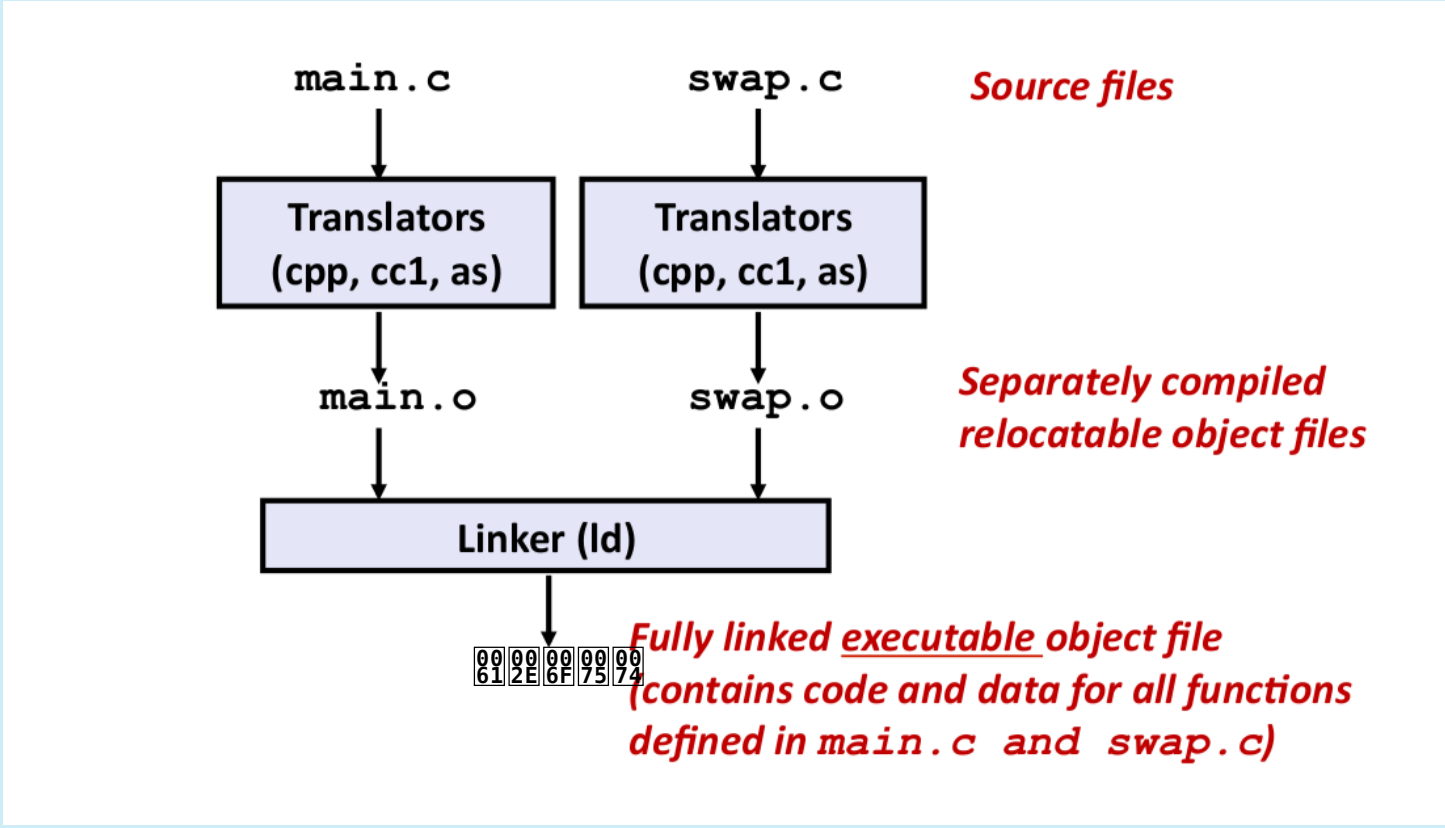
Interesting switches:

```
-fverbose-asm   -relates to source variables
-fdump-tree-gimple - gimple code
-fno-exceptions -fno-asynchronous-unwind-tables
```

Producing object files:

```
gcc -c main.c swap.c          -- produces the relocatable object files main.o
                                 and swap.o
```

