

# Embedded Software Essentials

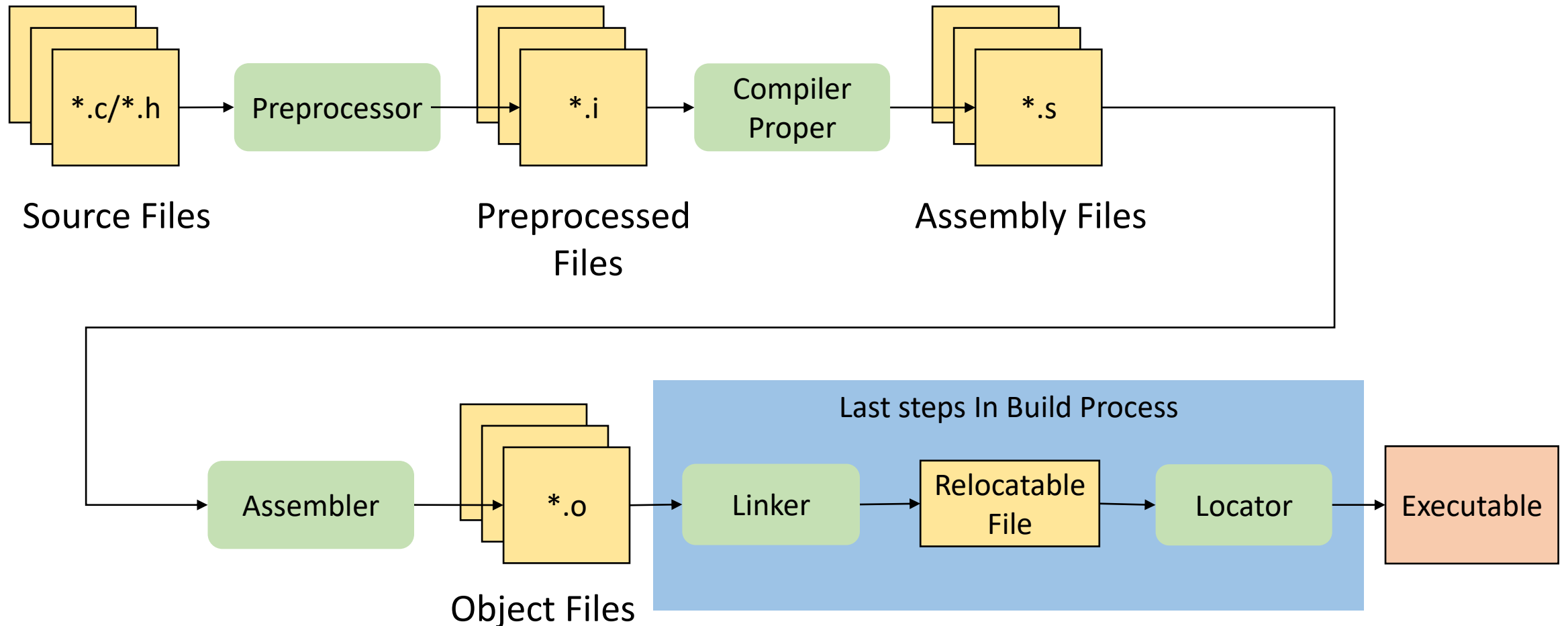
*Linkers*

**C1 M2 V5**

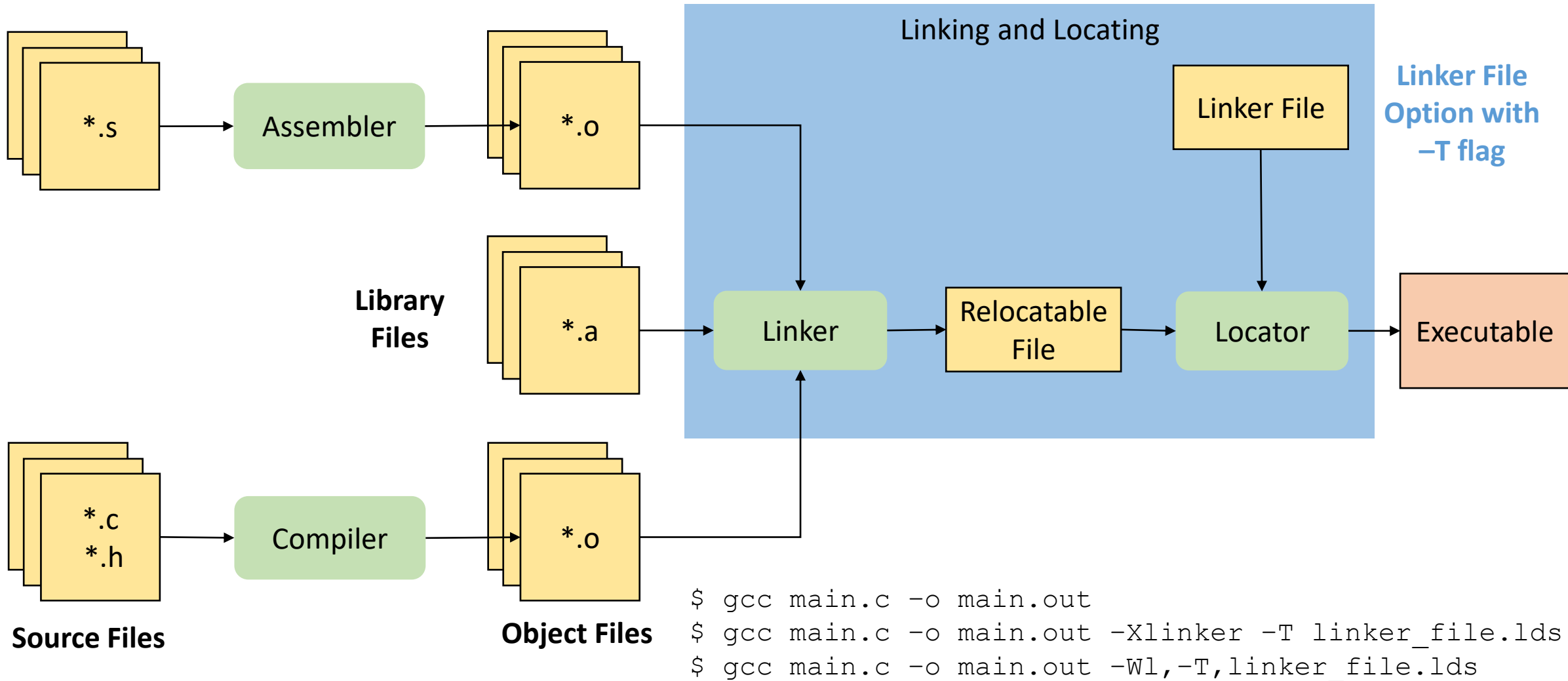


Copyright

# Linking and Locating [S1]

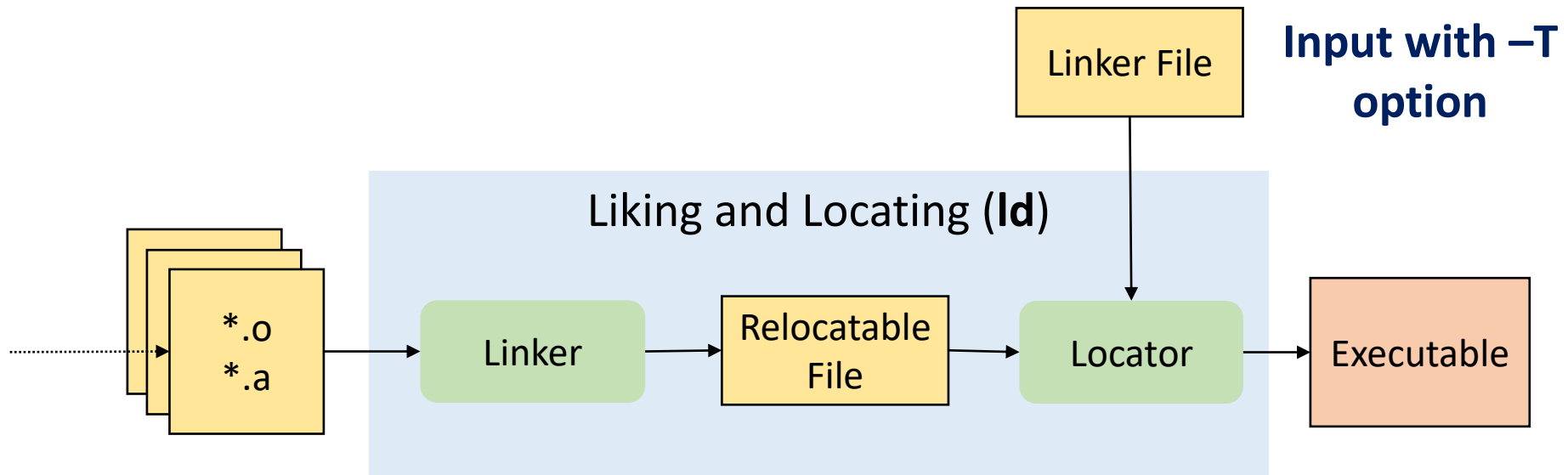


# Typical Build Process [S2]



# Linkers [S3a]

- Combines all of objects files into a single executable
  - Object code uses **symbols** to reference other functions/variables



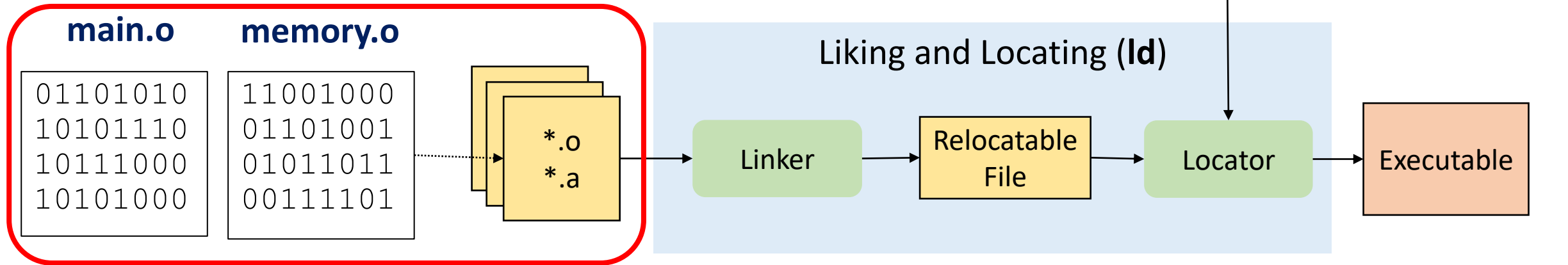
Invoke the linker indirectly from compiler (and with no options)

```
$ gcc -o main.out main.c
```

# Linkers [S3b]

- Combines all of objects files into a single executable
  - Object code uses **symbols** to reference other functions/variables

Cannot be executed



Invoke the linker indirectly from compiler (and with no options)

```
$ gcc -o main.out main.c
```

# Linking Object Files [S4a]

Three source files (\*.h & \*.c)

Must convert \*.c files into object code

## main.c

```
#include "memory.h"
int main(){
    char arr[10];
    memzero(arr, 10);
    return 0;
}
```

## memory.h

```
#include "memory.h"
char memzero(char * src, int length){
    int i;
    for(i = 0; i < length; i++){
        *src++ = 0;
    }
}
```

## memory.h

```
#ifndef __MEMORY_H__
#define __MEMORY_H__
char memzero(char * src, int length);
#endif /* __MEMORY_H__ */
```

