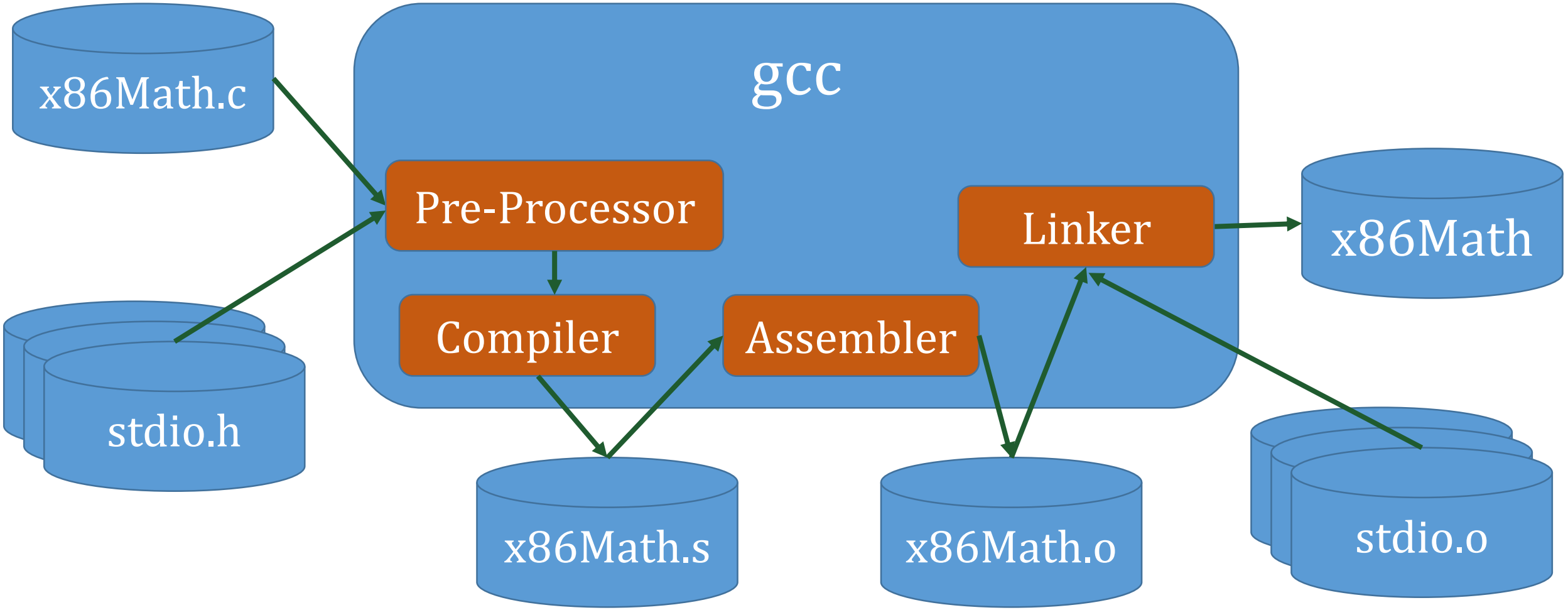
Loading Code

Computer Systems Chapter 7.5, 7.8, 7.9

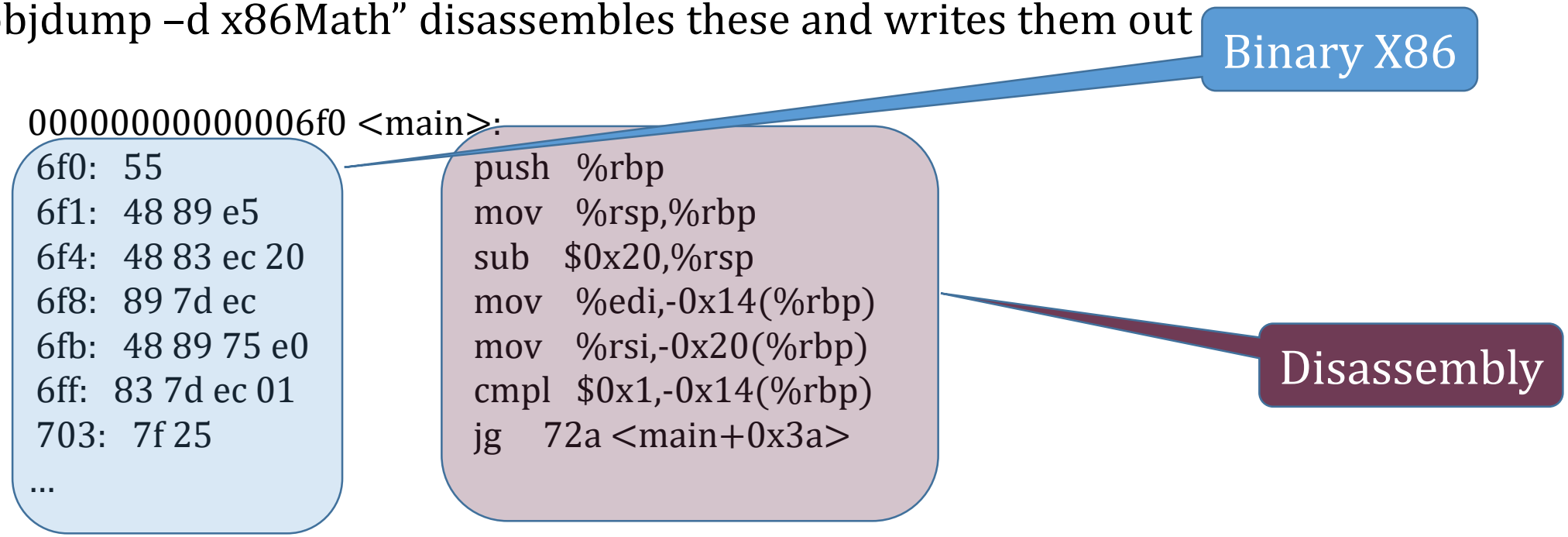
gcc –g –o ttt ttt.c

gcc –g –o x86Math x86Math.c



What is in a binary executable file?

- Binary representation of X86 instructions
  - "objdump –d x86Math" disassembles these and writes them out



