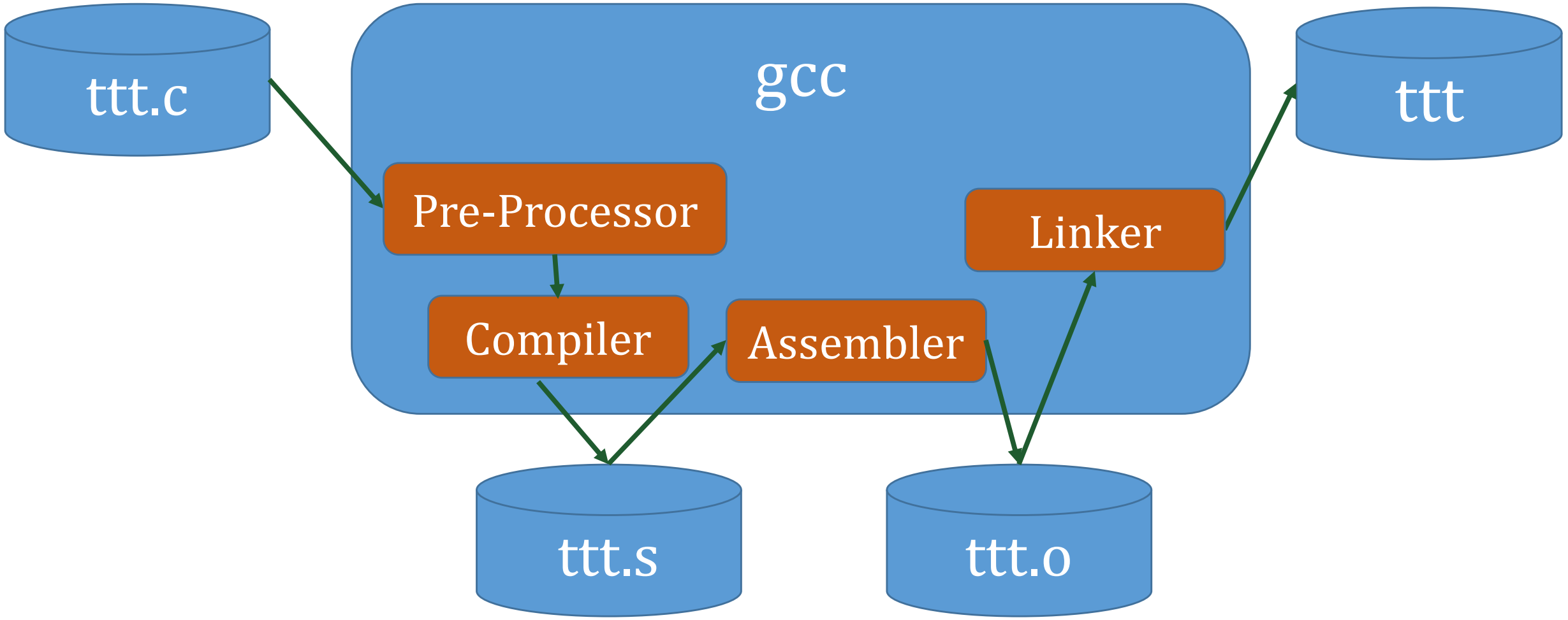
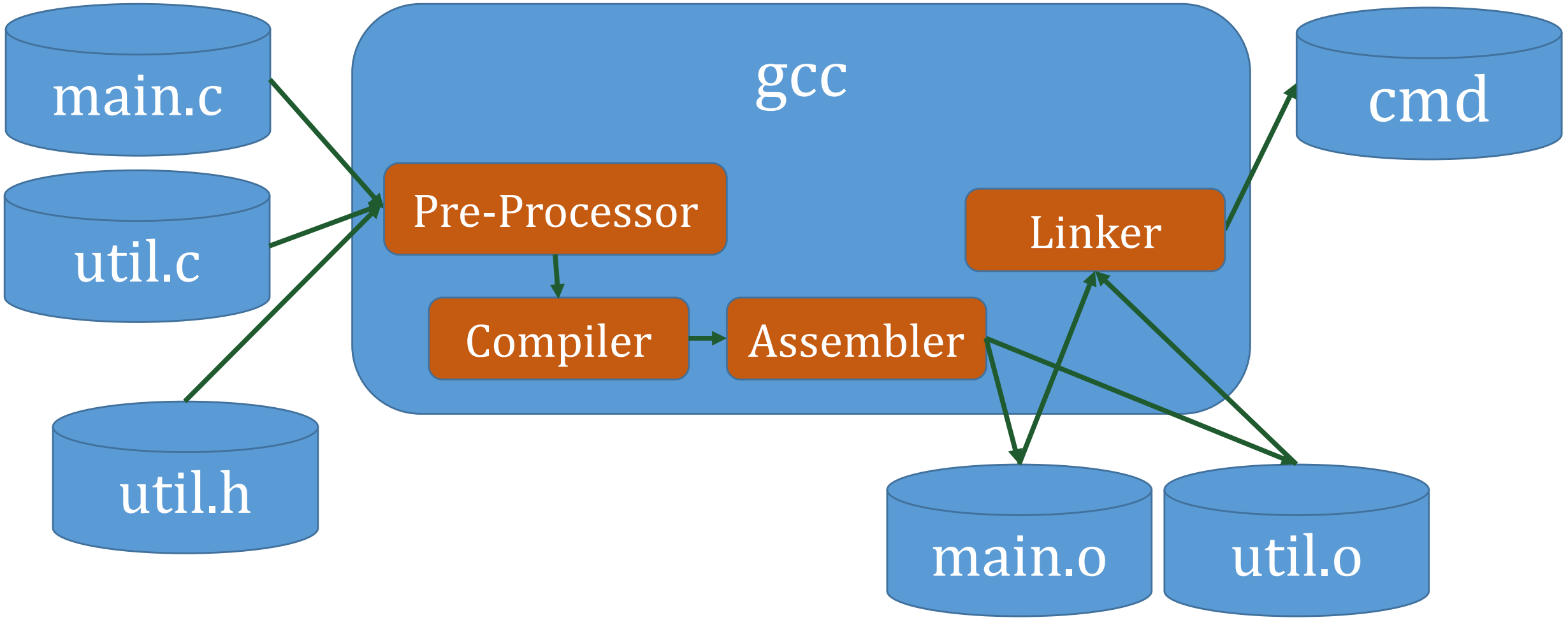
Link Edits and Relocatable Code

