C Programming Part 6: C Compilation

ECEN 330: Introduction to Embedded Programming

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Why should you learn about the C compilation process?

1. You'll need to compile C programs, so you need to understand these steps

2. It will make you a faster debugger.
   - If you understand why the compiler is giving you a certain error message, you can likely figure out why to fix your code much faster.

3. Understand why C is organized the way it is:
   - Why do we have header files?
   - Why do we have function prototypes (forward declarations)?
   - What should I put in my .h file versus my .c file?

You may have compiled multiple files together like this:

```
gcc myFile1.c myFile2.c myFile3.c
```

This gives can sometimes mislead new C programmers into believing C files are compiled in unison.

In reality, **every C file is compiled separately, and then “linked” together.**

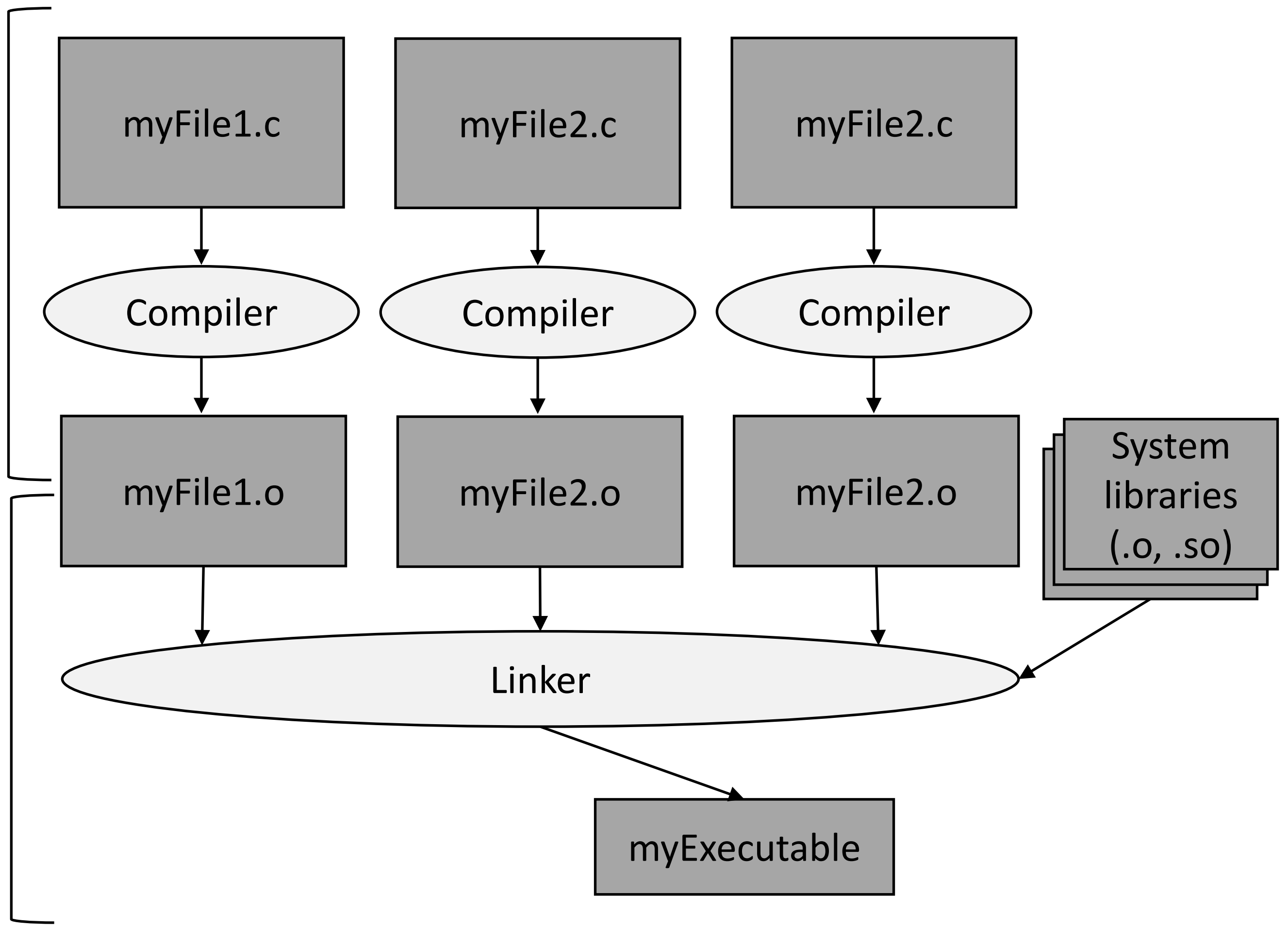
No “real” software projects are compiled using a single gcc command like above. Why?