- What else is in the binary executable file? …

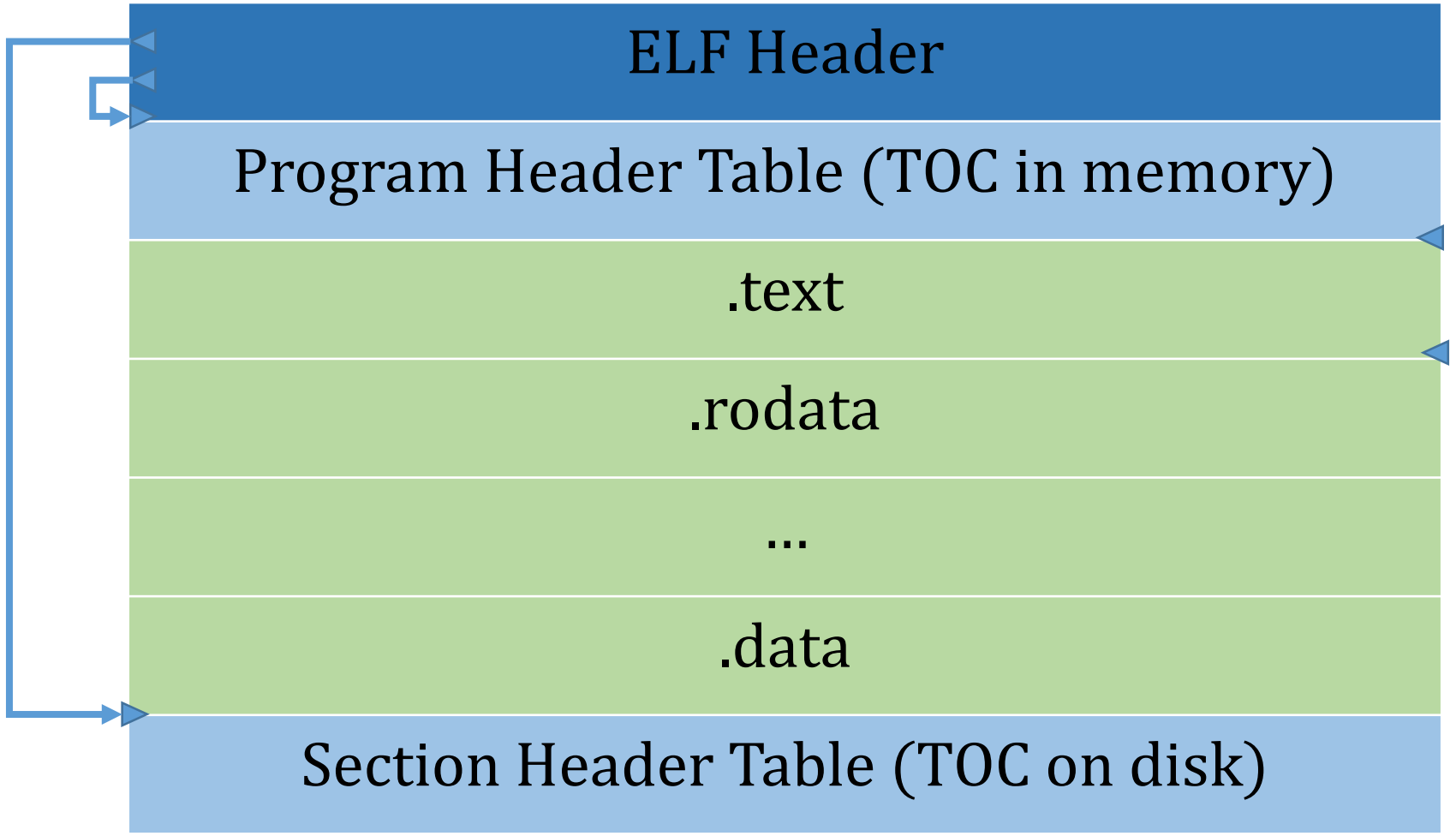
What else is in a binary file?

- 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… "Invoke as %s <number>\n"
  - 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/Executable 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 Sections



ELF Header

- "Magic Number" – First 4 bytes identify this as ELF (0x7f + 'ELF')

x7f	x45	x4C	x46
.	E	L	F

- Information about this file:
  - 32/64 bit addresses, big/little endian
  - Target operating system and architecture
  - relocatable, executable, shared, or core
- Program Table Info (loc, size, #entries) for use after load
- Offset of first instruction
- 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 : Disassemble x86 binary code (.text) segment
- -s : dump everything in hex
- -j<section> to restrict to a specific section
- -t : print symbol table

ELF Header Information

```
> objdump -f x86Math

x86Math:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000000005c0
```

ELF file 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

objdump –h x86Math (section headers)

```
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000000238  0000000000000238  00000238  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
…
 10 .init         00000017  0000000000000568  0000000000000568  00000568  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 11 .plt          00000030  0000000000000580  0000000000000580  00000580  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .plt.got      00000008  00000000000005b0  00000000000005b0  000005b0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .text         000002d2  00000000000005c0  00000000000005c0  000005c0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .fini         00000009  0000000000000894  0000000000000894  00000894  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .rodata       00000050  00000000000008a0  00000000000008a0  000008a0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
…
 31 .debug_str    000002be  0000000000000000  0000000000000000  00001750  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Some Interesting Sections

Index	Name	Size	Addr	Flags
…				
13	.text	x02d2	5c0	CODE,ALLOC,READONLY
…	…			
15	.rodata	x0050	8a0	DATA,ALLOC,READONLY
…	…			
24	.data	x0010	201028	DATA,ALLOC…
24	.bss	x0008	201038	ALLOC
…				

.text section – x86 binary instructions

