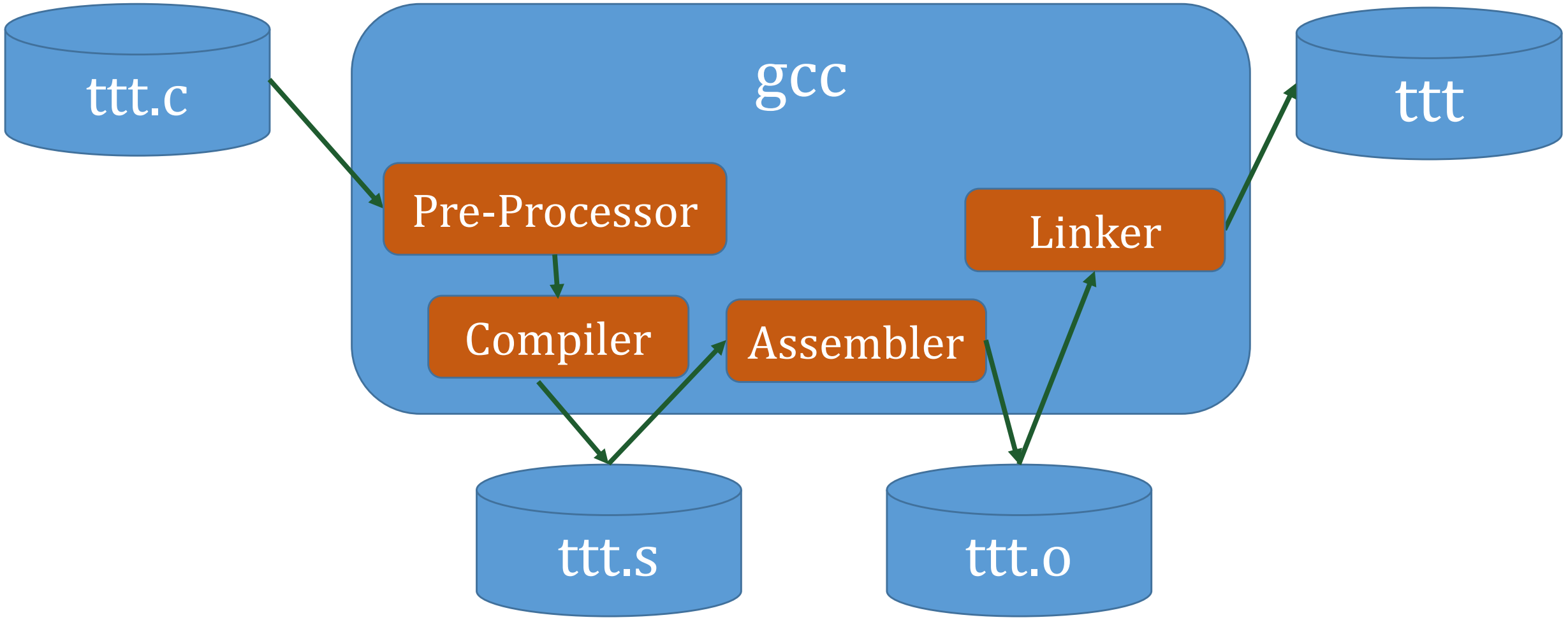
Loading Code

Computer Systems Chapter 7.5, 7.8, 7.9

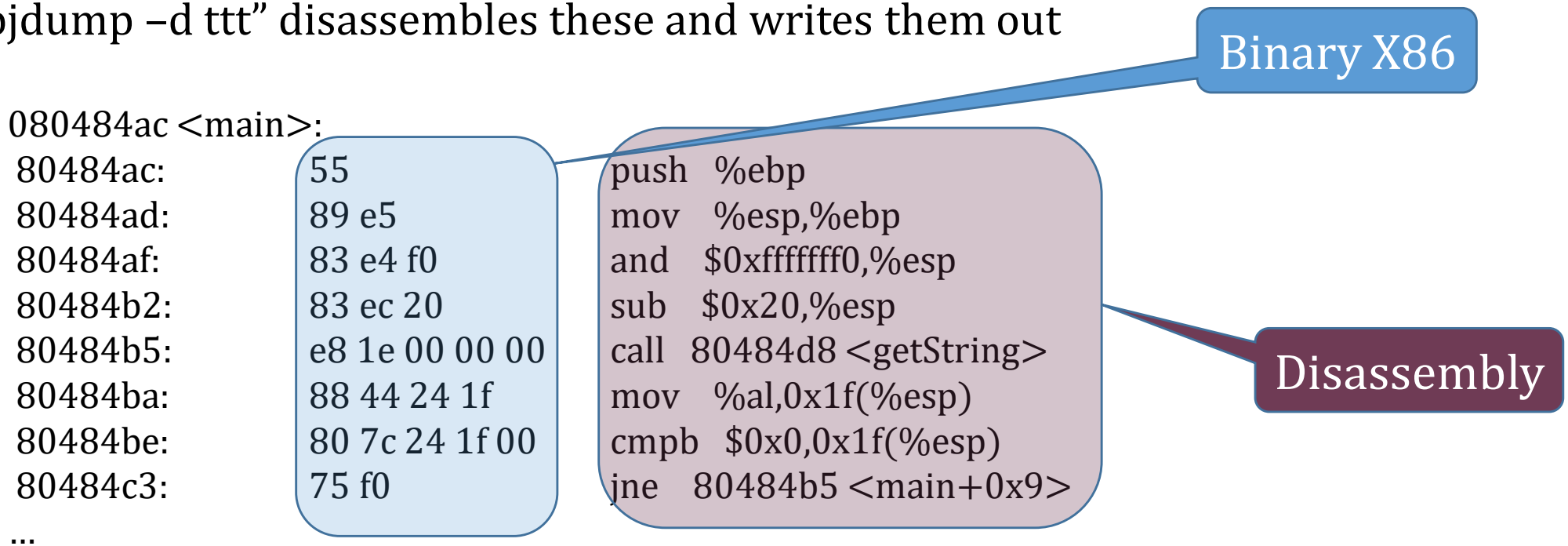
gcc –g –o ttt ttt.c

gcc –g –o ttt ttt.c



What is in a binary executable file?