Computer Systems Chapter 7.4-7.7

gcc –g –o ttt ttt.c



gcc –g –o ttt ttt.c



Example: main.c preamble

```
#include <stdio.h>
#include <string.h>
#include "util.h“

char globalBuffer[128];
char globalBufferInit[128]=
        "Hello World... This is an initialized global variable value";

char * mainFunc(char * arg1,char *arg2);
```

Example: util.h

```
char * utilFunc(char *p1,char *p2);
```

Example: main.c main function

```
int main(int argc, char **argv) {
    char localBuffer[128]; int i;
    sprintf(localBuffer,"num args is %d",argc);
    strcat(globalBufferInit,localBuffer);
    globalBuffer[0]=0x00;
    for (i=0; i<argc; i++) { strcat(globalBuffer,argv[i]); }
    utilFunc(globalBufferInit,localBuffer);
    printf("%s %s %s\n",
        localBuffer,globalBuffer,mainFunc(globalBufferInit,localBuffer));
    return 0;
}
```

Example: main.c mainFunc function

```
char * mainFunc(char * arg1,char *arg2) {
        return arg1;
}
```

Example: util.c

```
#include <string.h>
char globUtilBuf[128];
char globUtilBufInit[128]="This is a global in util.h";
char * utilFunc(char *p1,char *p2) {
    char locUtilBuf[128];
    strcpy(p1,locUtilBuf);
    strcat(locUtilBuf,globUtilBufInit);
    strcpy(locUtilBuf,globUtilBuf);
    return globUtilBuf;
}
```