```
> objdump -s -j.text x86Math

x86Math:     file format elf64-x86-64

Contents of section .text:
 05c0 31ed4989 d15e4889 e24883e4 f050544c  1.I..^H..H...PTL
 05d0 8d05ba02 0000488d 0d430200 00488d3d  ......H..C...H.=
…
> objdump -d -j.text x86Math
Disassembly of section .text:

00000000000005c0 <_start>:
 5c0:   31 ed                    xor    %ebp,%ebp
 5c2:   49 89 d1                 mov    %rdx,%r9
 5c5:   5e                       pop    %rsi

…
```

.data –initialized global/static data

```
> objdump -s -j.data x86Math

x86Math:     file format elf64-x86-64

Contents of section .data:
 201028 00000000 00000000 30102000 00000000  ........0. .....
```

.bss–Size of uninitialized data

- Takes no space in object code (or in file)!
- “bss” acronym for “Block Storage Start”
  - or “Better Save Space”
- Header indicates where this segment should start in memory, and how many bytes should be reserved for uninitialized data

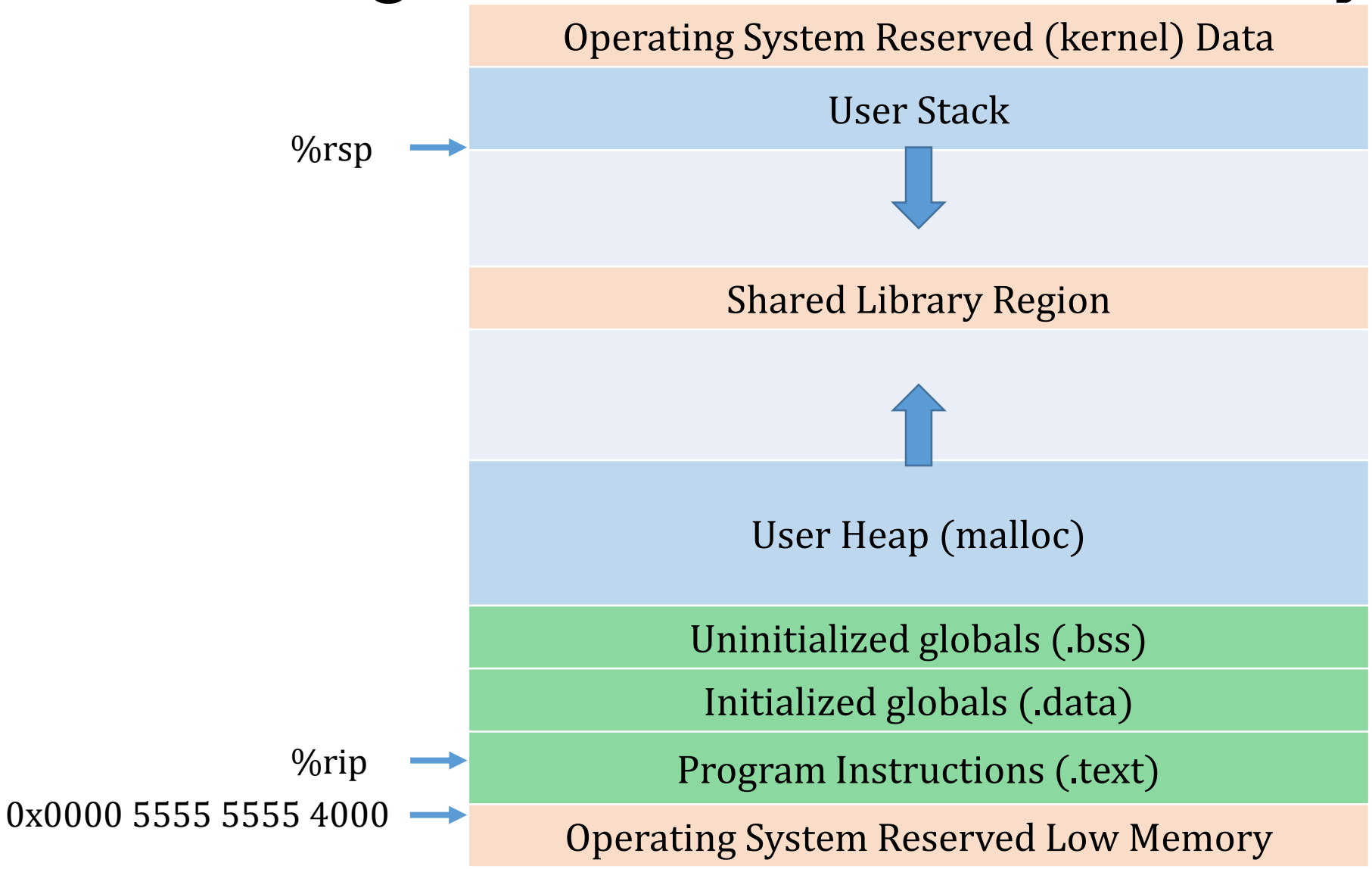
.rodata–Read only data (constants)