- Binary representation of X86 instructions
  - "objdump –d ttt" disassembles these and writes them out



- What else is in the binary executable file??

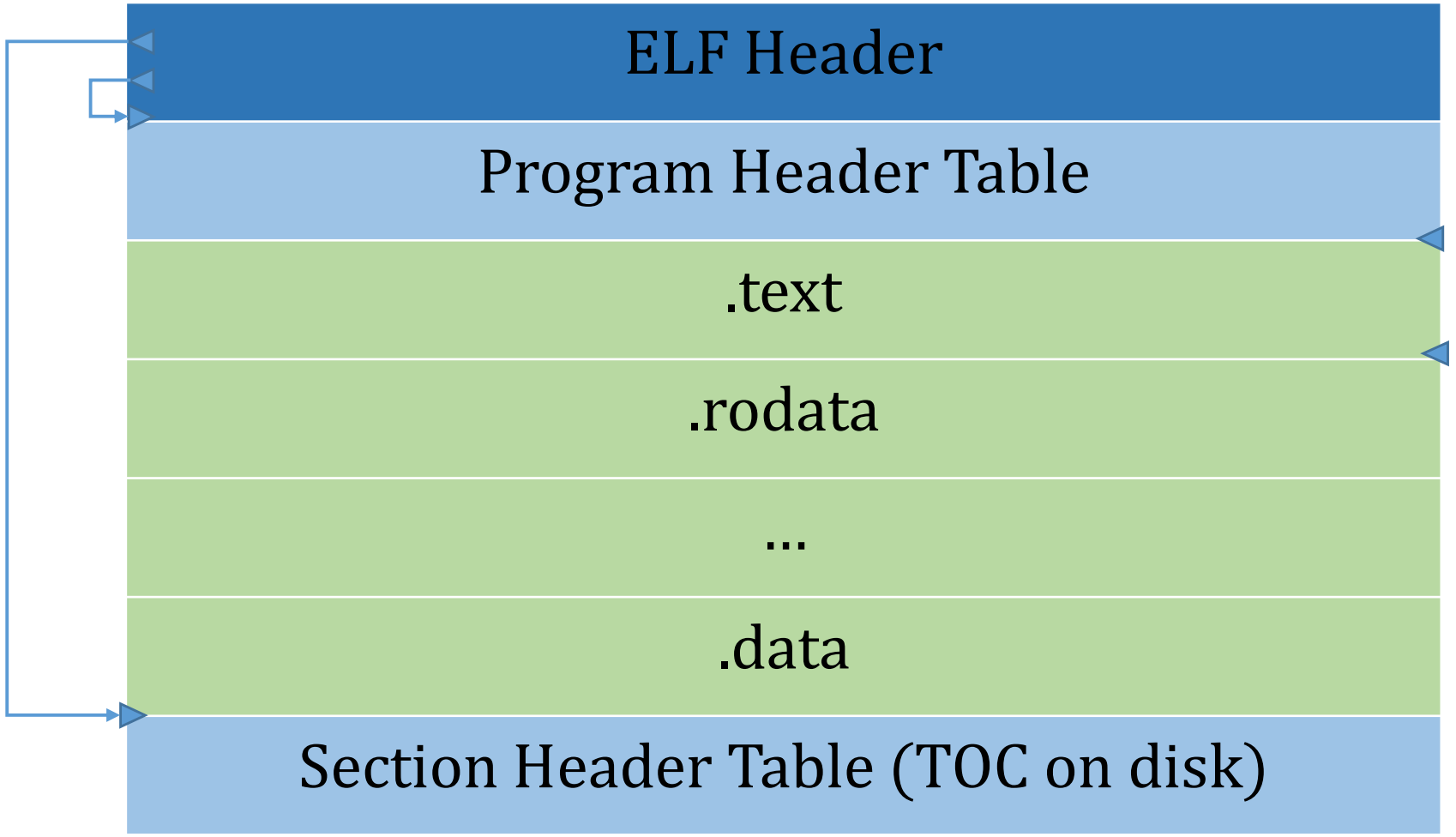
What else is in a binary file?

