Introduction to 8086 Assembly

Lecture 3

Object files, compiling, assembling and linking

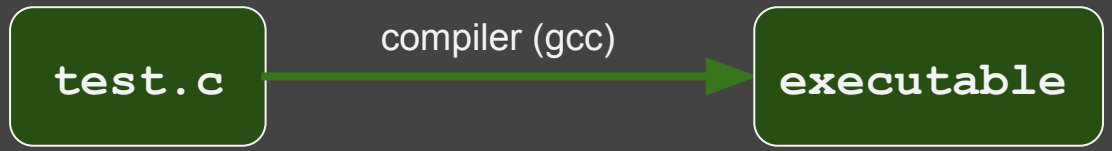
Compiling a C file

test.c

```c
#include <stdio.h>

int main() {

  printf("Salaaaaam!!!\n");

  return  0;
}
```

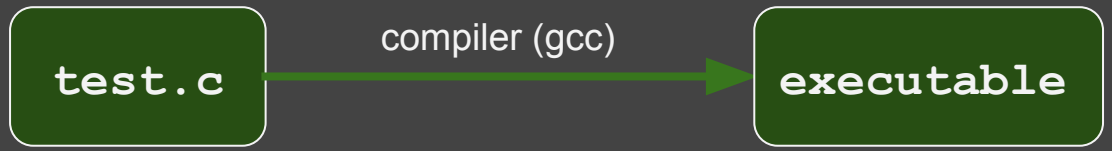
Compiling a C file

K. N. Toosi
University of Technology



Compiling a C file

K. N. Toosi
University of Technology




```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
```

Compiling a C file

K. N. Toosi
University of Technology

test.c

```c
#include <stdio.h>

int main() {

  printf("Salaaaaam!!!\n");

  return  0;
}
```

```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
```

Compiling a C file

K. N. Toosi
University of Technology

```c
#include <stdio.h>

int main() {

  printf("Salaaaaam!!!\n");

  return  0;
}
```

Who wrote **printf**?
Where is the code for **printf**?

```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
```

Compiling a C file

test.c

```
extern int printf (const char *__restrict __format, ...);

int main() {

  printf("Salaaaaam!!!\n");

  return  0;
}
```

Compiling a C file

K. N. Toosi
University of Technology

test.c

```c
extern int printf (const char *__restrict __format, ...);

int main() {

  printf("Salaaaaam!!!\n");

  return  0;
}
```

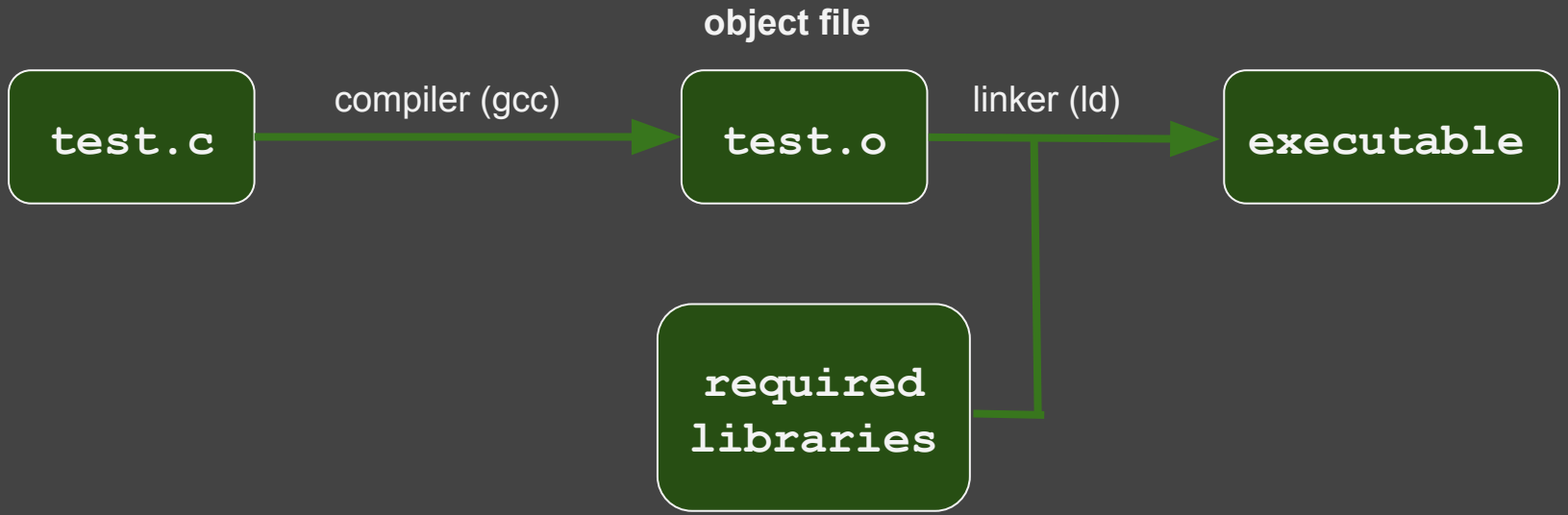
```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
b.nasihatkon@kntu:lecture3$
```

Compiling a C file, object files

K. N. Toosi
University of Technology

Object files, libraries and linking

K. N. Toosi
University of Technology

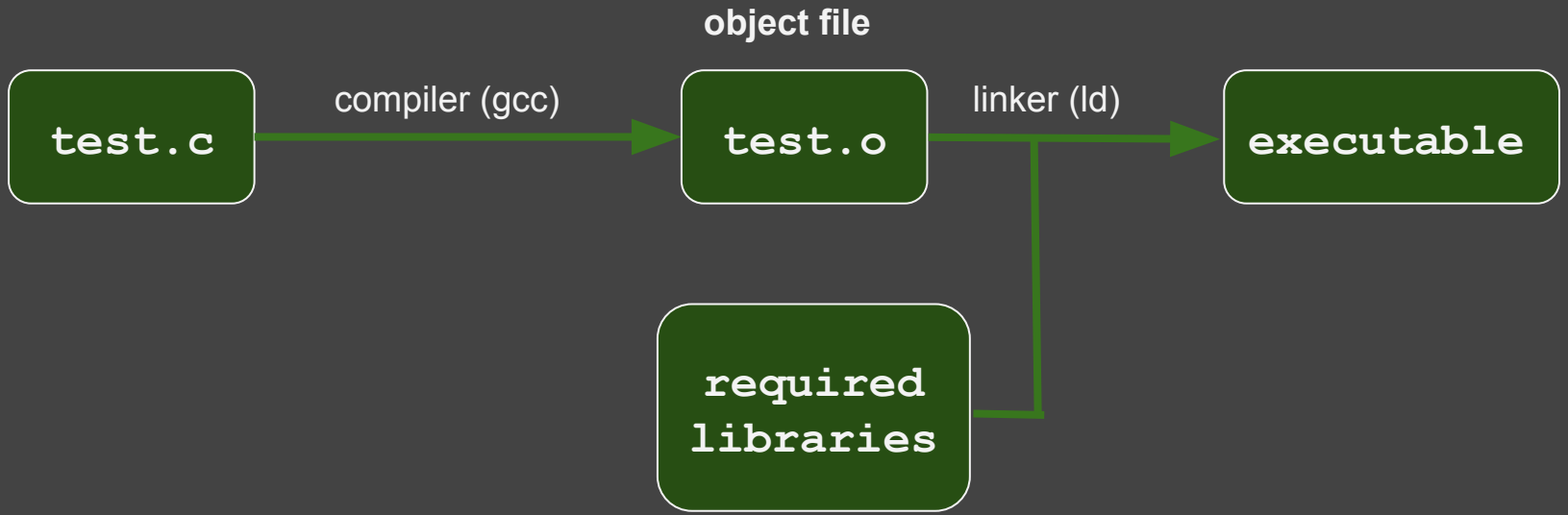


Object files

- Machine code + metadata (unresolved symbols, etc.)
  - for linking, debugging, etc.
  - https://en.wikipedia.org/wiki/Object_file
  - https://en.wikipedia.org/wiki/Object_code
- Object file formats
  - Common Object File Format (COFF)
  - Relocatable Object Module Format (OMF)
  - **Executable and Linkable Format (ELF)**