# Linking Object Files [S4b]

The object files have many **symbols** that need to be tracked and resolved

**main.c**

```
#include "memory.h"
int main(){
    char arr[10];
    memzero(arr, 10);
    return 0;
}
```

**memory.h**

```
#include "memory.h"
char memzero(char * src, int length){
    int i;
    for(i = 0; i < length; i++){
        *src++ = 0;
    }
}
```

**memory.h**

```
#ifndef __MEMORY_H__
#define __MEMORY_H__
char memzero(char * src, int length);
#endif /* __MEMORY_H__ */
```



# Linking Object Files [S4c]

After compilation, we have 2 object files (header file provide symbol reference)

Object files are NOT human readable

Symbol tables track important references

**main.o**

01101010
10101110
10111000
10101000

**main.o has  
references to  
symbols defined  
in memory.o**

**memory.o**

11001000
01101001
01011011
00111101

**memory.o has the  
definitions of  
these special  
symbols**

# Linking Object Files [S4d]

Function memmove is not defined in included files

Causes an error

**main.c**

```
#include <stdlib.h>
int main(){
    char a[10], b[10];
    memmove(a, b, 10);
    return 0;
}
```

???

**memory.h**

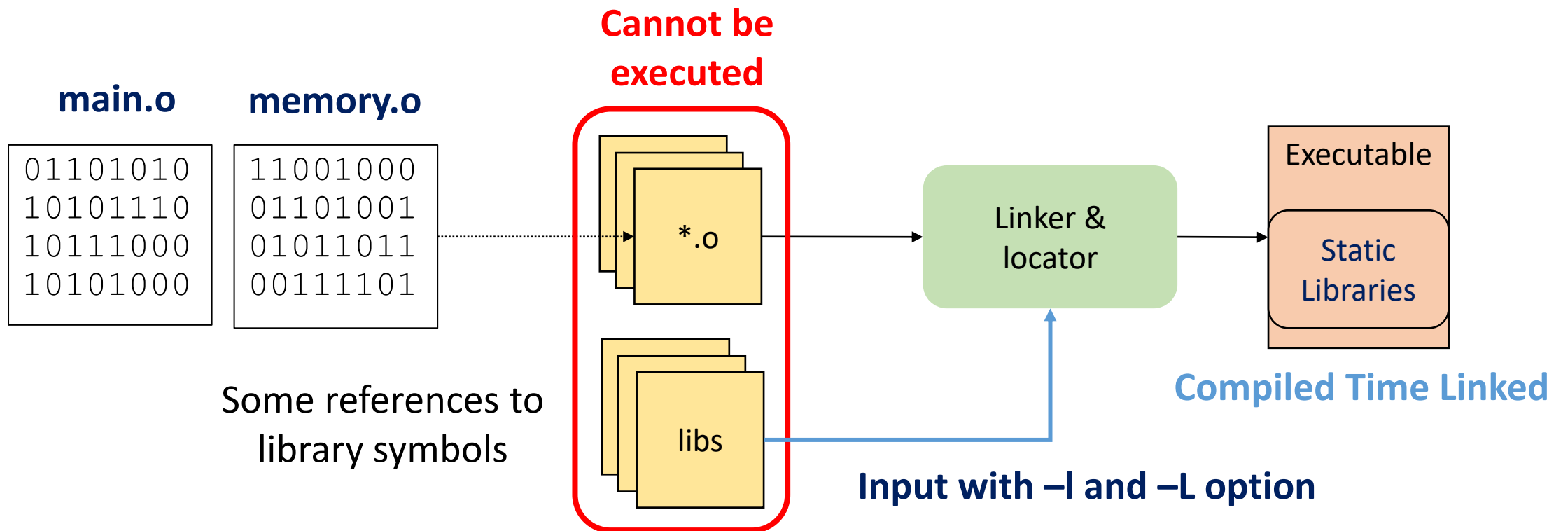
```
#include "memory.h"
char memzero(char * src, int length){
    int i;
    for(i = 0; i < length; i++){
        *src++ = 0;
    }
}
```

**memory.h**

```
#ifndef __MEMORY_H__
#define __MEMORY_H__
char memzero(char * src, int length);
#endif /* __MEMORY_H__ */
```