- Information about WHERE in memory the code is placed
- Cross-reference between function name and location in memory
- Information about constants
  - Some constants can be literal values in X86 instructions… $12
  - Not all constants fit in instructions… “Enter your next move”
  - Binary file must contain both the value and location of constants
- Information about global variables
  - Where each global variable exists in memory
  - What the initial value of the global variable is (if initialized)
- All the debug information created by -g

Object Code ELF format

- ELF –Acronym from: Executable and Linkable Format
- First defined in 1983 UNIX “System V”
- Used for many different architectures (very popular)
- In 1996, chosen as standard for X86
- See https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

ELF File Format



ELF Header

- "Magic Number" – First 4 bytes identify this as ELF (0x72 + 'ELF')

x72	x45	x4C	x46
r	E	L	F

- Information about this file:
  - 32/64 bit addresses, big/little endian
  - Target operating system and architecture
  - relocatable, executable, shared, or core
- Starting Load address
- Program Table Info (loc, size, #entries) for use after load
- Section Table Info (offset, size, #entries) File table of contents

“Reading” ELF files : objdump

- -f : Interpret ELF header
- -h : List section headers (table of contents)
- -d / -D : Disassemble x86 binary code (.text) segment
- -t/-T : Interpret symbol table
- -s : dump everything in hex
  - -j<section> to restrict to a specific section

ELF Header Information

>objdump –f ttt

cmd: file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x080483c0

Segments or Sections

- Need different sections for different types of data
- Each section has it's own internal data format
- ELF header points to section table
- Section Table keeps "Section Header" for each segment
  - What kind of section is this
    - Section type/name
    - Section Flags
  - Starting location on disk (offset from beginning of the file)
  - Size of section
  - Location / Alignment in memory for this section

objdump –h cmd (dump section headers)

```
Sections:
Idx Name  Size      VMA       LMA       File off  Algn
 0  .interp  00000013  08048134  08048134  00000134  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
…
13 .text     00000290  080483c0  080483c0  000003c0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
14 .fini     00000017  08048650  08048650  00000650  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
15 .rodata 00000021  08048668  08048668  00000668  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
…
```

Example extract of Section Table

Index	Name	Size	Addr	File Off.	Align	Flags
…						
13	.text	x0290	080483c0	x03c0	2**4	CODE,ALLOC,READONLY
14	.fini					
15	.rodata	x0021	08048668	x0668	2**2	DATA,ALLOC,READONLY
…	…					
24	.data	x0120	08049880	x0880	2**5	DATA,ALLOC…
25	.bss	x0120	080499a0	x09a0	2**2	ALLOC
26	.comment	x0038	0	x09a0	2**0	… READONLY …
…						