Libraries

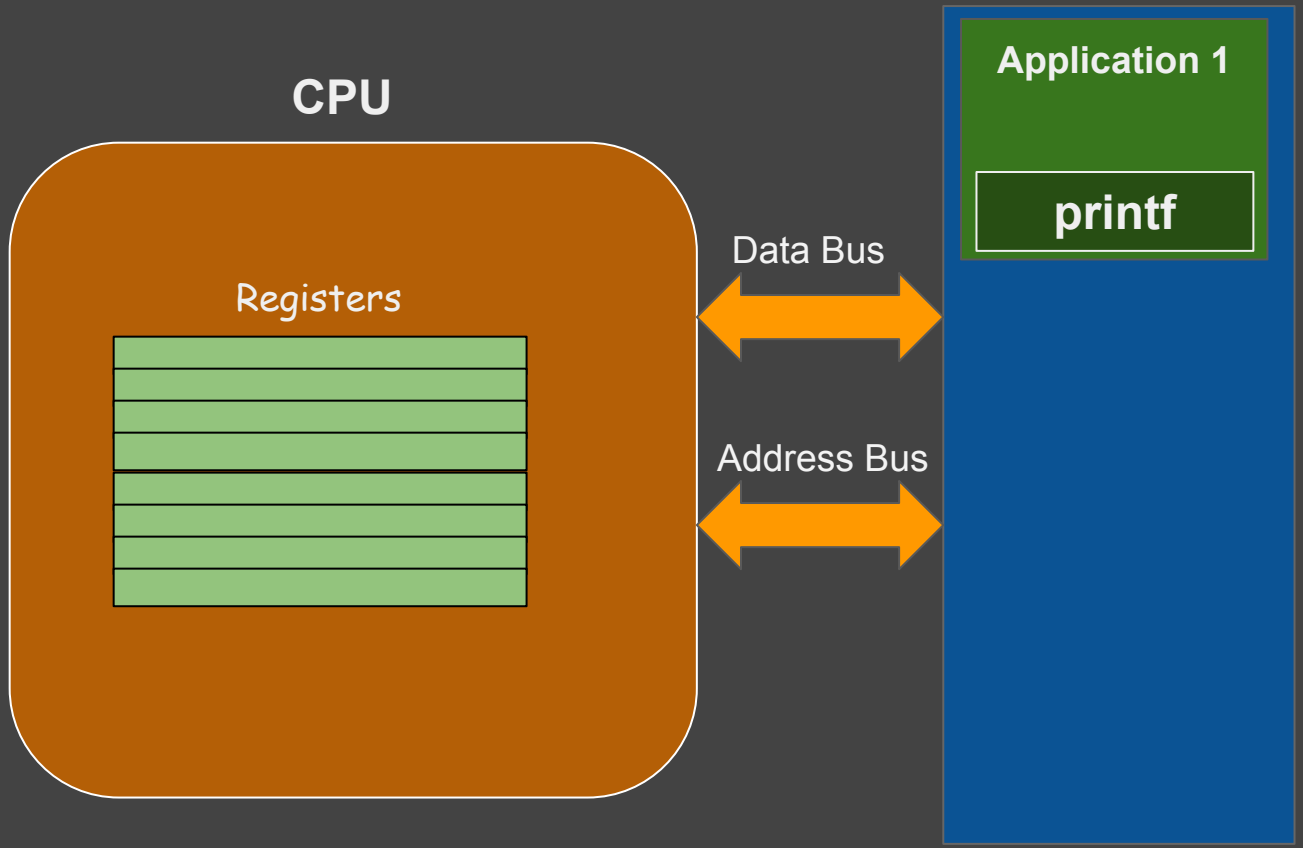
K. N. Toosi
University of Technology



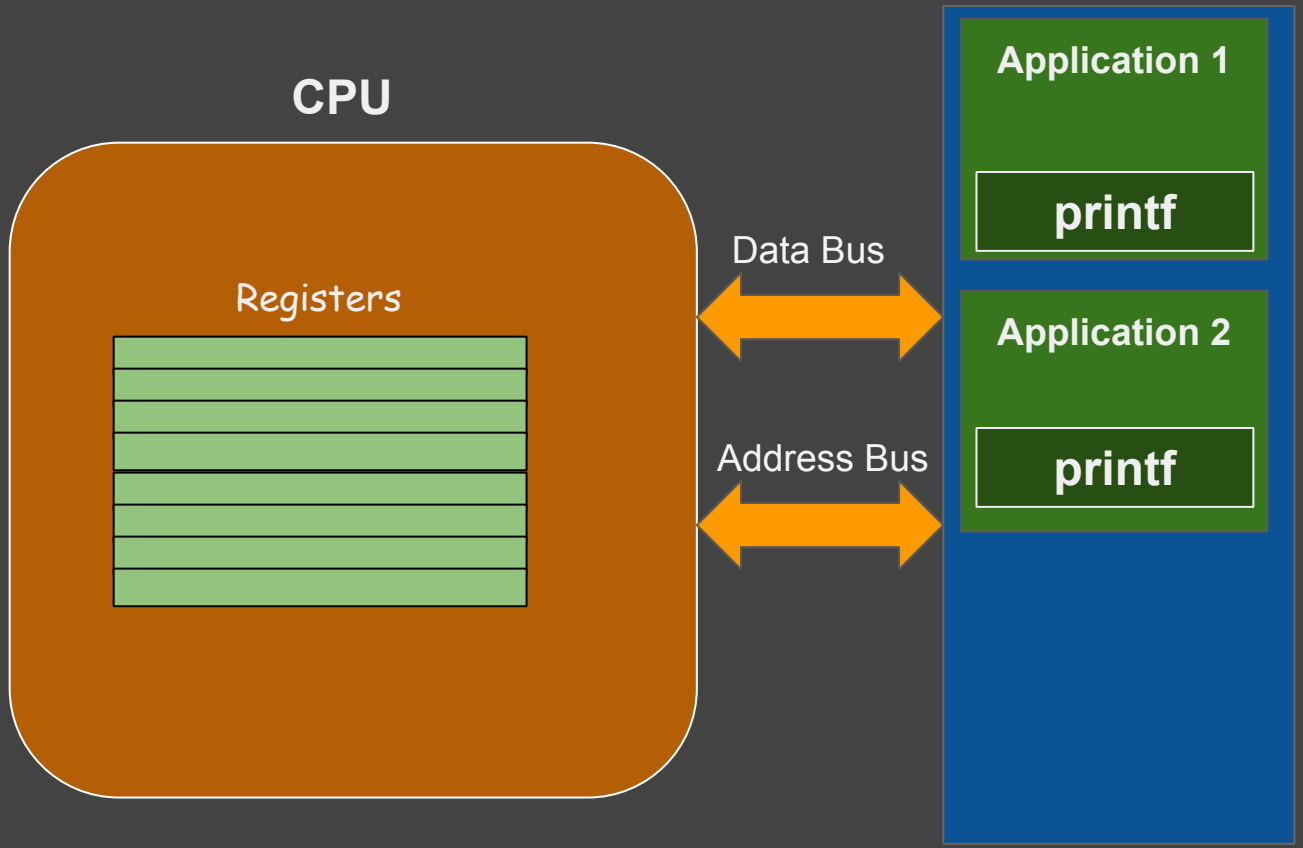
Libraries

- Collection of object files
- Static vs. Dynamic Linking
- Shared Objects (SO), Dynamic-Link Libraries (DLL)