Example: Makefile

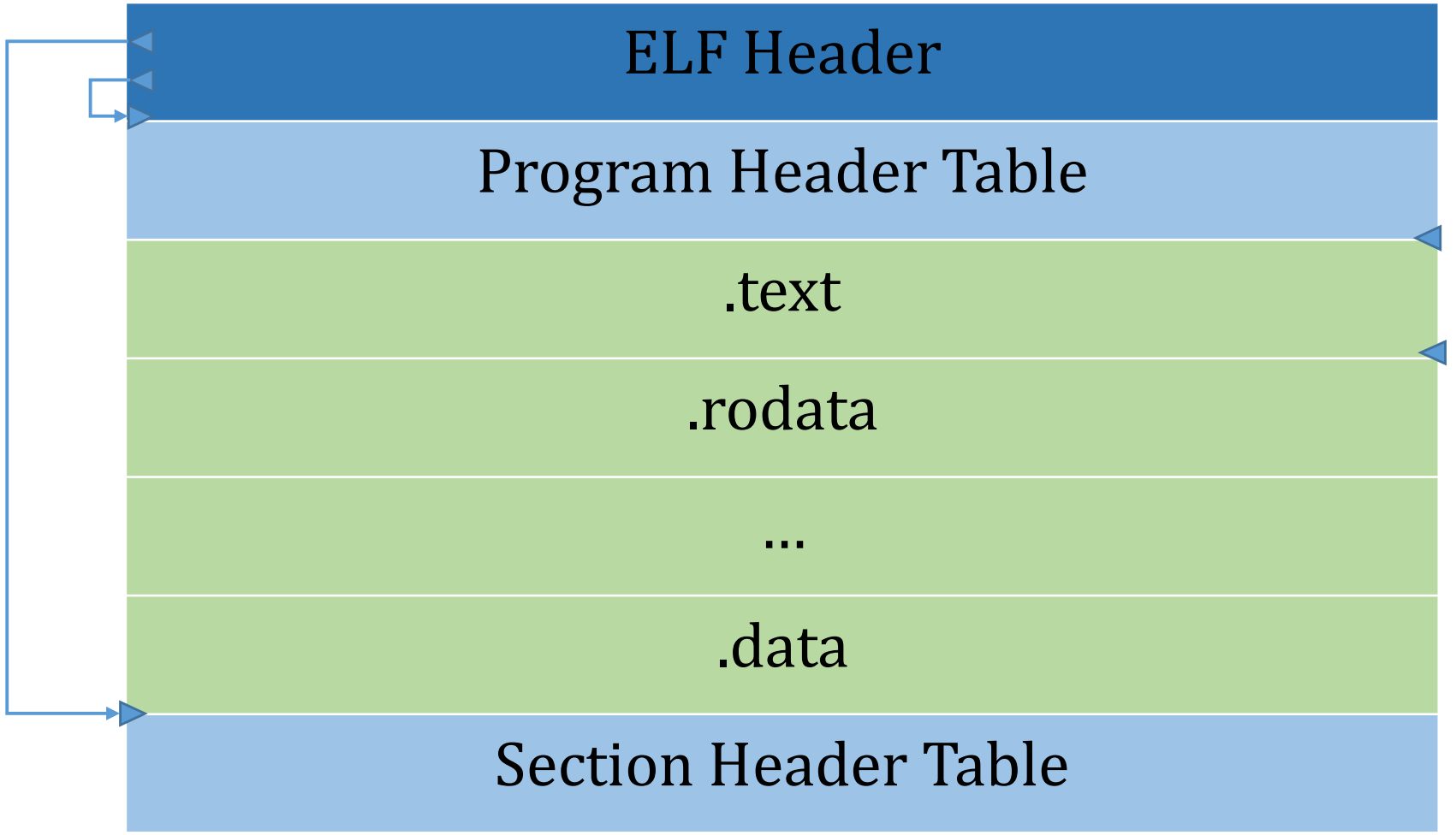
```
cmd: main.outil.o
	gcc-g -m32 -o cmd main.o util.o

main.o: main.c util.h
	gcc-g -m32 -c -o main.o main.c

util.o: util.c util.h
	gcc-g -m32 -c -o util.o util.c

clean :
-rm util.o main.o cmd
```