Separate compilation



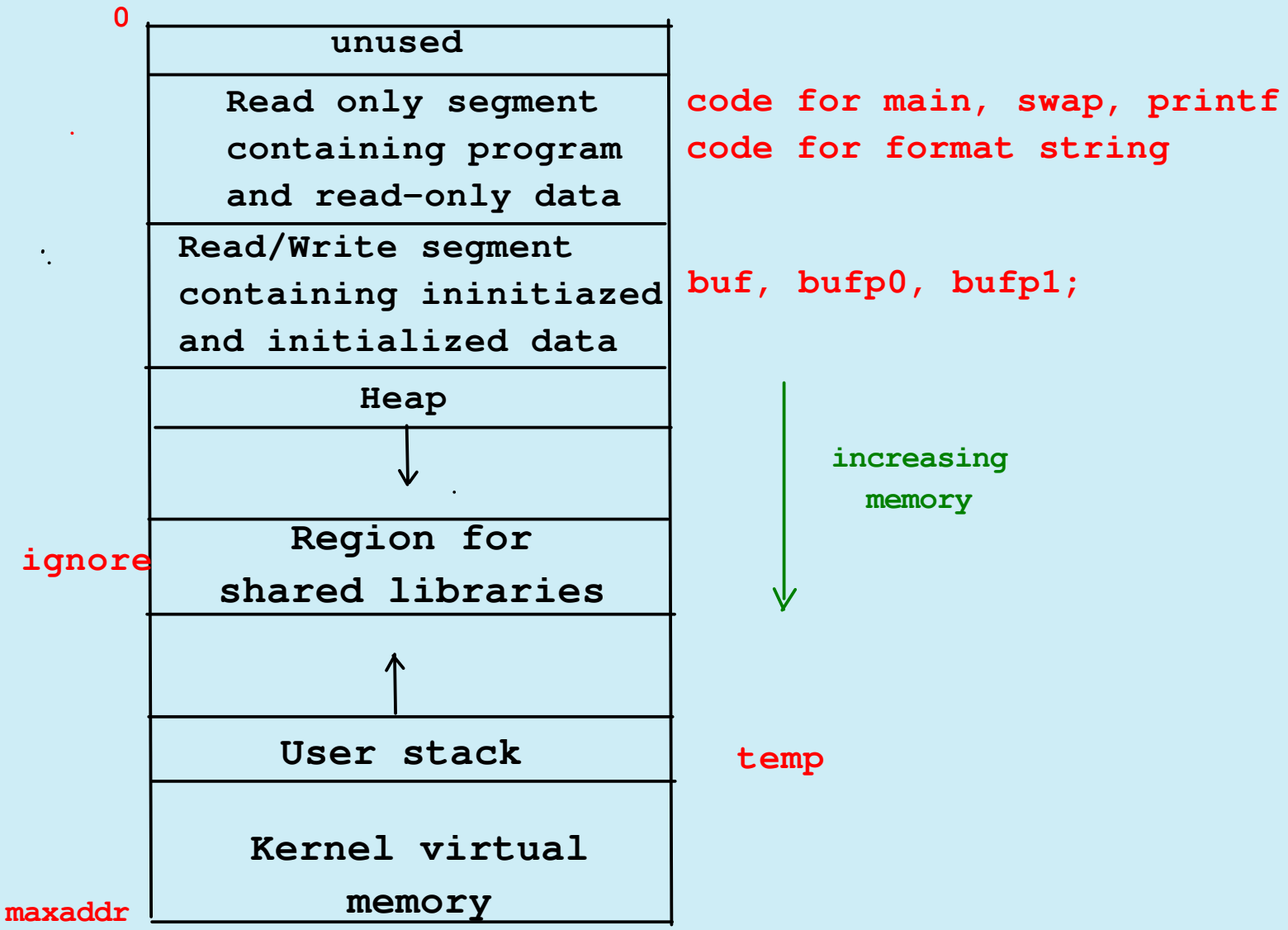
objdump -S -s x.o

-S annotates disassembled code with source statements. But the .o files must be produced with the -g option
-s Display the full contents of any sections requested. By default all non-empty sections are displayed.

readelf -W -r -s x.o

-W produce a "wide output"
-r produce rlocation information
-s produce symboltable

The memory image of a program executing in the linux environment:



How is this memory image created?