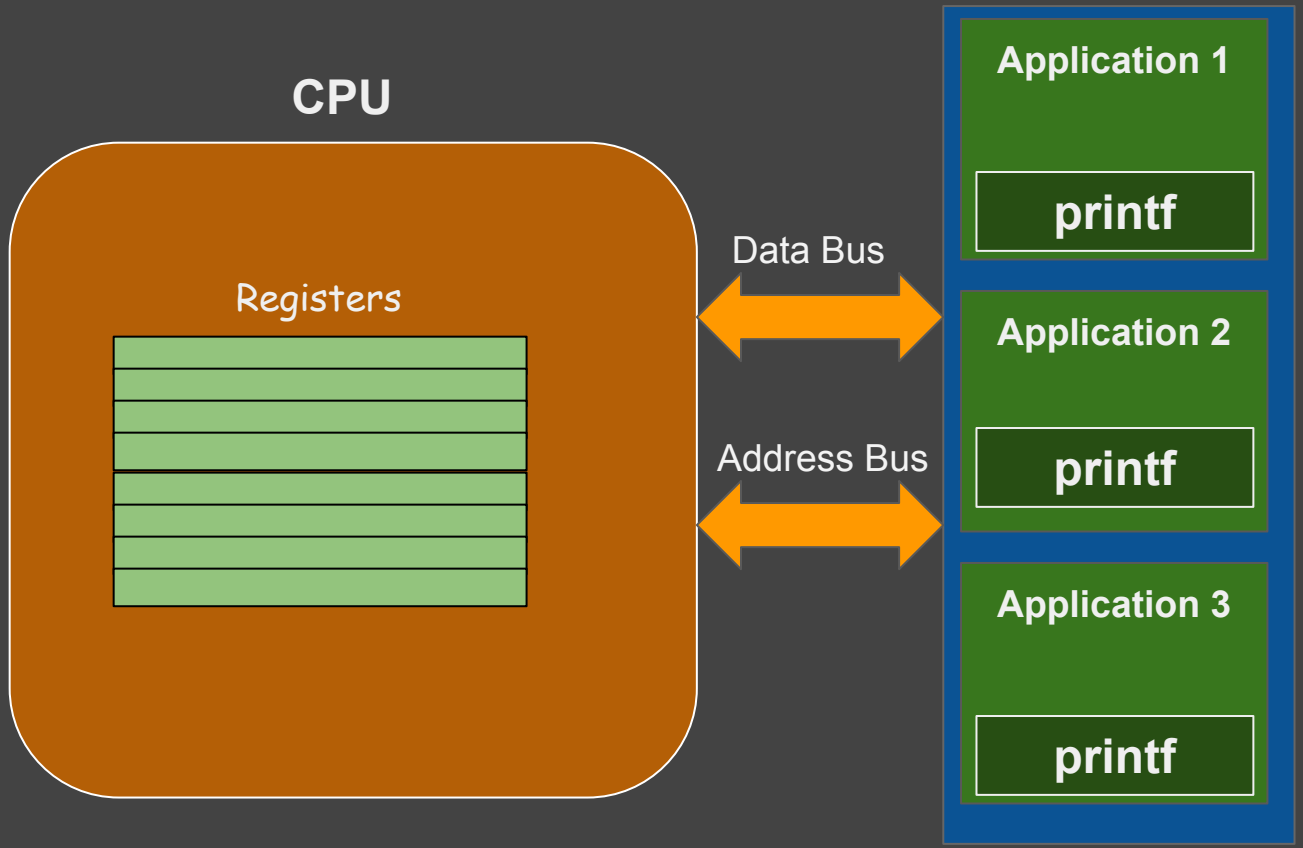
Static Libraries, Static Linking



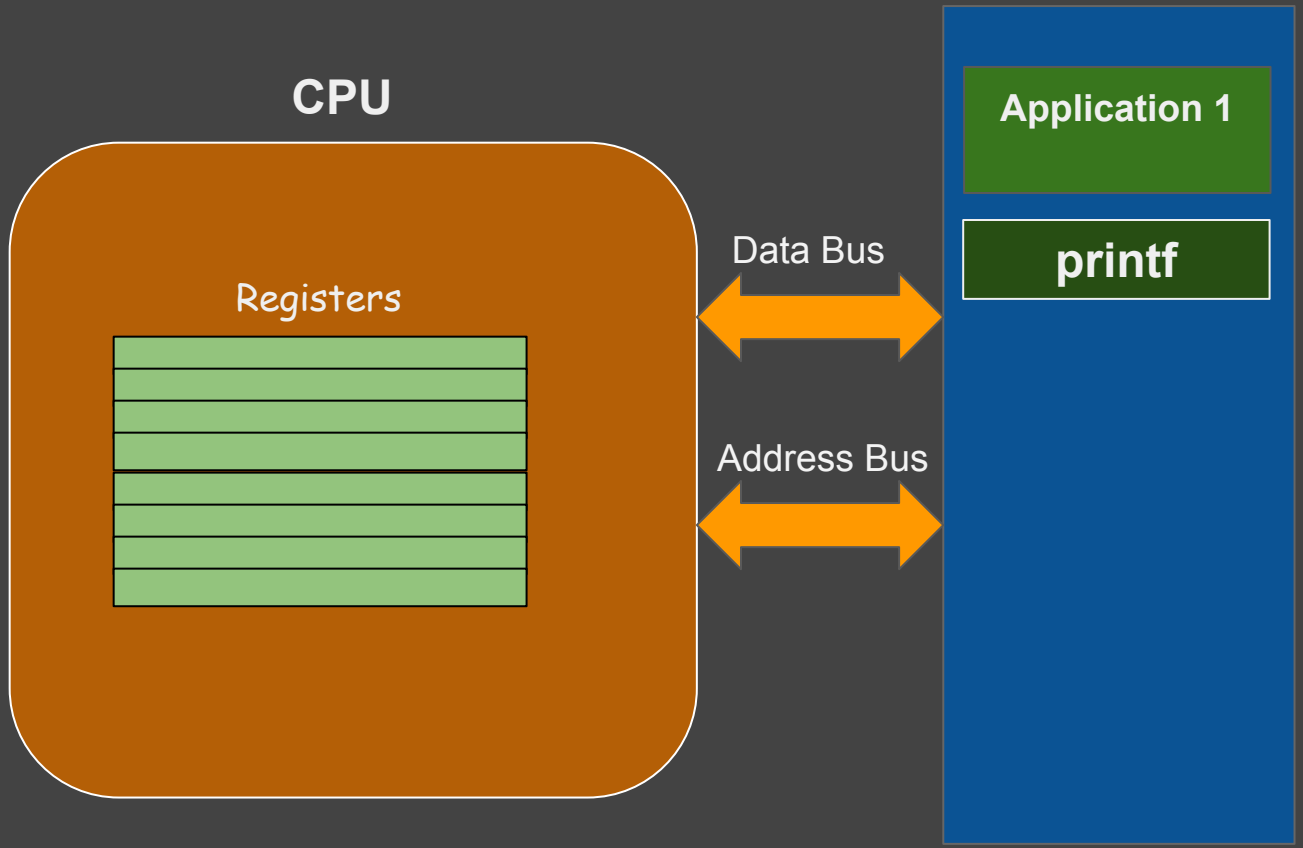
Static Libraries, Static Linking



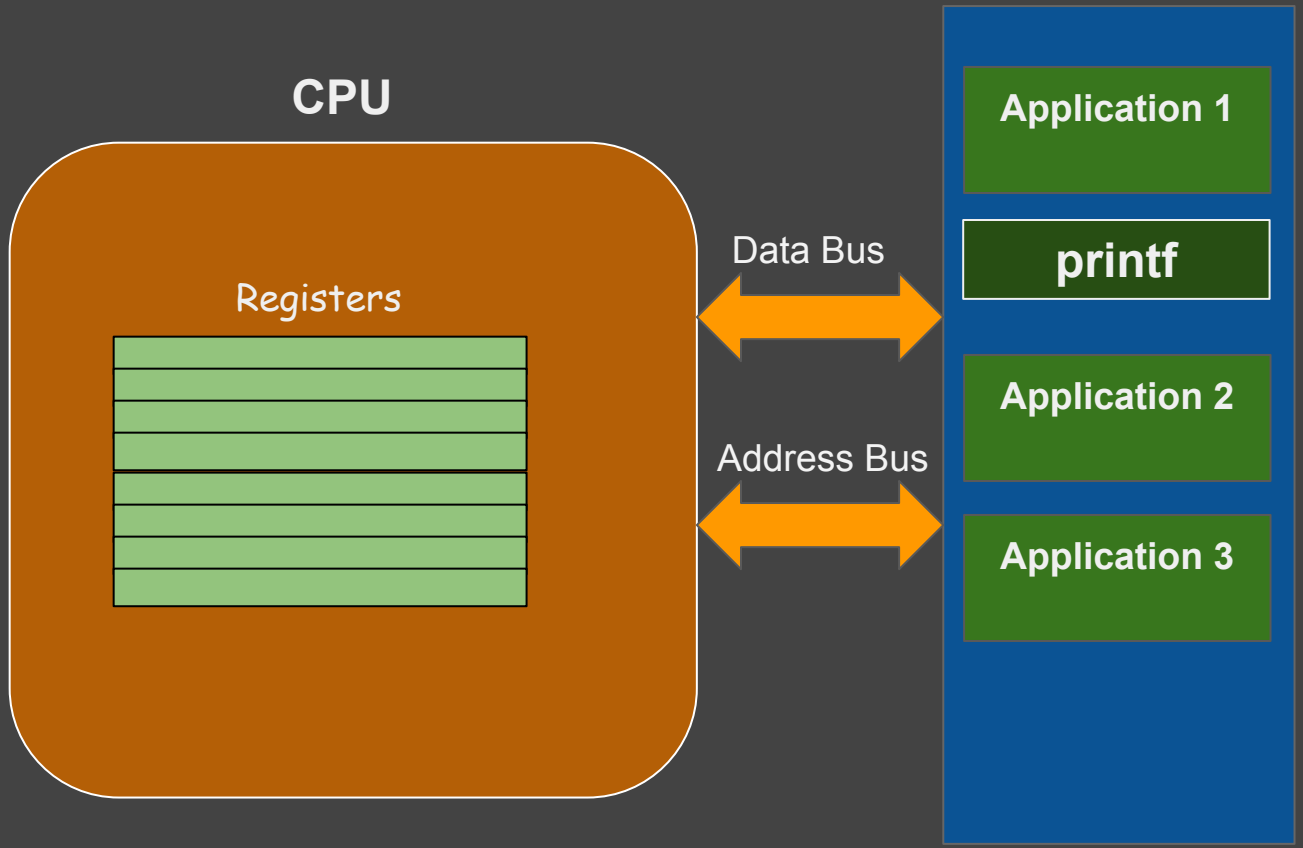
Static Libraries, Static Linking



Dynamic Libraries, Dynamic Linking



Dynamic Libraries, Dynamic Linking

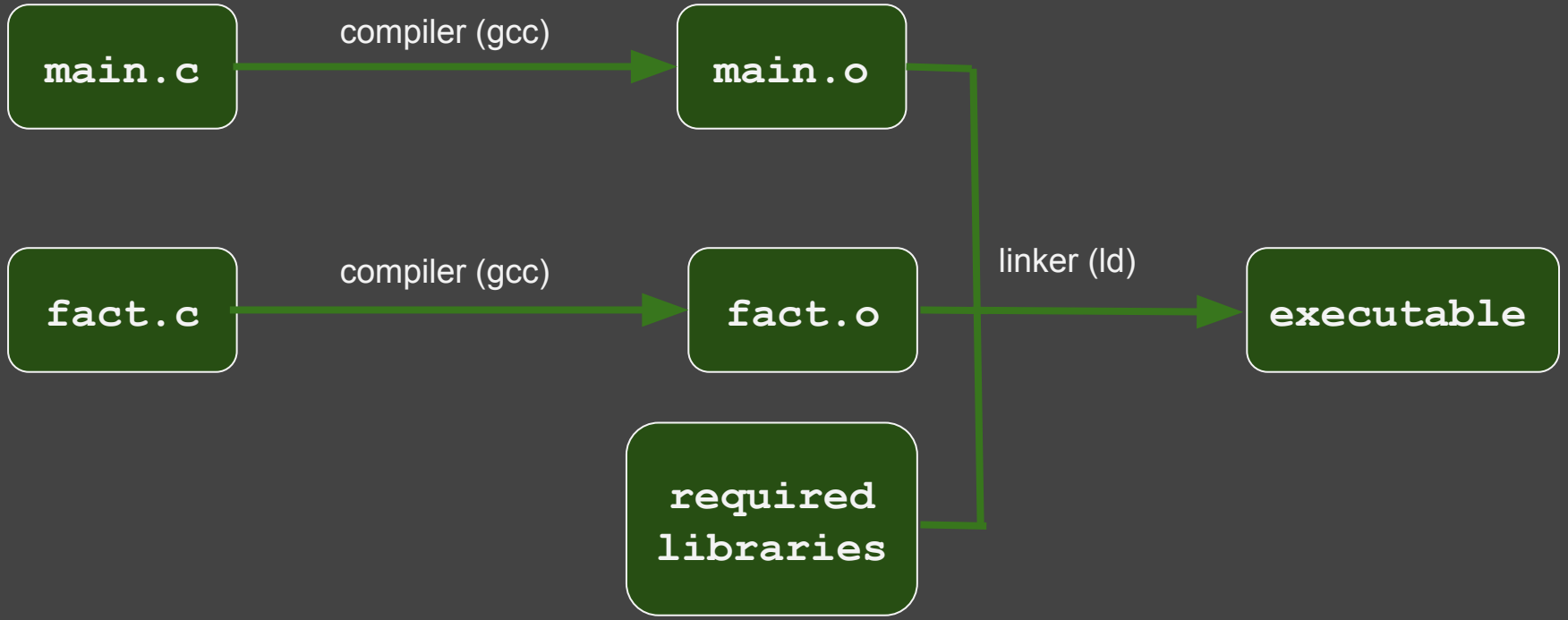


Dynamic vs. Static Linking

- Advantages of Static Libraries (.a, .lib)
  - faster linking
  - usually faster function call
  - standalone executable
- Advantages of Dynamic Libraries (shared objects) (.so, .dll)
  - no need to recompile all programs if library updated
  - smaller executables
  - less memory per app

Modular programming in C

K. N. Toosi
University of Technology

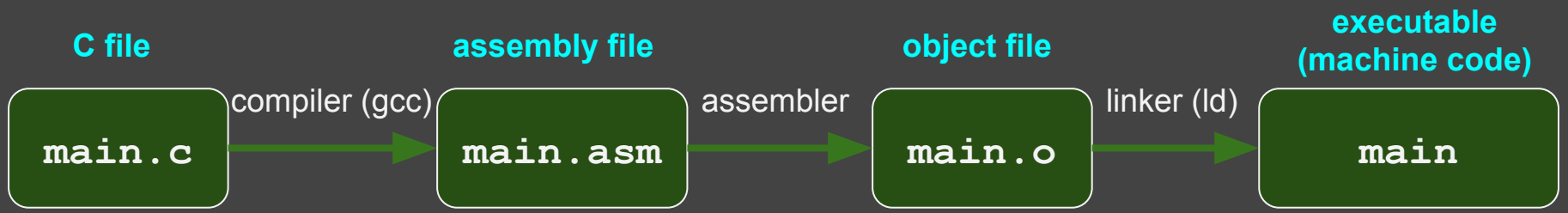


Assembly files?

K. N. Toosi
University of Technology

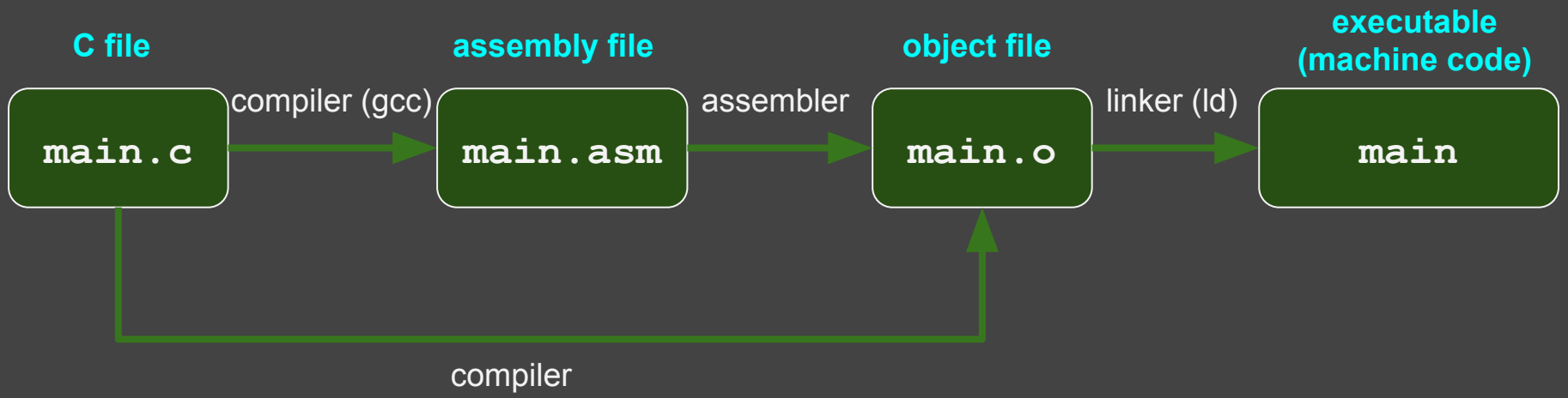
high-level to low-level compilation hierarchy

K. N. Toosi
University of Technology



high-level to low-level compilation hierarchy

K. N. Toosi
University of Technology



Assembling assembly files

K. N. Toosi
University of Technology

Our first assembly program (64 bit)

first.asm

```
segment    .data
msg:  db        "Salaaaaaaam!!", 10   ; message + newline character

global     _start

segment    .text
_start:                    ; entry point (linker needs this)

        mov    rax, 1      ; system call number (sys_write=1)
        mov    rdi, 1      ; file descriptor (stdout=1)
        mov    rsi, msg    ; message to write
        mov    rdx, 14     ; number of bytes to write

        syscall            ; invoke system call (sys_write)

        mov    rax, 60   ; system call number (sys_exit=60)
        mov    rdi, 0    ; exit code 0

        syscall          ; invoke system call (sys_exit)
```