ELF File Format – used for .o files



“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
- -r : dump relocation records

ELF Header Information

```
> objdump-f main.o
main.o: file format elf32-i386
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000
```

```
>objdump –f cmd
cmd:     file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x080483c0
```

Relocatable vs. Fixed Object Code

Relocatable

- References are in terms of offsets from registers (e.g. %esp and %ebp or %eip)
- Certain references remain references to symbols which must be resolved when loaded
- Can be loaded anywhere in memory

Unrelocatable–Position Dependent

- References are fixed addresses as well as offsets from registers
- All references have been resolved –no references to symbols… only addresses
- Can only be loaded in one specific place in memory

Why Relocate?

- Keep object code flexible
- One object file can be used in multiple commands
- Might end up at different locations for different commands
- But relocatable object code is NOT executable!
- All symbolic references MUST be resolved before execution can take place!

```
main.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000df  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000080  00000000  00000000  00000120  2**5
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000001a0  2**2
                  ALLOC
  3 .rodata       00000019  00000000  00000000  000001a0  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
...
```

Example: main.c

```
char globalBuffer[128];
char globalBufferInit[128]=
        "Hello World... This is an initialized global variable value";

int main(int argc, char **argv) {
    char localBuffer[128]; int i;
    sprintf(localBuffer,"num args is %d",argc);
    strcat(globalBufferInit,localBuffer);
    globalBuffer[0]=0x00;
    for (i=0; i<argc; i++) { strcat(globalBuffer,argv[i]); }
    utilFunc(globalBufferInit,localBuffer);
    printf("%s %s %s\n",
        localBuffer,globalBuffer,mainFunc(globalBufferInit,localBuffer));
    return 0;
}
```

Uninitialized Global in .bss

Initialized Global in .data

Parameters in stack

Dynamic Local Variables in stack

Constant in .rodata

Constant in instruction (.text)

objdump –s –j.data main.o

```
Contents of section .data:
 0000 48656c6c 6f20576f 726c642e 2e2e2054  Hello World... T
 0010 68697320 69732061 6e20696e 69746961  his is an initia
 0020 6c697a65 6420676c 6f62616c 20766172  lized global var
 0030 6961626c 65207661 6c756500 00000000  iable value.....
 0040 00000000 00000000 00000000 00000000  ................
 0050 00000000 00000000 00000000 00000000  ................
 0060 00000000 00000000 00000000 00000000  ................
 0070 00000000 00000000 00000000 00000000  ................
```