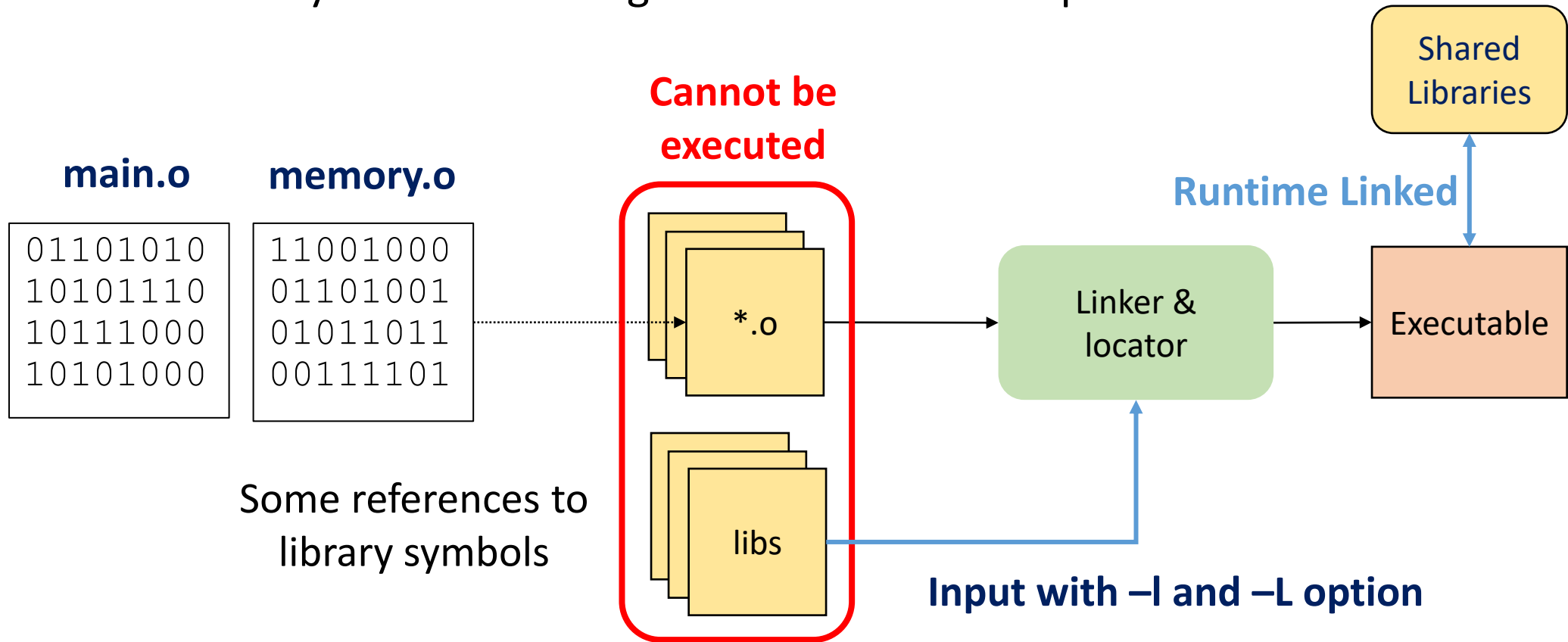
# Libraries [S6a]

- Linker must know **name** and **path** to library to link with it
  - Static and Dynamic libraries get linked at different points



# Libraries [S6b]

- Linker must know **name** and **path** to library to link with it
  - Static and Dynamic libraries get linked at different points



# Linking Object Files [S7]

Standard libraries can be statically or dynamically linked

Entry and exit points from main are included in a standard library that is automatically included by the linker

Can stop auto link of standard libs with **-nostdlib** flag

How do we enter main?

How do we exit or return from main?