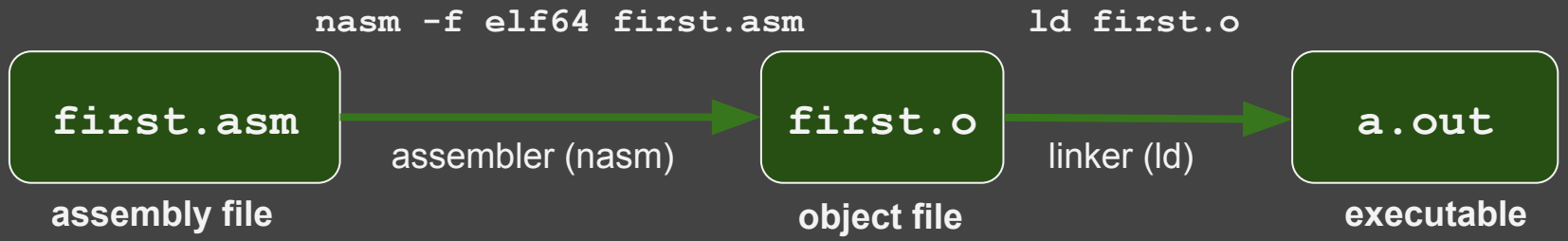
Our first assembly program (64 bit)

first.asm

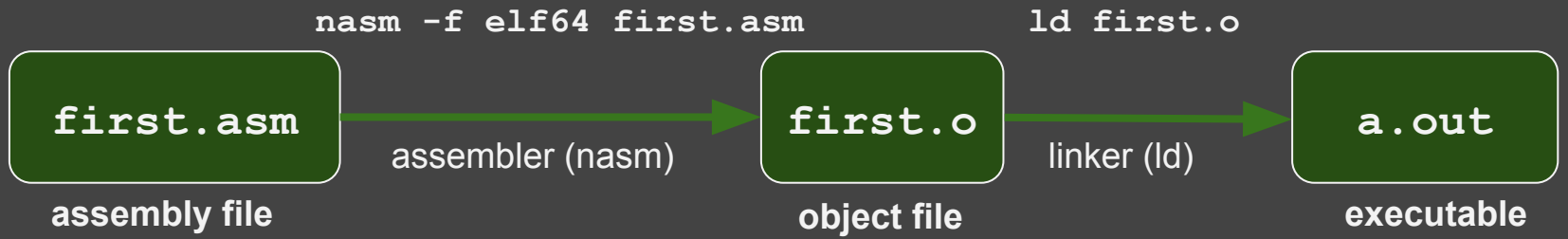
```
segment    .data
msg:  db         "Salaaaaaaam!!", 10   ; message + newline character

global     _start

segment    .text
_start:                    ; entry point (linker needs this)

        mov    rax, 1      ; system call number (sys_write=1)
        mov    rdi, 1      ; file descriptor (stdout=1)
        mov    rsi, msg    ; message to write
        mov    rdx, 14     ; number of bytes to write

        syscall

        mov    rax, 60     ; b.nasihatkon@kntu:lecture3$ nasm -f elf64 first.asm
        mov    rdi, 0      ; b.nasihatkon@kntu:lecture3$ ld first.o
                           ; b.nasihatkon@kntu:lecture3$ ./a.out
        syscall            ; Salaaaaaaam!!
                           ; invoke system call (sys_exit)
```

Assembling, linking and running (64 bit)

- **Assemble:** `nasm -f elf64 first.asm`
- **Link:** `ld first.o`
- **Execute:** `./a.out`



Assembling, linking and running (64 bit)



Our first assembly program (32 bit)

first.asm

```
segment .data
msg:  db        "Salaaaaaaam!!", 10   ; message + newline character

segment .text

global _start      ; make entry point visible

_start:             ; entry point (linker needs this)
        mov     eax, 4       ; (32 bit) system call number (sys_write=4)
        mov     ebx, 1       ; file descriptor (stdout=1)
        mov     ecx, msg     ; address of message to write
        mov     edx, 14      ; message length
        int     0x80         ; interrupt no. x80 = system call

        mov     eax,1        ; (32 bit) system call number (sys_exit=1)
        mov     ebx,0        ; exit code 0
        int     0x80         ; interrupt no. x80 = system call
```

Our first assembly program (32 bit)

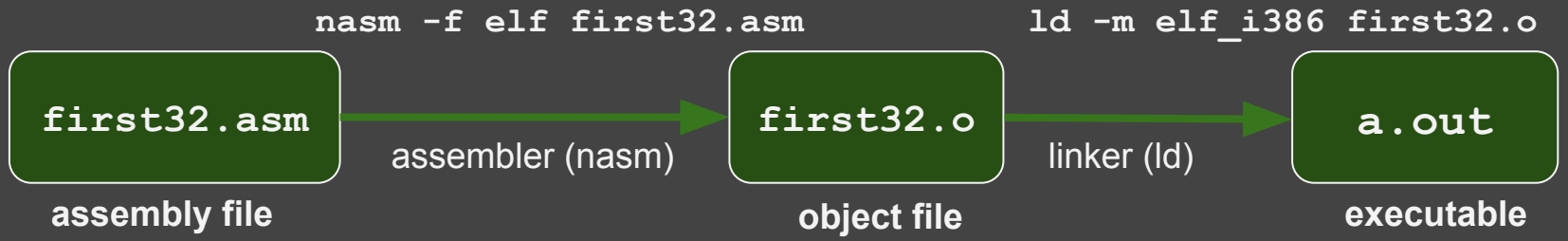
first.asm

```
segment .data
msg:  db        "Salaaaaaaam!!", 10   ; message + newline character

segment .text

global _start      ; make entry point visible

_start:              ; entry point (linker needs this)
        mov     eax, 4       ; (32 bit) system call number (sys_write=4)
        mov     ebx, 1       ; file descriptor (stdout=1)
        mov     ecx, msg     ; address of message to write
        mov     edx, 14      ; message length
        int     0x80

        mov     eax,1
        mov     ebx,0
        int     0x80         ; interrupt no. x80 = system call
```

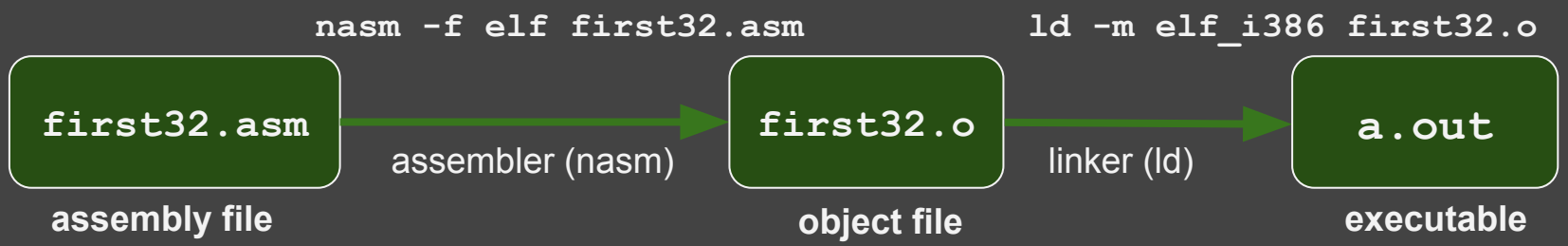
```
CS@kntu:lecture3$ nasm -f elf first32.asm
CS@kntu:lecture3$ ld -m elf_i386 first32.o
CS@kntu:lecture3$ ./a.out
Salaaaaaaam!!
```

Assembling, linking and running (32 bit)

- **Assemble:** `nasm -f elf first32.asm`
- **Link:** `ld -m elf_i386 first32.o`
- **Execute:** `./a.out`



Assembling, linking and running (32 bit)



Assembling, linking and running (32 bit)

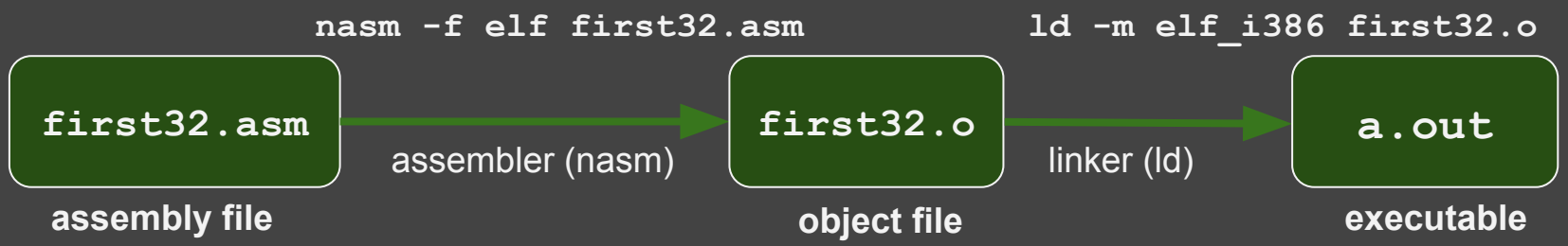
K. N. Toosi
University of Technology

linker

create 32 bit executable

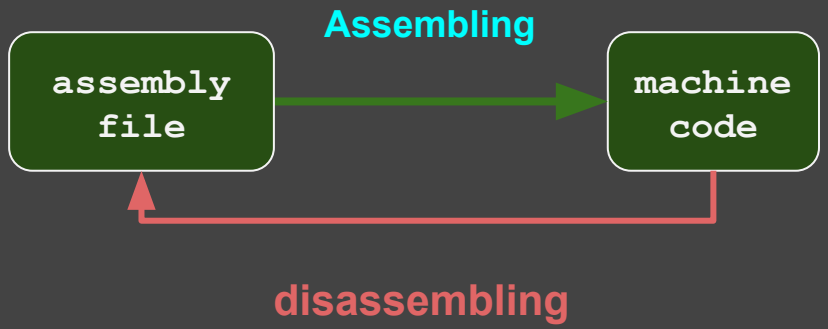
input object file (32 bit ELF format)

```
ld    -m elf_i386 first32.o
```