.text section – x86 binary instructions

```
Contents of section .text:
 80483c0 31ed5e89 e183e4f0 50545268 e0850408  1.^.....PTRh....
 80483d0 68f08504 08515668 ac840408 e8bfffff  h....QVh........
 80483e0 fff49090 90909090 90909090 90909090  ................
 80483f0 b8a39904 082da099 040883f8 067702f3  .....-.......w..
 8048400 c3b80000 000085c0 74f55589 e583ec18  ........t.U.....
 8048410 c70424a0 990408ff d0c9c390 8d742600  ..$..........t&.
 8048420 b8a09904 082da099 0408c1f8 0289c2c1  .....-..........
```

…

.text section – x86 binary instructions

```
Contents of section .text:
 80483c0 31ed5e89 e183e4f0 50545268 e0850408  1.^.....PTRh....
 80483d0 68f08504 08515668 ac840408 e8bfffff  h....QVh........
```

Disassembly of section .text:

080483c0 <_start>:

```
 80483c0:        31 ed           xor    %ebp,%ebp
 80483c2:        5e              pop    %esi
 80483c3:        89 e1           mov    %esp,%ecx
 80483c5:        83 e4 f0        and    $0xfffffff0,%esp
 80483c8:        50              push   %eax
 80483c9:        54              push   %esp
 80483ca:        52              push   %edx
 80483cb:        68 e0 85 04 08  push   $0x80485e0
 80483d0:        68 f0 85 04 08  push   $0x80485f0
 80483d5:        51              push   %ecx
 80483d6:        56              push   %esi
 80483d7:        68 ac 84 04 08  push   $0x80484ac
 80483dc:        e8 bf ff ff ff  call   80483a0 <__libc_start_main@plt>
```

.data –initialized global/static data

```
Contents of section .data:
 8049880 00000000 00000000 00000000 00000000  ................
 8049890 00000000 00000000 00000000 00000000  ................
 80498a0 48656c6c 6f20576f 726c642e 2e2e2054  Hello World... T
 80498b0 68697320 69732061 6e20696e 69746961  his is an initia
 80498c0 6c697a65 6420676c 6f62616c 20766172  lized global var
 80498d0 6961626c 65207661 6c756500 00000000  iable value.....
```

…

.bss–Size of uninitialized data

- Takes no space in object code!
- "bss" acronym for "Block Storage Start"
  - or "Better Save Space"

```
Idx Name          Size      VMA       LMA       File off  Algn
25 .bss           00000120  080499a0  080499a0  000009a0  2**5
                  ALLOC
```

.rodata–Read only data (constants)

```
Contents of section .rodata:
 8048668 03000000 01000200 6e756d20 61726773  ........num args
 8048678 20697320 25640025 73202573 2025730a   is %d.%s %s %s.
 8048688 00
```

Symbol Table – Name Cross Reference

```
cmd:     file format elf32-i386

SYMBOL TABLE:
08048134 l    d  .interp          00000000              .interp
08048148 l    d  .note.ABI-tag    00000000              .note.ABI-tag
…
08048668 g     O .rodata          00000004              _fp_hw
080499a0 g       *ABS*            00000000              __bss_start
080484ac g     F .text            000000d7              main
00000000  w      *UND*            00000000              _Jv_RegisterClasses
0804858c g     F .text            00000051              utilFunc
00000000       F *UND*            00000000              sprintf@@GLIBC_2.0
080499a0 g     O .data            00000000              .hidden __TMC_END__
080499c0 g     O .bss             00000080              globalBuffer
```