Compiler & Linker

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

- Each C file is compiled separately to generate an object (.o) file
- These files are then linked together, along with system libraries, to create an executable.
- It can be confusing because we use "gcc" to run both the **compiler** and the **linker.**
    - This is actually a helper program that determines which tool to use based on the input files.
        - Compiler (cc)
        - Linker (ldd)

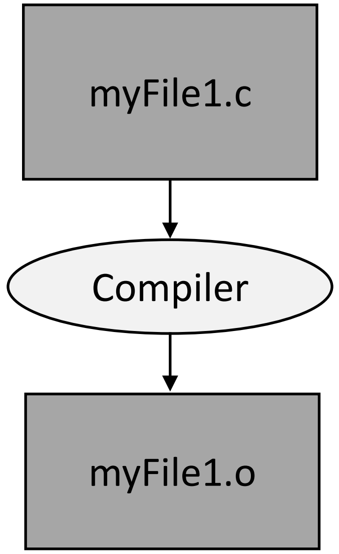
```
gcc -c myFile1.c
```



```
gcc myFile1.o myFile2.o myFile3.o -o myExecutable
```

C Compiler

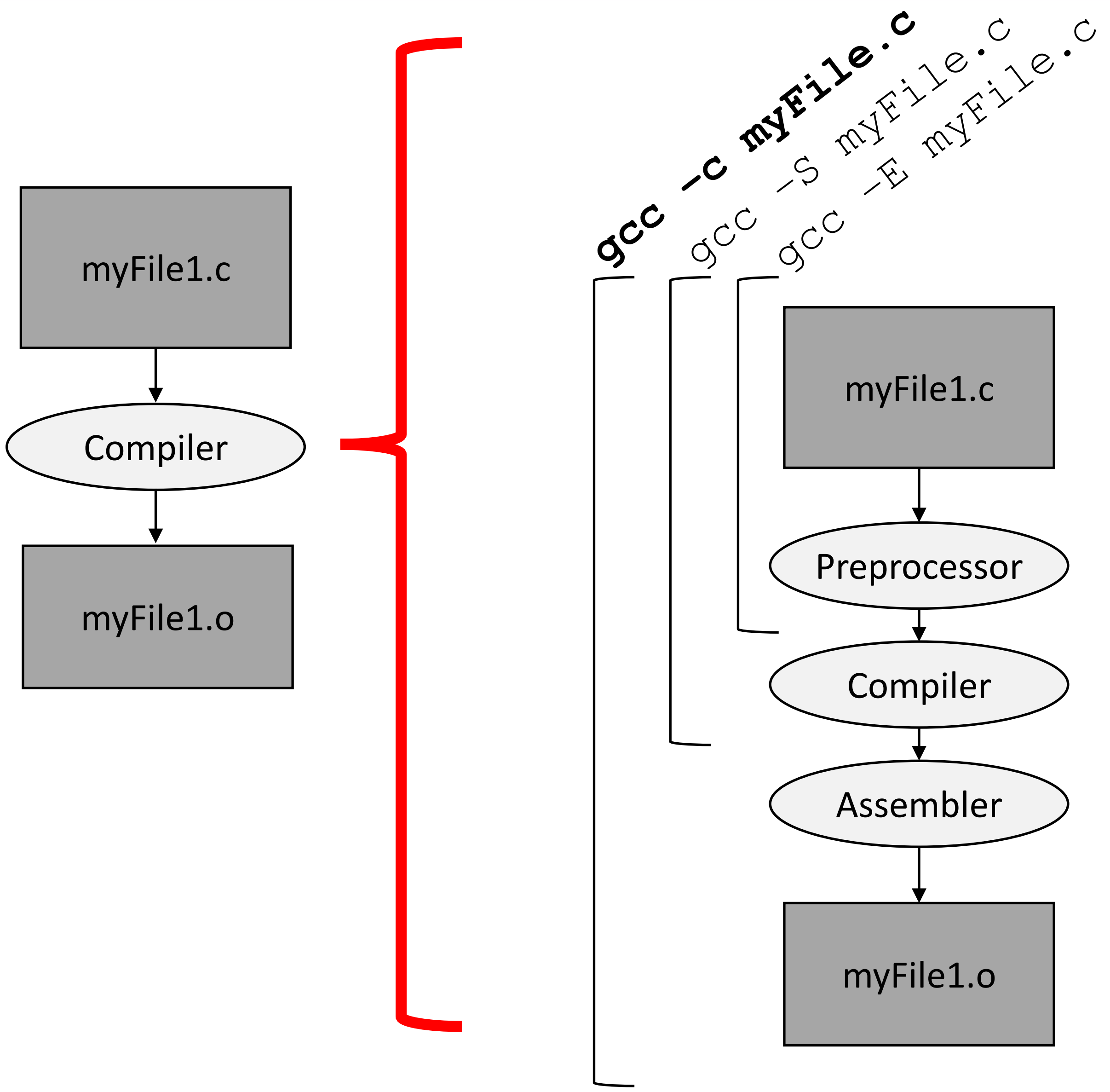
- The compiler takes human-readable C code, and translates it into object byte code.
- These are computer instructions encoded into binary, specific to the type of processor you are using.
- The byte code is **incomplete**, as it doesn't include code from the other files in your program yet.
- Example: If you call *printf* inside myFile1.c the object file will have a symbolic reference to the function, but it won't know where it is yet.