Disassembling

K. N. Toosi
University of Technology



Disassembling


```
CS@kntu:lecture3$ objdump -d -M intel a.out

a.out:     file format elf32-i386


Disassembly of section .text:

08048080 <_start>:
 8048080:       b8 04 00 00 00          mov    eax,0x4
 8048085:       bb 01 00 00 00          mov    ebx,0x1
 804808a:       b9 a4 90 04 08          mov    ecx,0x80490a4
 804808f:       ba 0e 00 00 00          mov    edx,0xe
 8048094:       cd 80                   int    0x80
 8048096:       b8 01 00 00 00          mov    eax,0x1
 804809b:       bb 00 00 00 00          mov    ebx,0x0
 80480a0:       cd 80                   int    0x80
```

Compiling and linking C files

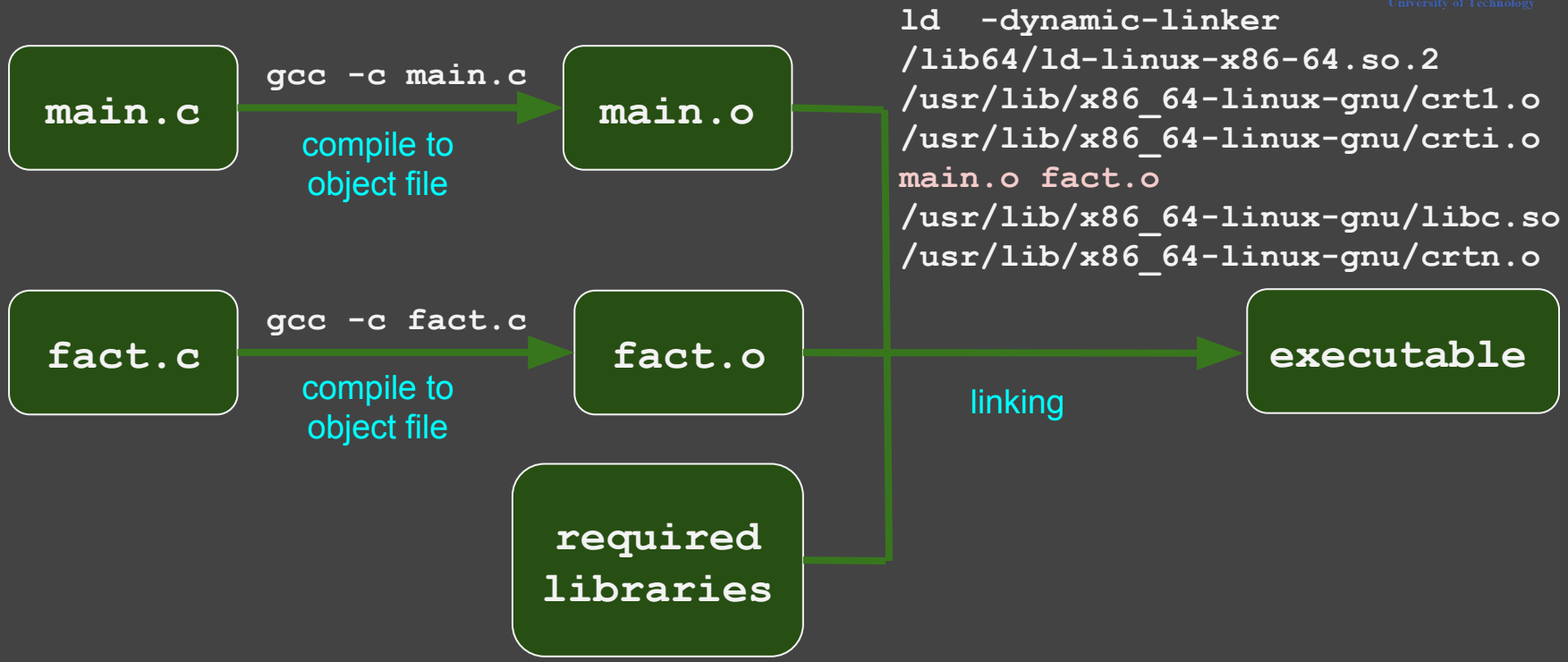
main.c

```c
#include <stdio.h>

int fact(int);

int main() {
  int x = 8;

  int u = printf("x!=%d\n", fact(x));

  return  0;
}
```

fact.c

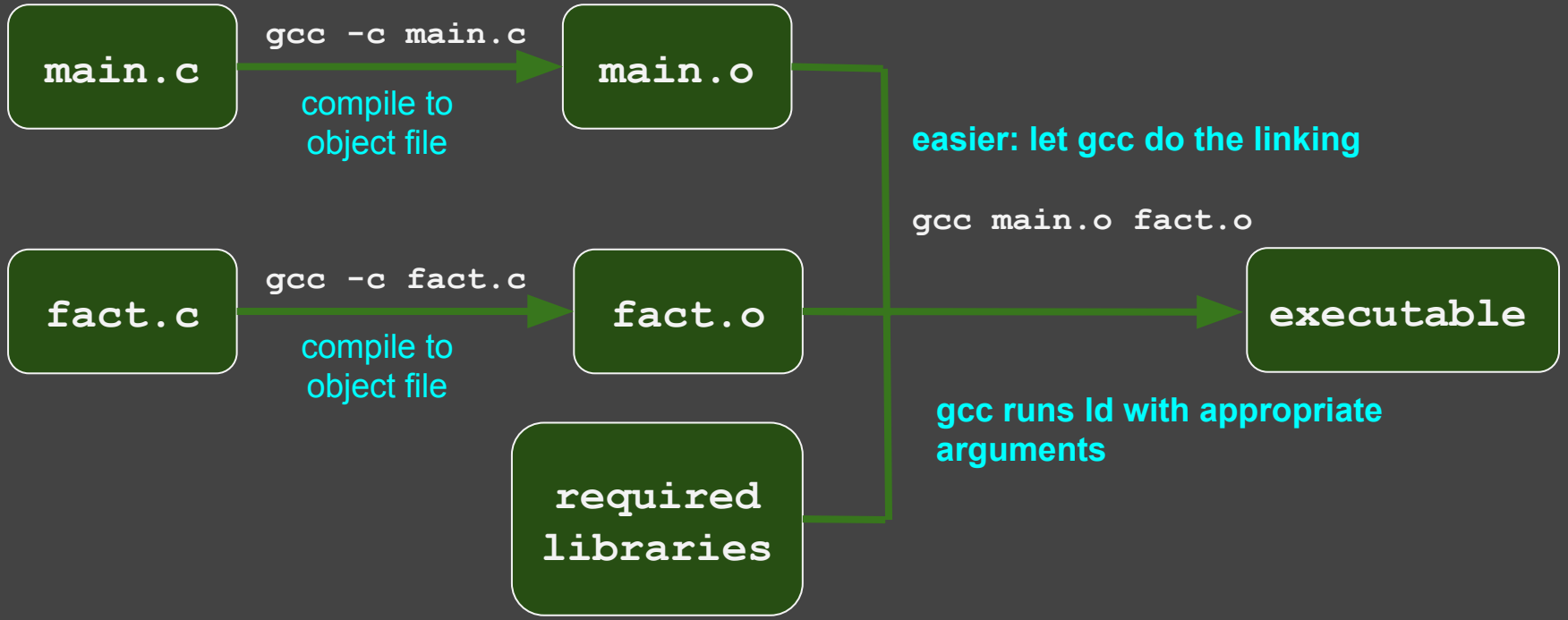
```c
int fact(int n) {
  return n == 0 ?  1 :  n * fact(n-1);
}
```