objdump –s –j.rodata

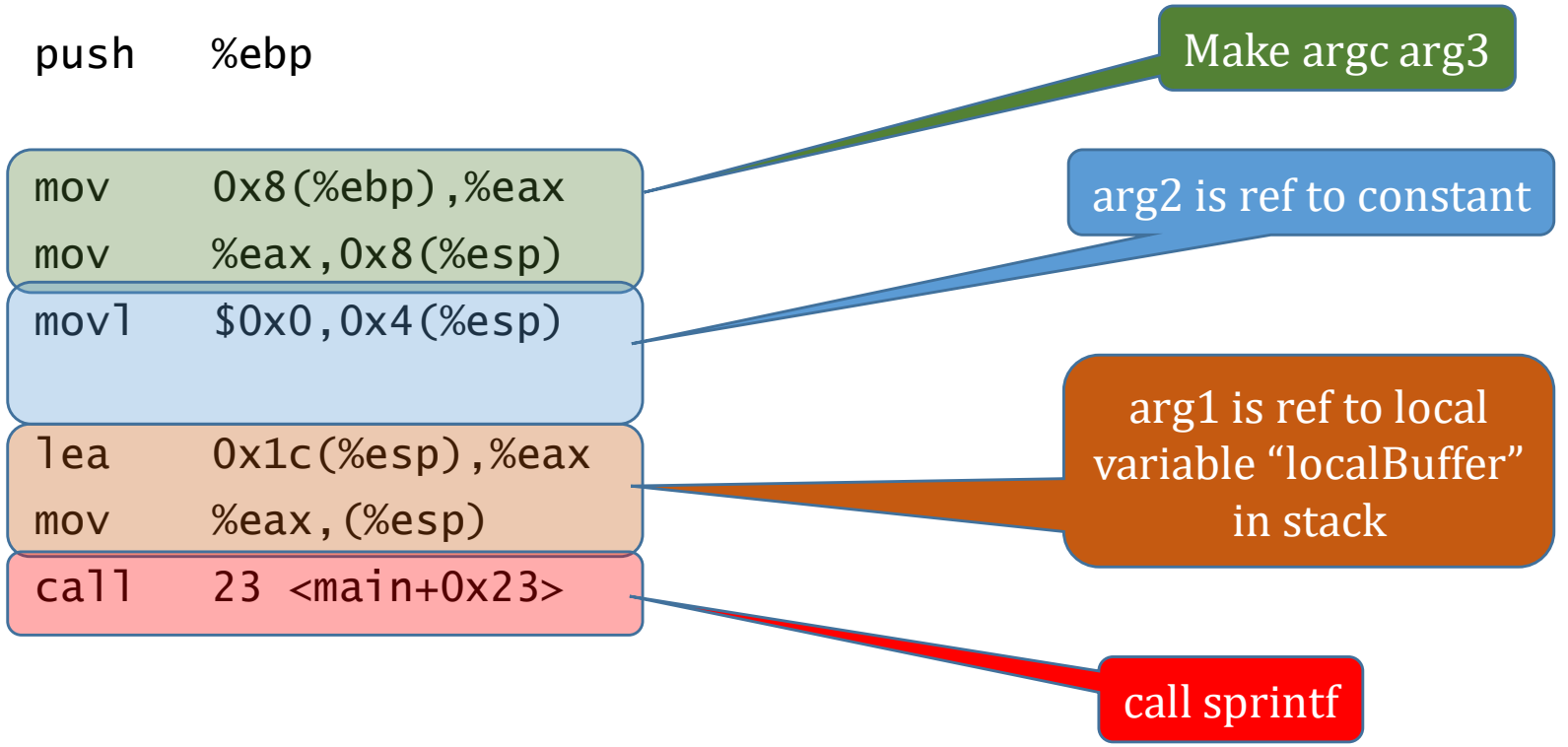
```
Contents of section .rodata:
 0000 6e756d20 61726773 20697320 25640025  num args is %d.%
 0010 73202573 2025730a 00                 s %s %s..
```

objdump –d main.o

```
Disassembly of section .text:

00000000 <main>:
   0:   55                      push   %ebp
   …
   c:   8b 45 08                mov    0x8(%ebp),%eax
   f:   89 44 24 08             mov    %eax,0x8(%esp)
  13:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
  1a:   00
  1b:   8d 44 24 1c             lea    0x1c(%esp),%eax
  1f:   89 04 24                mov    %eax,(%esp)
  22:   e8 fc ff ff ff          call   23 <main+0x23>
```

sprintf(localBuffer,"num args is %d",argc);



objdump –r main.o

```
main.o:     file format elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000017 R_386_32          .rodata
00000023 R_386_PC32        sprintf
00000032 R_386_32          globalBufferInit
00000037 R_386_PC32        strcat
...
```

objdump –d cmd (non-relocatable)

```
Disassembly of section .text:

080484ac <main>:
 80484ac:      55                      push   %ebp
…
 80484b8:      8b 45 08                mov    0x8(%ebp),%eax
 80484bb:      89 44 24 08             mov    %eax,0x8(%esp)
 80484bf:      c7 44 24 04 70 86 04    movl   $0x8048670,0x4(%esp)
 80484c6:      08
 80484c7:      8d 44 24 1c             lea    0x1c(%esp),%eax
 80484cb:      89 04 24                mov    %eax,(%esp)
 80484ce:      e8 dd fe ff ff          call   80483b0 <sprintf@plt>
```

sprintf(localBuffer,"num args is %d",argc);