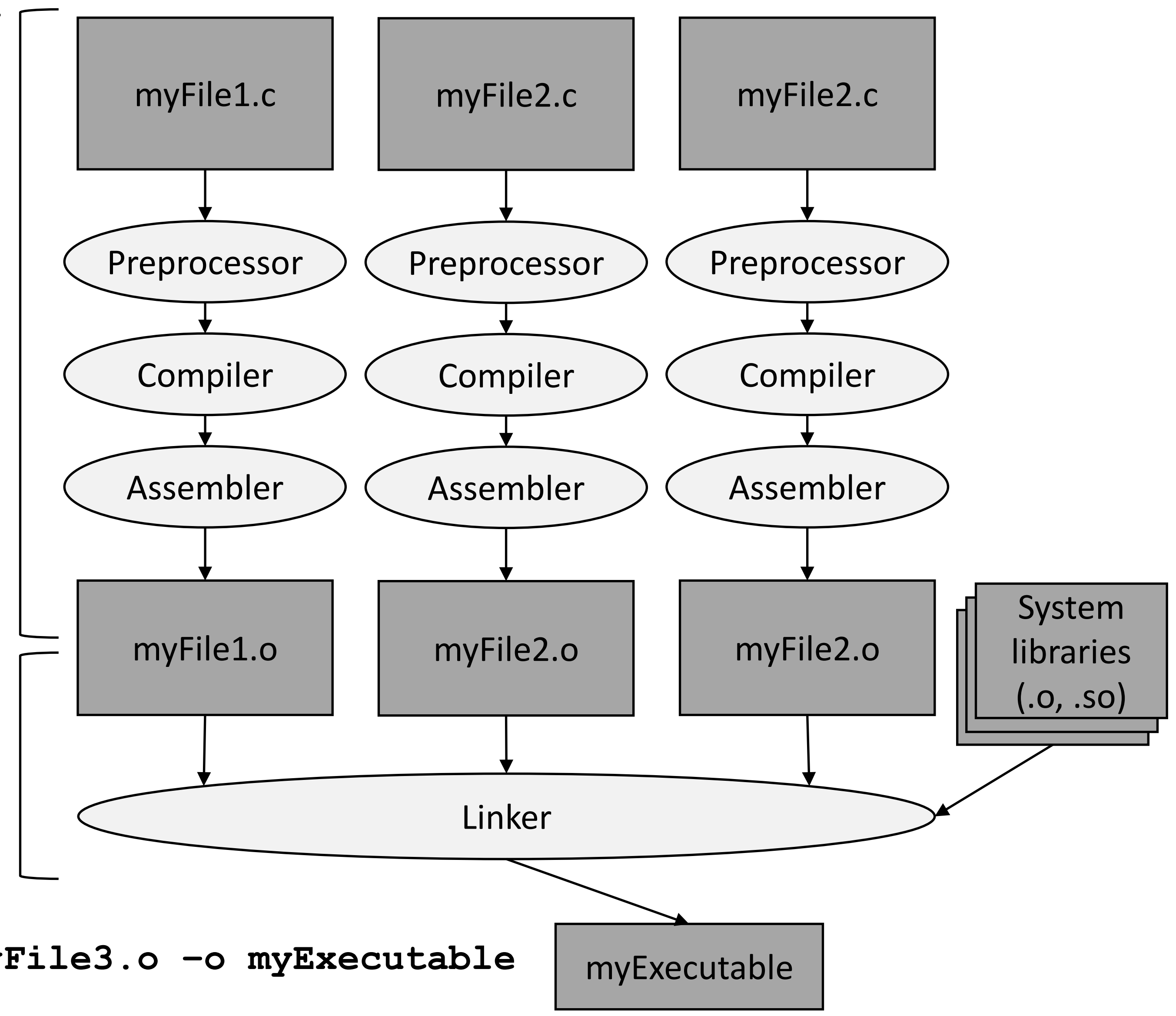
Breaking down Compilation

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

- The compilation step is actually made up of several sub-steps
- It's not common to run these partial steps
  - Usually you just compile .c files into .o files
  - But we will look inside the intermediate steps to see what it looks like



```
gcc -c myFile.c
```



```
gcc myFile1.o myFile2.o myFile3.o -o myExecutable
```

C Compilation

Each C File is compiled:

- From start to finish in one pass
- Without any knowledge of what is in other C files