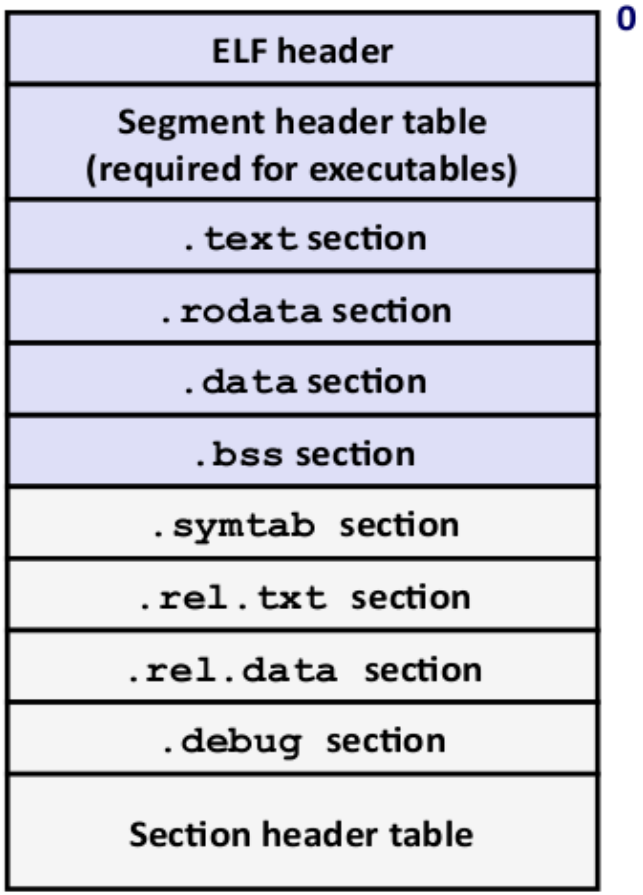
(a) The .s files are converted into .o files (relocatable object files)
- **The file format in the Linux system is called the elf format**
- **While the .o files are binary files, the information can be examined by objdump, readelf.**

(b) The .o files are linked together into the executable file (typically called a.out).
- **Also in elf format, can be read by objdump, readelf**

(c) The loader populates part of the memory image while loading. The running program creates the rest.