```
> objdump -s -j.rodata x86Math

x86Math:     file format elf64-x86-64

Contents of section .rodata:
 08a0 01000200 00000000 496e766f 6b652061  ........Invoke a
 08b0 73202573 203c6e75 6d626572 3e0a0000  s %s <number>...
 08c0 783d2564 2c207820 73717561 72656420  x=%d, x squared
 08d0 2d203478 202b2034 203d2564 20646976  - 4x + 4 =%d div
 08e0 69646564 20627920 782d323d 25640a00  ided by x-2=%d..
```

Running a command

- Loading the command:
  - Find the binary file in the file system
  - Create a new address space (and process ID) for the command
  - Load the binary file into memory (new address space)
  - Parse the command line into argc and argv
- Executing the command:
  - Put argc in %rdi and argv in %rsi
  - callq main (address is in symbol table)

Loading an ELF file into Memory



Symbol Table

- ELF keeps track of various “symbol” names ; names of sections, functions, external global variables, etc.
- These are kept in a “symbol table”

Offset	Flags	Section	Length	Name
0000000000000238	l d	.interp	0000000000000000	.interp
00000000000005c0	l d	.text	0000000000000000	.text
00000000000007cf	g F	.text	0000000000000026	add
0000000000000000	F	*UND*	0000000000000000	printf@@GLIBC_2.2.5

- Loader resolves unresolved symbols

Object Code

- It is possible to compile a single file of a multi-file program, and save the x86 code generated
- The result is saved in an ELF format file <pgm>.o
- References to functions outside this file are unresolved, but still in the symbol table
- A “Link Editor” combines object files from several C files into a single executable
  - Invoked by the gcc command
  - Resolves any symbols it can

Make w/ Multiple Compiles

```
test : fact
	./fact

fact: fact.o stack.o
	gcc -g -Wall -rdynamic -o fact fact.o stack.o

fact.o : fact.c stack.h
	gcc -g -Wall -c fact.c

stack.o : stack.c stack.h
	gcc -g -Wall -c stack.c

clean:
	-rm fact fact.o stack.o
```

Link Editor resolves symbols

- objdump –d fact.o…

```
 82:    e8 00 00 00 00       callq  87 <factorial+0x54>
 87:    8b 45 fc             mov    -0x4(%rbp),%eax
```

- objdump –d fact

```
b42:    e8 05 00 00 00       callq  b4c <printStackInfo>
b47:    8b 45 fc             mov    -0x4(%rbp),%eax
```

- gdb --args fact

```
   0x0000555555554b42 <+79>:    callq  0x555555554b4c <printStackInfo>
=> 0x0000555555554b47 <+84>:    mov    -0x4(%rbp),%eax
```

Dynamically Resolved Symbols

- Library code often in “Dynamically Loadable Libraries” (DLL’s)
- DLL’s are loaded when they are first used
  - Prevents loading enormous amounts of library code for each program
  - May be shared across different programs
- Invocation of a function in a DLL is resolved when it is called
  - Code invokes a wrapper function, e.g. printf@plt
  - printf@plt function knows if printf has been resolved
    - If not, load the DLL, find the address of printf, and save it
    - In either case, call the real printf