Make argc arg3

arg2 is ref to constant

arg1 is ref to local variable “localBuffer” in stack

call sprintf

objdump –d main.o

utilFunc(globalBufferInit,localBuffer);

```
Disassembly of section .text:
…
  88:    8d 44 24 1c
  8c:    89 44 24 04
  90:    c7 04 24 00 00 00 00
  97:    e8 fc ff ff ff
```

```
lea    0x1c(%esp),%eax
mov    %eax,0x4(%esp)
movl   $0x0,(%esp)
call   98 <main+0x98>
```

arg2 is ref to local variable "localBuffer" in stack

arg1 is ref to variable "globalBufferInit" in .data

call utilFunc

```
Relocation records…
00000093 R_386_32          globalBufferInit
00000098 R_386_PC32        utilFunc
```

objdump –d cmd

utilFunc(globalBufferInit,localBuffer);

```
Disassembly of section .text:
…
 8048534:        8d 44 24 1c
 8048538:        89 44 24 04
 804853c:        c7 04 24 a0 98 04 08
 8048543:        e8 44 00 00 00
```

```
lea     0x1c(%esp),%eax
mov     %eax,0x4(%esp)
movl    $0x80498a0,(%esp)
call    804858c <utilFunc>>
```

arg2 is ref to local variable “localBuffer” in stack

arg1 is ref to variable “globalBufferInit” in .data

call utilFunc

objdump –t main.o

```
main.o:     file format elf32-i386

SYMBOL TABLE:
…
00000080         O *COM*     00000020 globalBuffer
00000000 g       O .data     00000080 globalBufferInit
00000000 g       F .text     000000d7 main
00000000           *UND*     00000000 sprintf
00000000           *UND*     00000000 strcat
00000000           *UND*     00000000 utilFunc
000000d7 g       F .text     00000008 mainFunc
00000000           *UND*     00000000 printf
```

objdump –t cmd

```
cmd:     file format elf32-i386

SYMBOL TABLE:
…
080499c0 g     O .bss   00000080              globalBuffer
080498a0 g     O .data  00000080              globalBufferInit
080484ac g     F .text  000000d7              main
00000000       F *UND*  00000000              sprintf@@GLIBC_2.0
00000000       F *UND*  00000000              strcat@@GLIBC_2.0
0804858c g     F .text  00000051              utilFunc
08048583 g     F .text  00000008              mainFunc
00000000       F *UND*  00000000              printf@@GLIBC_2.0
```

Static Libraries

- Libraries of object code
- Parameters to Link Editor instruct the link editor to attempt to resolve unresolved functions by including these libraries
- For instance, gcc "-lm" causes the link editor to include /usr/lib32/libm.so
- Object code from these files is included in your executable file
  - Makes your command larger
  - Guarantees you will find these functions when you load and run your command

Dynamically Loaded Libraries

- Link editor leaves references to functions in dynamically loaded libraries unresolved

For instance

```
00000000       F *UND*   00000000                 sprintf@@GLIBC_2.0
00000000       F *UND*   00000000                 strcat@@GLIBC_2.0
```

- Link editor inserts code to load these libraries at run time
  - For instance, GLIBC_2.0 library loaded at run time
  - Once loaded, unresolved symbols are replaced
- Dynamic libraries take no space in executable file, but may not be resolved (or resolved correctly) at run time

Shared Libraries

- Dynamically loaded libraries that are used by multiple commands
- Typical of C standard libraries
- More about sharing once we learn about virtual memory