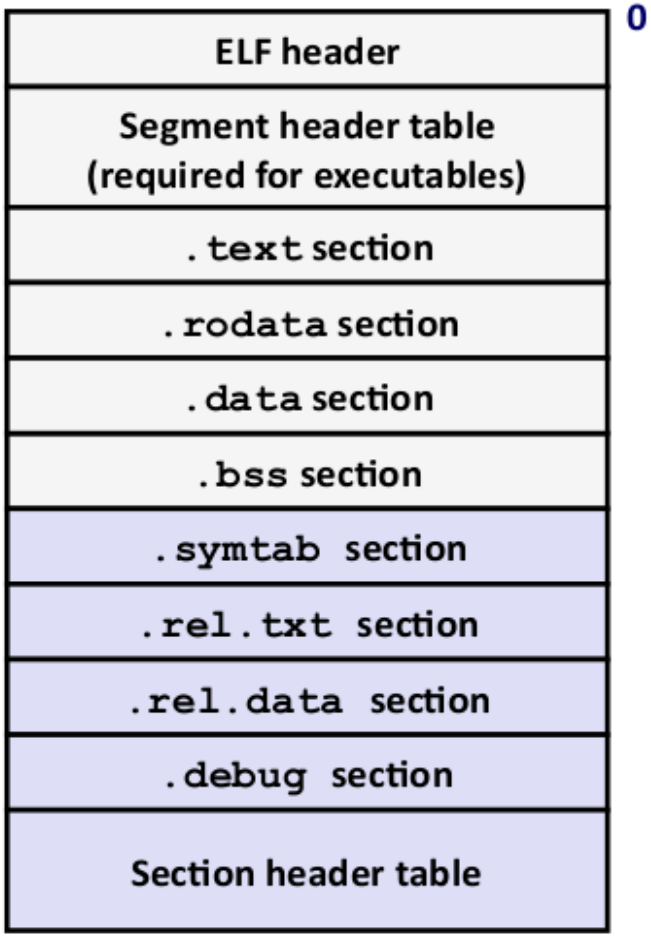
ELF Object File Format

- **Elf header**
  - Word size, byte ordering, file type (.o, exec, .so), machine type, etc.
- **Segment header table**
  - Page size, virtual addresses memory segments (sections), segment sizes.
- **`.text` section**
  - Code (disassebly and source also shown)
- **`.rodata` section**
  - Read only data: jump tables, format strings
- **`.data` section**
  - Initialized global variables (buf)
- **`.bss` section**
  - Uninitialized global variables (bufp1)
  - "Block Started by Symbol"
  - "Better Save Space"
  - Has section header but occupies no space



ELF Object File Format (cont.)

- **`.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



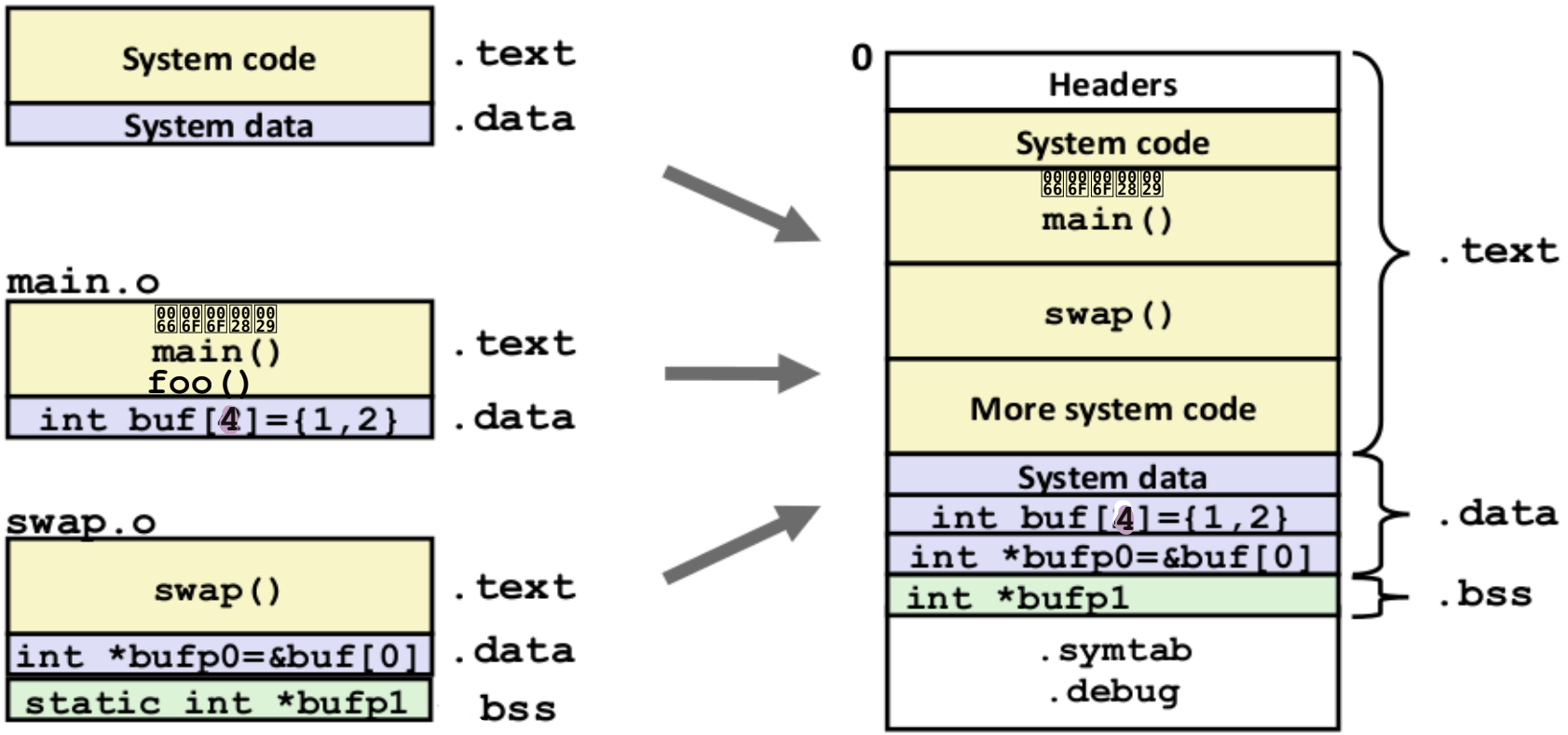
Linker Symbols

- **Global symbols**
  - Symbols defined by module *m* that can be referenced by other modules.
  - E.g.: non-`static` C functions and non-`static` global variables.