Compiling and linking C files

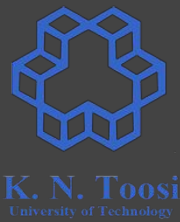


Compiling and linking C files

K. N. Toosi
University of Technology



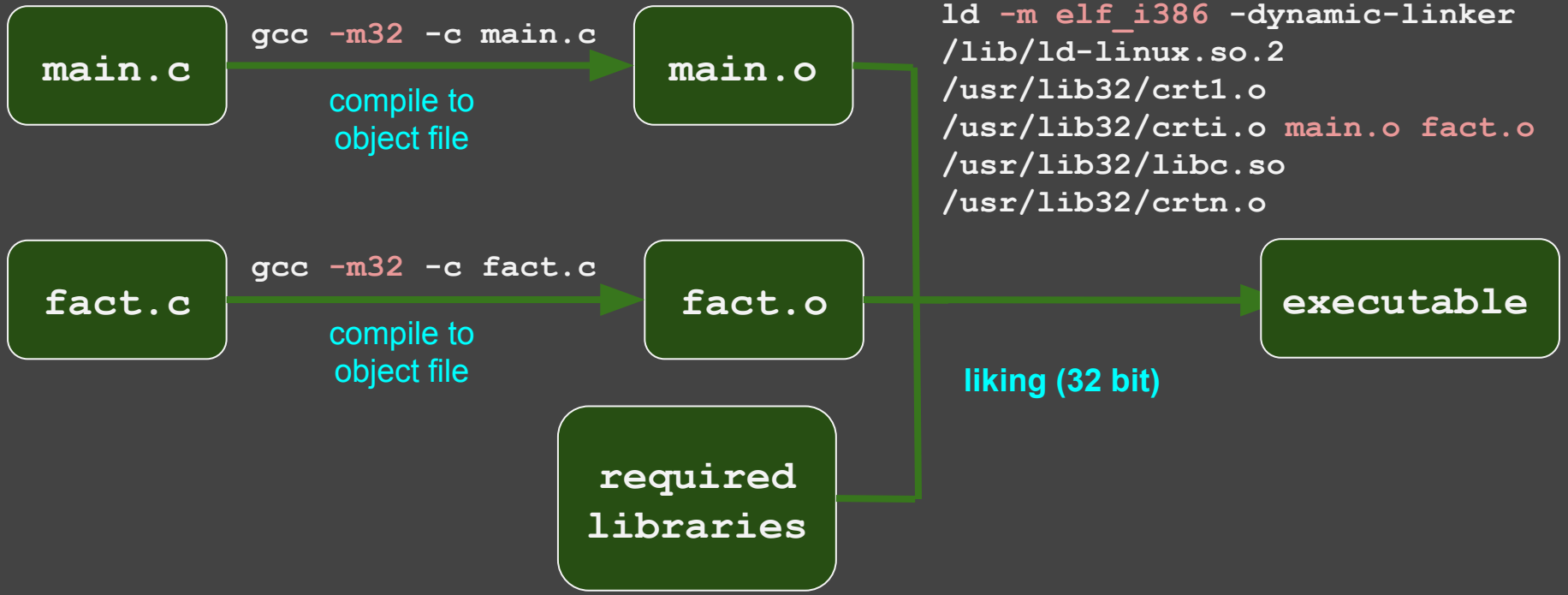
Compiling C files to 32 bit programs



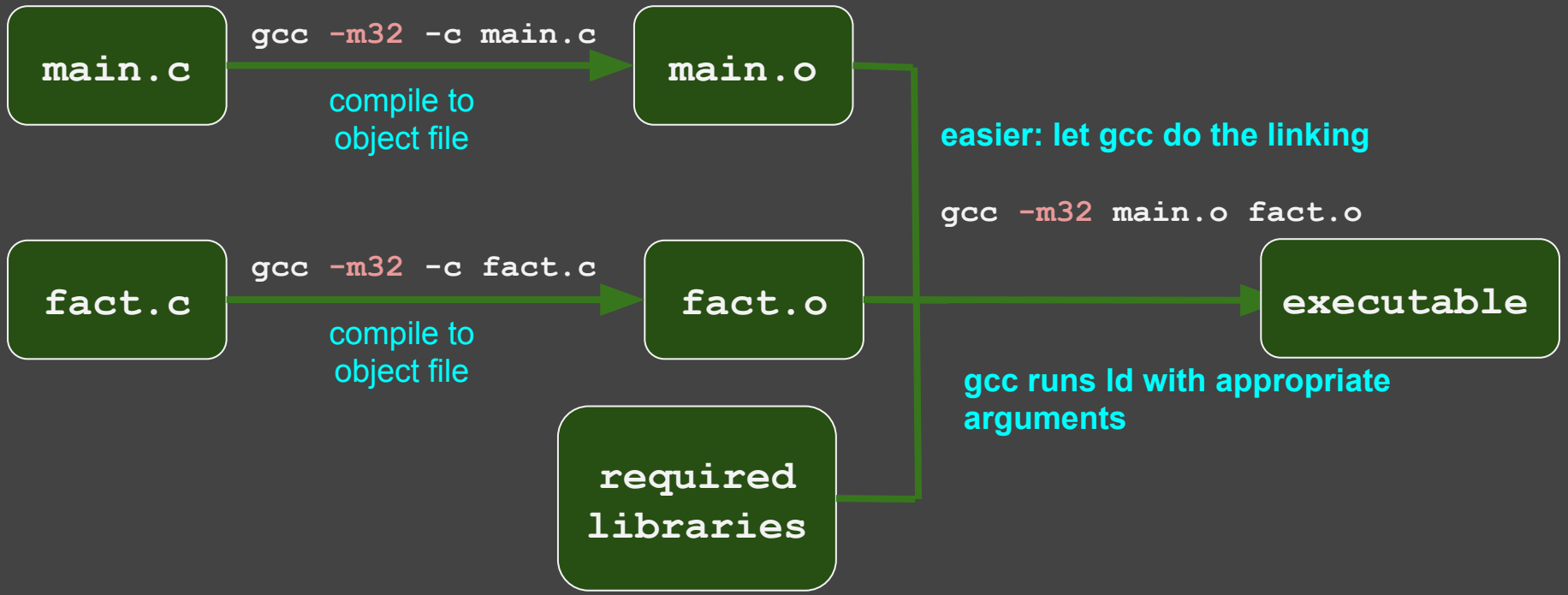
Compile to 32 bit executable on a 64 bit system:

- First, install the 32 bit libraries:
  - `sudo apt-get install libc6-dev-i386`
  - `sudo apt-get install libx32gcc-4.8-dev`
  - `sudo apt-get install gcc-multilib`
- You might need to install:
  - `sudo apt-get install gcc-6-multilib`

32 bit Compiling and linking C files



32 bit Compiling and linking C files



Our second assembly program!

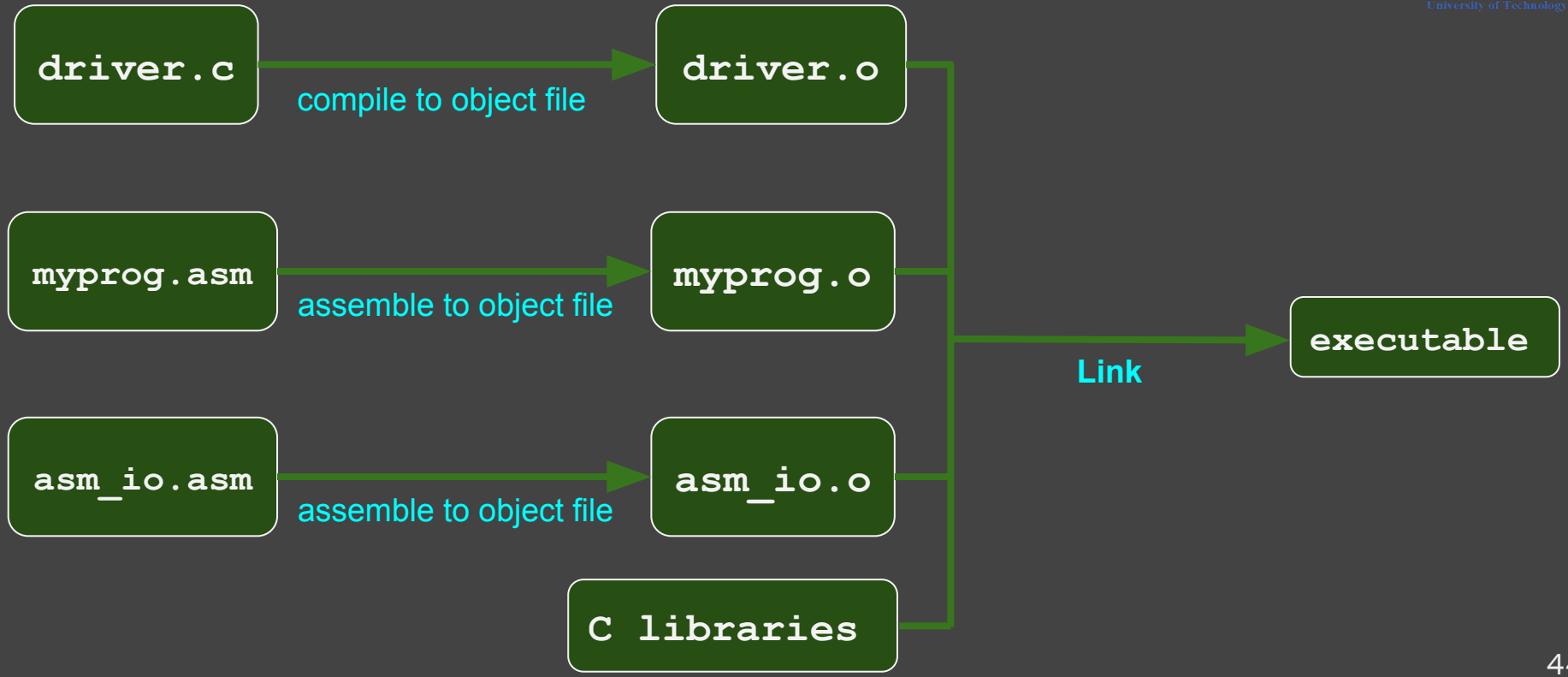
- We mostly do 32 bit assembly
- We use the functions/macros from the book (Carter, PC Assembly Language, 2007) for IO

I/O functions and macros from the book

`call print_int`	prints EAX as singed integer
`call print_char`	prints the character whose ascii code is stored in AL
`call print_string`	prints the string whose starting address is stored in EAX, string must be null-terminated (C-string)
`call print_nl`	prints a newline character
`call read_int`	reads an integer from standard input and stores it into EAX
`call read_char`	read a character from standard input and stores its ascii code in EAX
`dump_regs <num>`	(MACRO) prints out values of registers and flags (<num> is some number like 12 making debugging easier)

Carter, *PC Assembly Language*, 2007.