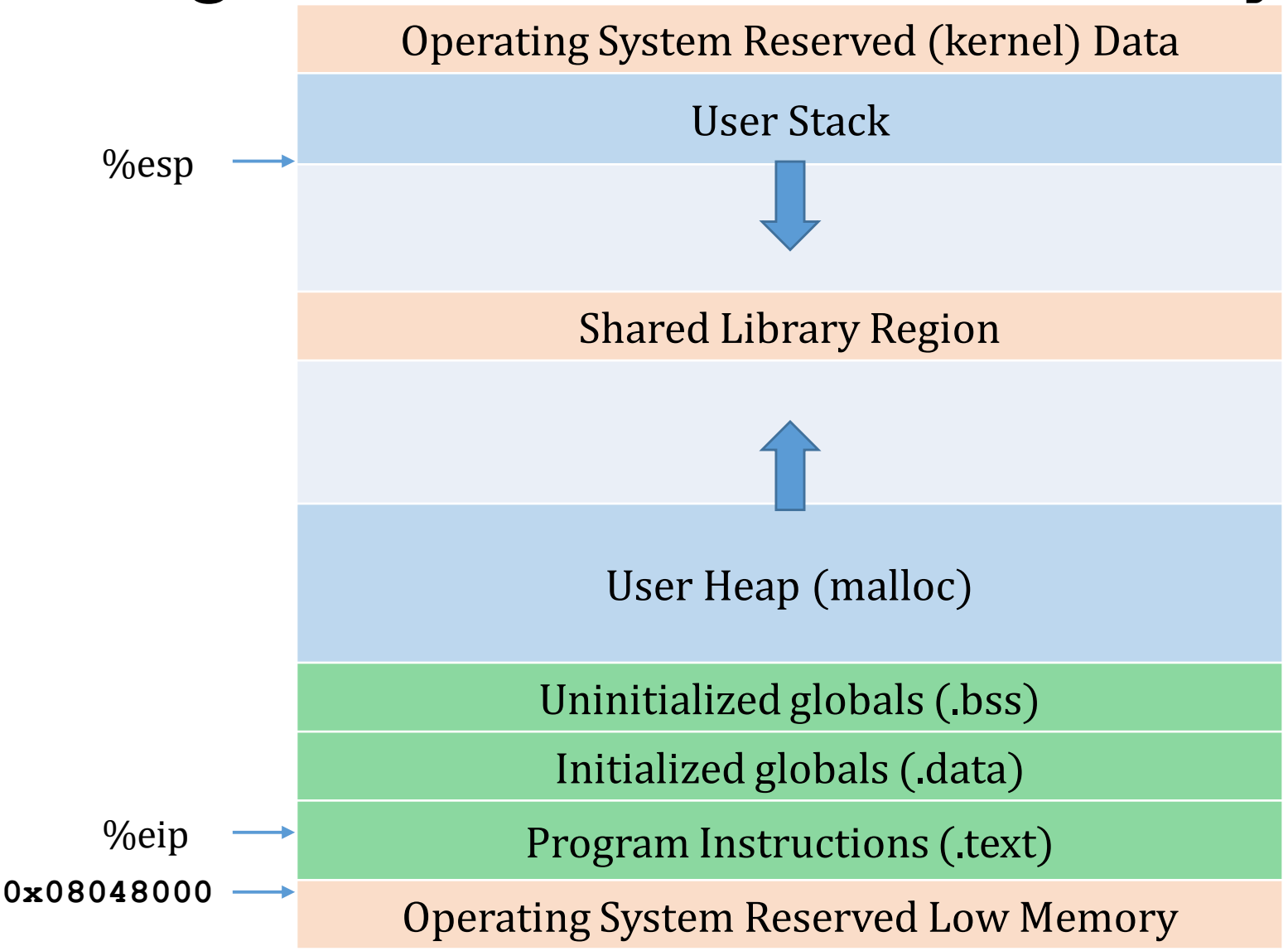
Symbol Table – Name Cross Reference

```
Location FLAGS   Section          Length/Alignment      Name
-------- ------- -------------    ----------------      ----------------------
08048134 l    d  .interp          00000000              .interp
08048148 l    d  .note.ABI-tag    00000000              .note.ABI-tag
…
08048668 g     O .rodata          00000004              _fp_hw
080499a0 g       *ABS*            00000000              __bss_start
080484ac g     F .text            000000d7              main
00000000  w      *UND*            00000000              _Jv_RegisterClasses
0804858c g     F .text            00000051              utilFunc
00000000       F *UND*            00000000              sprintf@@GLIBC_2.0
080499a0 g     O .data            00000000              .hidden __TMC_END__
080499c0 g     O .bss             00000080              globalBuffer
```

Symbol Table Flags

- Col 1: l=local, g=global, !=both, u=unique global, blank=neither
- Col 2: w=weak, blank=strong
- Col 3: C=constructor, blank=ordinary symbol
- Col 4: W=warning, blank=normal
- Col 5: I=indirect ref, i=function for relocation, blank=normal
- Col 6: d=debugging symbol, D=dynamic symbol, blank=normal
- Col 7: F=function, f=file, O=object, blank=normal

Loading an ELF file into Memory



Running a command

- Loading the command:
  - Finding the binary file in the file system
  - Parsing the command line into argc and argv
  - Creating a new address space (and process ID) for the command
  - Loading a binary file into memory (new address space)
- Executing the command:
  - Push argc and argv on the bottom of the stack
  - call main