- **External symbols**
  - Global symbols that are referenced by module *m* but defined by some other module.

- **Local symbols**
  - Symbols that are defined and referenced exclusively by module *m*.
  - E.g.: C functions and variables defined with the `static` attribute.
  - **Local linker symbols are *not* local program variables**

Relocating Code and Data



The .o files are "stitched" into a single executable

The "stitching" process is called linking

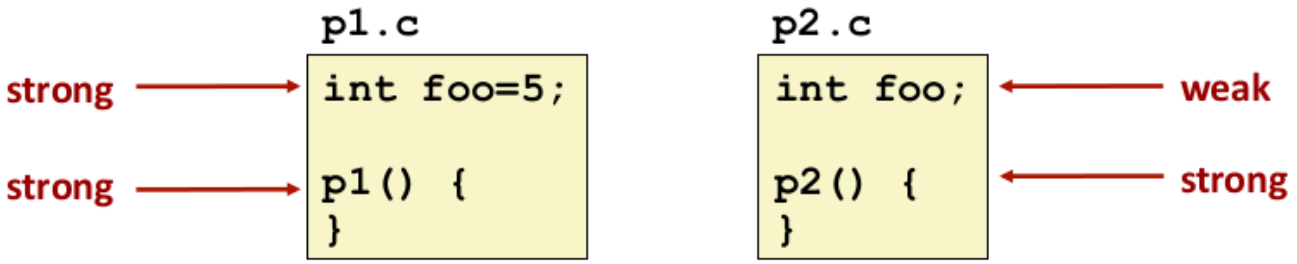
Two steps:

Symbol resolution – Matching symbol references with symbol definition

Relocation – Assigning final runtime address to each symbol

Strong and Weak Symbols

- **Program symbols are either strong or weak**
  - ***Strong***: procedures and initialized globals
  - ***Weak***: uninitialized globals



Linker Puzzles

`int x;` `p1() {}`	`p1() {}`	Link time error: two strong symbols (**p1**)
`int x;` `p1() {}`	`int x;` `p2() {}`	References to **x** will refer to the same uninitialized int. Is this what you really want?

`int x;`
`int y;`
`p1() {}`
	`double x;`
	`p2() {}`
		Writes to **x** in **p2** might overwrite **y**!
		Evil!

`int x=7;`
`int y=5;`
`p1() {}`
	`double x;`
	`p2() {}`
		Writes to **x** in **p2** will overwrite **y**!
		Nasty!

`int x=7;` `p1() {}`	`int x;` `p2() {}`	References to **x** will refer to the same initialized variable.

Relocation

```
Command: objdump -S -s main.o > main.objdump

main.o:     file format elf64-x86-64

Contents of section .data:
 0000 22000000 38000000 00000000 00000000  "...8...........
Contents of section .rodata:
 0000 6275665b 305d3d20 25642062 75665b31  buf[0]= %d buf[1
 0010 5d3d2025 640a00                      ]= %d..

Disassembly of section .text:

0000000000000000 <foo>:
void swap (); /* declaration */

int buf [4] = {34, 56}; /* initialised global */

void foo ()
{
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
  buf[0] = buf[1] + 1;
   8:   8b 05 00 00 00 00       mov    0x0(%rip),%eax
   e:   83 c0 01                add    $0x1,%eax
  11:   89 05 00 00 00 00       mov    %eax,0x0(%rip)
  buf[2]=2;
  17:   c7 05 00 00 00 00 02    movl   $0x2,0x0(%rip)
  1e:   00 00 00
  buf[3]=3;
  21:   c7 05 00 00 00 00 03    movl   $0x3,0x0(%rip)
  28:   00 00 00
}
  2b:   90                      nop
  2c:   5d                      pop    %rbp
  2d:   c3                      retq
```