Our second assembly program!



steps to run our program

0. Install the Netwide Assembler
   - `sudo apt install nasm`
1. Download the example files (including IO files) from the books website: http://pacman128.github.io/pcasm/
   - for linux click on the link linux_examples to download the files
   - there are links for other platforms as well
2. Copy the files `asm_io.inc`, `asm_io.asm` and `cdecl.h` to your current working directory.
3. Compile the file asm_io.asm to object file (creating `asm_io.o`)
   - `nasm -f elf -d ELF_TYPE asm_io.asm`
4. Create a file named `driver.c`

steps to run our program

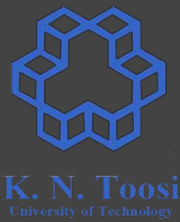
4. Create a file named `driver.c` simply calling an assembly function:

driver.c

```
void asm_main();

int main() {

    asm_main();

    return 0;
}
```

(alternatively, copy the files `driver.c` and `cdecl.h` to your current directory.)

steps to run our program



5. Install the 32 bit C libraries (if not installed)
   - `sudo apt-get install libc6-dev-i386`
   - `sudo apt-get install libx32gcc-4.8-dev`
   - `sudo apt-get install gcc-multilib` **(if needed)**
   - `sudo apt-get install gcc-6-multilib` **(if needed)**

6. Compile driver.c to 32 bit object file (creating driver.o)
   - `gcc -m32 -c driver.c`

steps to run our program

7. Create your main assembly file containing the `asm_main` label

myprog.asm

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
     enter 0,0
     pusha

     mov eax, 100
     mov ebx, 20
     sub eax, ebx

     call print_int ; print EAX
     call print_nl  ; print a new line

     dump_regs 1111 ; printing the system state (registers, etc.)

     popa
     leave
     ret
```

steps to run our program

7. Create your main assembly file containing the `asm_main` label

myprog.asm

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
        enter 0,0
        pusha

        mov eax, 100
        mov ebx, 20
        sub eax, ebx

        call print_int      ; print EAX
        call print_nl       ; print a new line

        dump_regs 1111      ; print registers, etc

        popa
        leave
        ret
```

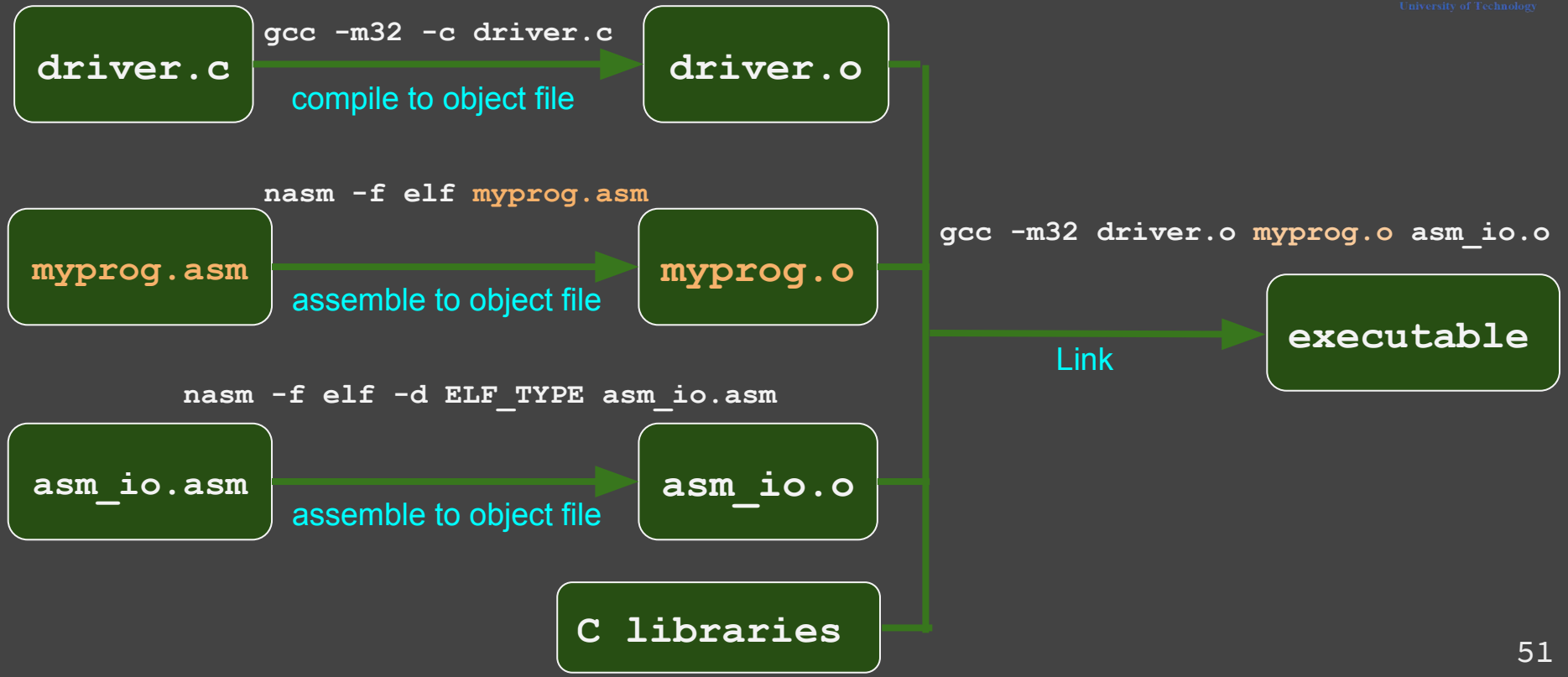
driver.c

```
void asm_main();

int main() {

    asm_main();

    return 0;
}
```

steps to run our program

8. Assemble your assembly code to 32 bit object file (creating `myprog.o`)
   - `nasm -f elf myprog.asm`
9. link the object files `myprog.o`, `asm_io.o`, `driver.o` (and the C libraries) to create the executable
   - `gcc -m32 driver.o myprog.o asm_io.o`
10. run the executable and see the output
    - `./a.out`

steps to run our program



steps to run our program

myprog.asm

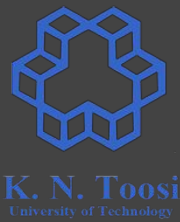
```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
     enter 0,0
     pusha

     mov eax, 100
     mov ebx, 20
     sub eax, ebx

     call print_int   ; print
     call print_nl    ; print

     dump_regs 1111   ; print

     popa
     leave
     ret
```

driver.c

```
void asm_main();

int main() {

     asm_main();

     return 0;
}
```

```
nasihatkon@kntu:lecture3$ nasm -f elf -d ELF_TYPE asm_io.asm
nasihatkon@kntu:lecture3$ gcc -m32 -c driver.c
nasihatkon@kntu:lecture3$ nasm -f elf myprog.asm
nasihatkon@kntu:lecture3$ gcc -m32 driver.o myprog.o asm_io.o
nasihatkon@kntu:lecture3$ ./a.out
80
Register Dump # 1111
EAX = 00000050 EBX = 00000014 ECX = FFDBFCB0 EDX = FFDBFCD4
ESI = F76B7000 EDI = F76B7000 EBP = FFDBFC88 ESP = FFDBFC68
EIP = 080484EB FLAGS = 0206                  PF
```

All commands




```
nasm -f elf -d ELF_TYPE asm_io.asm

gcc -m32 -c driver.c

nasm -f elf myprog.asm

gcc -m32 driver.o myprog.o asm_io.o

./a.out
```

Give output file a name (-o option)


```
nasm -f elf -d ELF_TYPE asm_io.asm

gcc -m32 -c driver.c

nasm -f elf myprog.asm

gcc -m32 driver.o myprog.o asm_io.o -o myprog

./myprog
```

Writing your own program

- Take the same steps as above
- You source file should be like this (or use `skel.asm` from the book)

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
      enter 0,0
      pusha

      ; write your assembly code here!


      popa
      leave
      ret
```

References

- Carter, Paul A. PC Assembly Language, 2007
- http://cs.lmu.edu/~ray/notes/nasmtutorial/
- https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.performance/when_dyn_linking_static_linking.htm