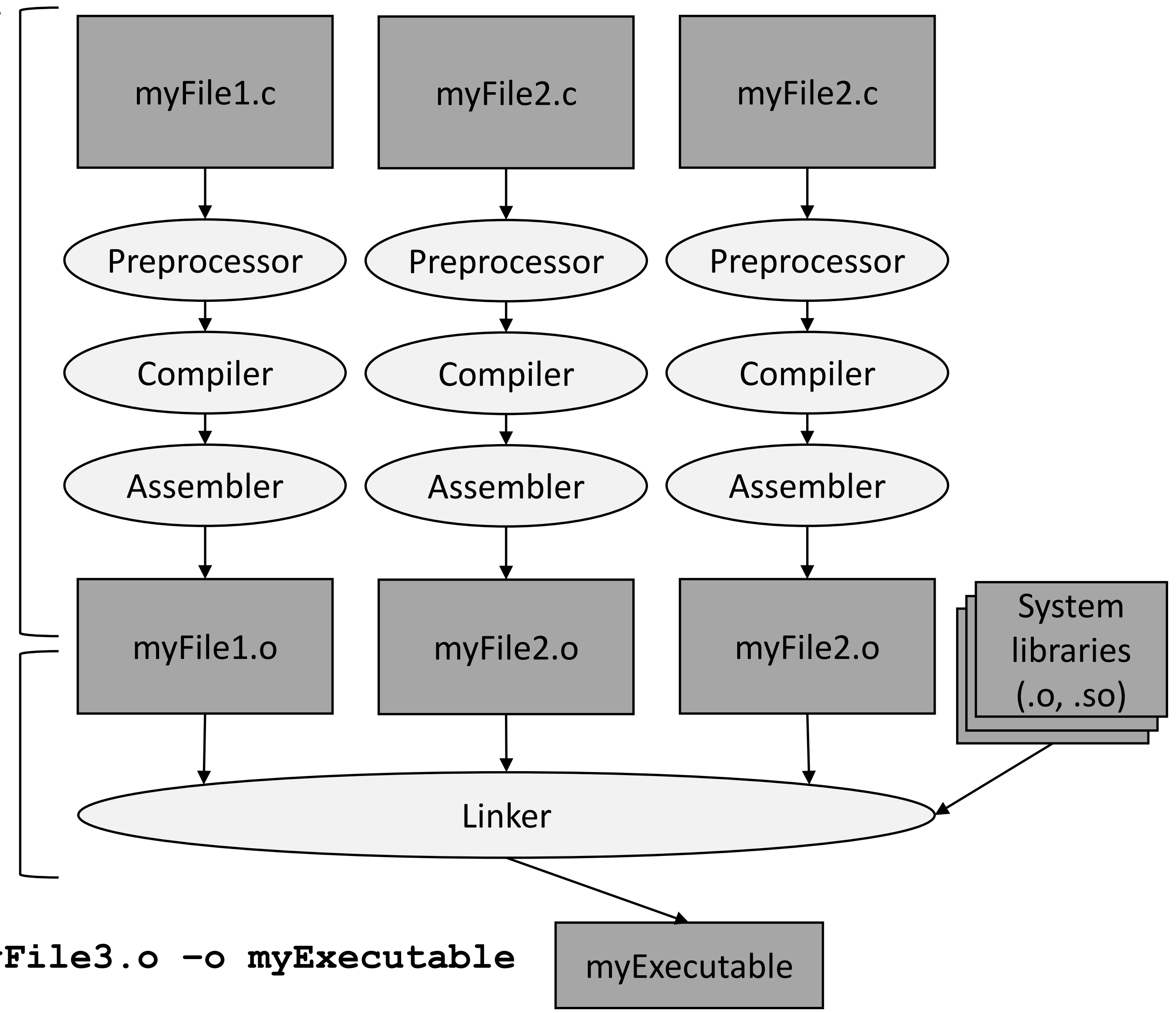
Why?

- You can compile big projects fast
- Multiple files can be compiled at the same time.

So how do you know what is available in other files?

- Header files provide declarations of:
  - Functions
  - Global variables
  - Types, Strcuts, etc.

```
gcc –c myFile.c
```



```
gcc myFile1.o myFile2.o myFile3.o –o myExecutable
```

Linker

- The linker stitches together the different object files.

- It replaces symbolic references from object files to the actual location of where a function, variable, etc. resides in another object file.

- The byte code is combined together into a single executable.

- The linker will throw an error if:
  1. It can't find a definition for something you have referenced
     - "undefined reference to _______________"
  2. There are too many (multiple) definitions for something you have referenced
     - "multiple definition of ______________"

How does this help you debug?

- If you see the linker error "undefined reference to ________________"
  - A function you are calling is missing from another file (maybe a typo?)
  - You forgot to compile the file that has that function?

- If you see the linker error "multiple definition of ________________"
  - You've declared the same function or global variable name in multiple files.
  - Either change one name, or use *static*.

- Other errors are compiler errors and usually mean the problem is contained to the .c file in question (or any .h files it #includes)

- If you see "implicit function" warning you know that shouldn't be ignored, and you are missing a function prototype, either:
  - At the top of your .c file for functions in the same file
  - In a .h file for functions in other files