Addr(reloc) + 4 + offset =

Addr(buf) + 4

offset = Addr(buf) - 0 - Addr(reloc)

offset = Address(target) - Addend - Address(reloc)

```
int main ()  /* definition main */
{
  2e:   f3 0f 1e fa             endbr64
  32:   55                      push   %rbp
  33:   48 89 e5                mov    %rsp,%rbp
  foo ();
  36:   b8 00 00 00 00          mov    $0x0,%eax
  3b:   e8 00 00 00 00          callq  40 <main+0x12>
  swap ();
  40:   b8 00 00 00 00          mov    $0x0,%eax
  45:   e8 00 00 00 00          callq  4a <main+0x1c>
  printf("buf[0]= %d buf[1]= %d\n", buf[0], buf[1]);
  4a:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 50 <main+0x22>
  50:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 56 <main+0x28>
  56:   89 c6                   mov    %eax,%esi
  58:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 5f <main+0x31>
  5f:   b8 00 00 00 00          mov    $0x0,%eax
  64:   e8 00 00 00 00          callq  69 <main+0x3b>
  return 0;
  69:   b8 00 00 00 00          mov    $0x0,%eax
}
  6e:   5d                      pop    %rbp
  6f:   c3                      retq
```

```
Command:
readelf -W -r -s main_executable > main_executable.readelf

Relocation section '.rela.text' at offset 0xc60 contains 10 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name + Addend
000000000000000a  0000000f00000002 R_X86_64_PC32          0000000000000000 buf + 0
0000000000000013  0000000f00000002 R_X86_64_PC32          0000000000000000 buf - 4
0000000000000019  0000000f00000002 R_X86_64_PC32          0000000000000000 buf + 0
0000000000000023  0000000f00000002 R_X86_64_PC32          0000000000000000 buf + 4
000000000000003c  0000001000000004 R_X86_64_PLT32         0000000000000000 foo - 4
0000000000000046  0000001300000004 R_X86_64_PLT32         0000000000000000 swap - 4
000000000000004c  0000000f00000002 R_X86_64_PC32          0000000000000000 buf + 0
0000000000000052  0000000f00000002 R_X86_64_PC32          0000000000000000 buf - 4
000000000000005b  0000000500000002 R_X86_64_PC32          0000000000000000 .rodata - 4
0000000000000065  0000001400000004 R_X86_64_PLT32         0000000000000000 printf - 4

Symbol table '.symtab' contains 21 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
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    9
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT   13
    11: 0000000000000000     0 SECTION LOCAL  DEFAULT   15
    12: 0000000000000000     0 SECTION LOCAL  DEFAULT   16
    13: 0000000000000000     0 SECTION LOCAL  DEFAULT   17
    14: 0000000000000000     0 SECTION LOCAL  DEFAULT   14
    15: 0000000000000000    16 OBJECT  GLOBAL DEFAULT    3 buf
    16: 0000000000000000    46 FUNC    GLOBAL DEFAULT    1 foo
    17: 000000000000002e    66 FUNC    GLOBAL DEFAULT    1 main
    18: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    19: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND swap
    20: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

```
swap.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <swap>:

int *bufp0 = &buf[0]; /* initialized global */
int *bufp1;   /* uninitialized global */

void swap ()   /* definition swap */
{
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
  int temp;    /* local */
  bufp1 = &buf[one];
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <swap+0xf>
   f:   48 89 05 00 00 00 00    mov    %rax,0x0(%rip)        # 16 <swap+0x16>
  temp = *bufp0;
  16:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 1d <swap+0x1d>
  1d:   8b 00                   mov    (%rax),%eax
  1f:   89 45 fc                mov    %eax,-0x4(%rbp)
  *bufp0 = *bufp1;
  22:   48 8b 15 00 00 00 00    mov    0x0(%rip),%rdx        # 29 <swap+0x29>
  29:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 30 <swap+0x30>
  30:   8b 12                   mov    (%rdx),%edx
  32:   89 10                   mov    %edx,(%rax)
  *bufp1 = temp;
  34:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 3b <swap+0x3b>
  3b:   8b 55 fc                mov    -0x4(%rbp),%edx
  3e:   89 10                   mov    %edx,(%rax)
}
  40:   90                      nop
  41:   5d                      pop    %rbp
  42:   c3                      retq
```

```
Relocation section '.rela.text' at offset 0x5c0 contains 6 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name + Addend
000000000000000b  0000001000000002 R_X86_64_PC32          0000000000000000 buf + 0
0000000000000012  0000001100000002 R_X86_64_PC32          0000000000000008 bufp1 - 4
0000000000000019  0000000f00000002 R_X86_64_PC32          0000000000000000 bufp0 - 4
0000000000000025  0000001100000002 R_X86_64_PC32          0000000000000008 bufp1 - 4
000000000000002c  0000000f00000002 R_X86_64_PC32          0000000000000000 bufp0 - 4
0000000000000037  0000001100000002 R_X86_64_PC32          0000000000000008 bufp1 - 4

Relocation section '.rela.data.rel' at offset 0x650 contains 1 entry:
    Offset             Info             Type               Symbol's Value  Symbol's Name + Addend
0000000000000000  0000001000000001 R_X86_64_64            0000000000000000 buf + 0


Symbol table '.symtab' contains 19 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS swap.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   12
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT   14
    11: 0000000000000000     0 SECTION LOCAL  DEFAULT   16
    12: 0000000000000000     0 SECTION LOCAL  DEFAULT   17
    13: 0000000000000000     0 SECTION LOCAL  DEFAULT   18
    14: 0000000000000000     0 SECTION LOCAL  DEFAULT   15
    15: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    5 bufp0
    16: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
    17: 0000000000000008     8 OBJECT  GLOBAL DEFAULT  COM bufp1
    18: 0000000000000000    67 FUNC    GLOBAL DEFAULT    1 swap
```

```
main_executable:     file format elf64-x86-64

Contents of section .data:
 4c00e0 00000000 00000000 00000000 00000000  ................

buf:
4c00f0 22000000 38000000 00000000 00000000  "...8...........
bufp0:
4c0100 f0004c00 00000000 00080000 00000000  ..L.............
bufp1:
4c3320 Not present in elf file

0000000000401cb5 <foo>:
void swap (); /* declaration */
int buf [4] = {34, 56}; /* initialised global */

void foo ()
{
  401cb5:       f3 0f 1e fa             endbr64
  401cb9:       55                      push   %rbp
  401cba:       48 89 e5                mov    %rsp,%rbp
  buf[0] = buf[1] + 1;
  401cbd:       8b 05 31 e4 0b 00       mov    0xbe431(%rip),%eax        # 4c00f4 <buf+0x4>
  401cc3:       83 c0 01                add    $0x1,%eax
  401cc6:       89 05 24 e4 0b 00       mov    %eax,0xbe424(%rip)        # 4c00f0 <buf>
  buf[2]=2;
  401ccc:       c7 05 22 e4 0b 00 02    movl   $0x2,0xbe422(%rip)        # 4c00f8 <buf+0x8>
  401cd3:       00 00 00
  buf[3]=3;
  401cd6:       c7 05 1c e4 0b 00 03    movl   $0x3,0xbe41c(%rip)        # 4c00fc <buf+0xc>
  401cdd:       00 00 00
}
```

```
void swap ()   /* definition swap */
{
  401d25:       f3 0f 1e fa             endbr64
  401d29:       55                      push   %rbp
  401d2a:       48 89 e5                mov    %rsp,%rbp
  int temp;    /* local */
  bufp1 = &buf[one];
  401d2d:       48 8d 05 c0 e3 0b 00    lea    0xbe3c0(%rip),%rax        # 4c00f4 <buf+0x4>
  401d34:       48 89 05 e5 15 0c 00    mov    %rax,0xc15e5(%rip)        # 4c3320 <bufp1>
  temp = *bufp0;
  401d3b:       48 8b 05 be e3 0b 00    mov    0xbe3be(%rip),%rax        # 4c0100 <bufp0>
  401d42:       8b 00                   mov    (%rax),%eax
  401d44:       89 45 fc                mov    %eax,-0x4(%rbp)
  *bufp0 = *bufp1;
  401d47:       48 8b 15 d2 15 0c 00    mov    0xc15d2(%rip),%rdx        # 4c3320 <bufp1>
  401d4e:       48 8b 05 ab e3 0b 00    mov    0xbe3ab(%rip),%rax        # 4c0100 <bufp0>
  401d55:       8b 12                   mov    (%rdx),%edx
  401d57:       89 10                   mov    %edx,(%rax)
  *bufp1 = temp;
  401d59:       48 8b 05 c0 15 0c 00    mov    0xc15c0(%rip),%rax        # 4c3320 <bufp1>
  401d60:       8b 55 fc                mov    -0x4(%rbp),%edx
  401d63:       89 10                   mov    %edx,(%rax)
}
```

Packaging Commonly Used Functions

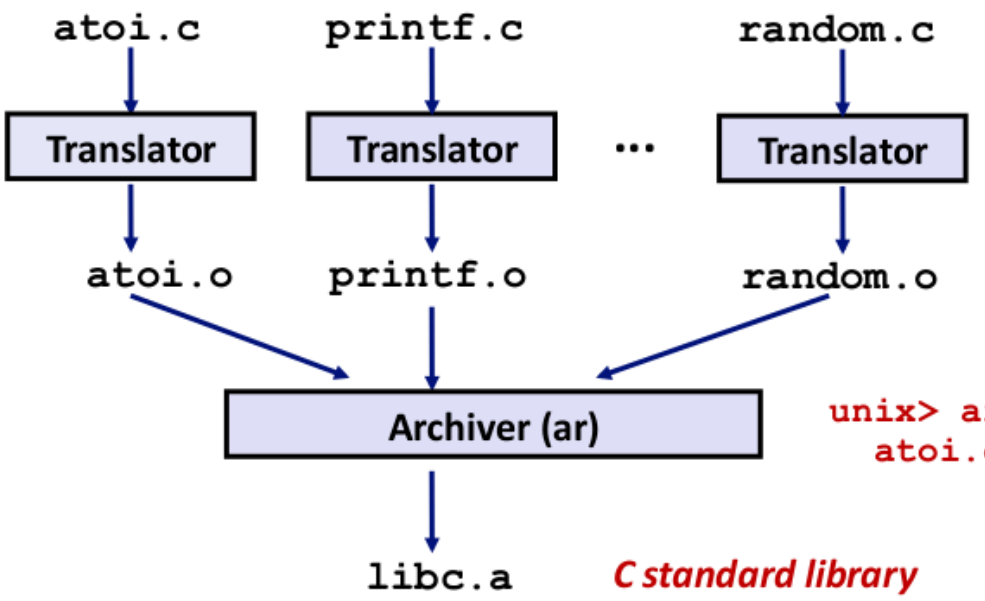
- **How to package functions commonly used by programmers?**
  - Math, I/O, memory management, string manipulation, etc.

- **Awkward, given the linker framework so far:**
  - **Option 1:** Put all functions into a single source file
    - Programmers link big object file into their programs
    - Space and time inefficient
  - **Option 2:** Put each function in a separate source file
    - Programmers explicitly link appropriate binaries into their programs
    - More efficient, but burdensome on the programmer

Solution: Static Libraries

- **Static libraries (.a archive files)**
  - Concatenate related relocatable object files into a single file with an index (called an *archive*).
  - Enhance linker so that it tries to resolve unresolved external references by looking for the symbols in one or more archives.
  - If an archive member file resolves reference, link it into the executable.

Creating Static Libraries




```
unix> ar rs libc.a \
  atoi.o printf.o … random.o
```

- Archiver allows incremental updates
- Recompile function that changes and replace .o file in archive.

Commonly Used Libraries

`libc.a` (the C standard library)

- 8 MB archive of 1392 object files.
- I/O, memory allocation, signal handling, string handling, data and time, random numbers, integer math

`libm.a` (the C math library)

- 1 MB archive of 401 object files.
- floating point math (sin, cos, tan, log, exp, sqrt, ...)

```
% ar -t /usr/lib/libc.a | sort
...
fork.o
...
fprintf.o
fpu_control.o
fputc.o
freopen.o
fscanf.o
fseek.o
fstab.o
...
```

```
% ar -t /usr/lib/libm.a | sort
...
e_acos.o
e_acosf.o
e_acosh.o
e_acoshf.o
e_acoshl.o
e_acosl.o
e_asin.o
e_asinf.o
e_asinl.o
...
```

Linking with Static Libraries