**main.c**

```
#include <stdlib.h>
#include <stdio.h>
int main() {
    char arr[10];

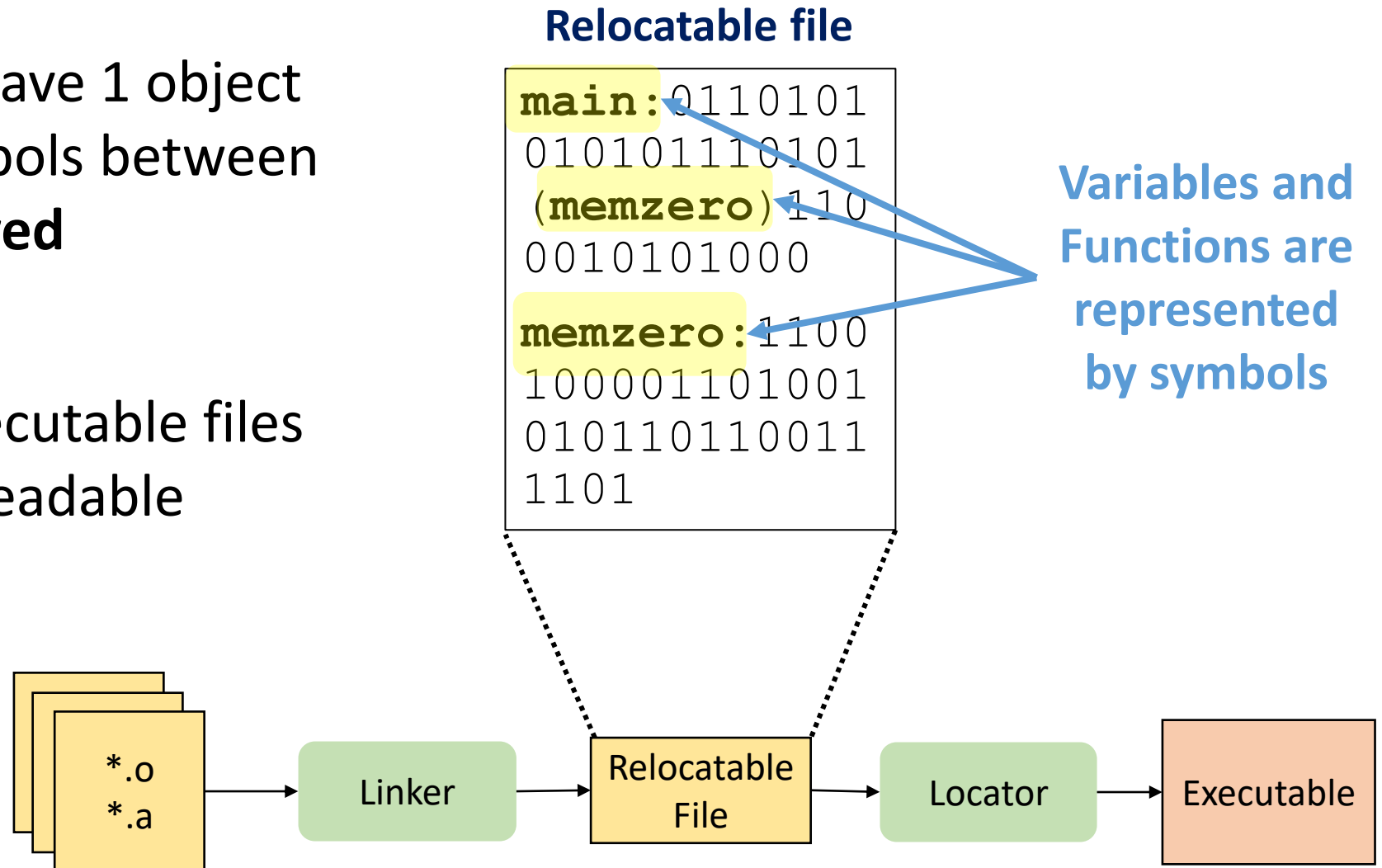
    printf("Hello World\n");

    return 0;
}
```

# Linking Object Files [S8]

After linking, we have 1 object files, and the symbols between the two are **resolved**

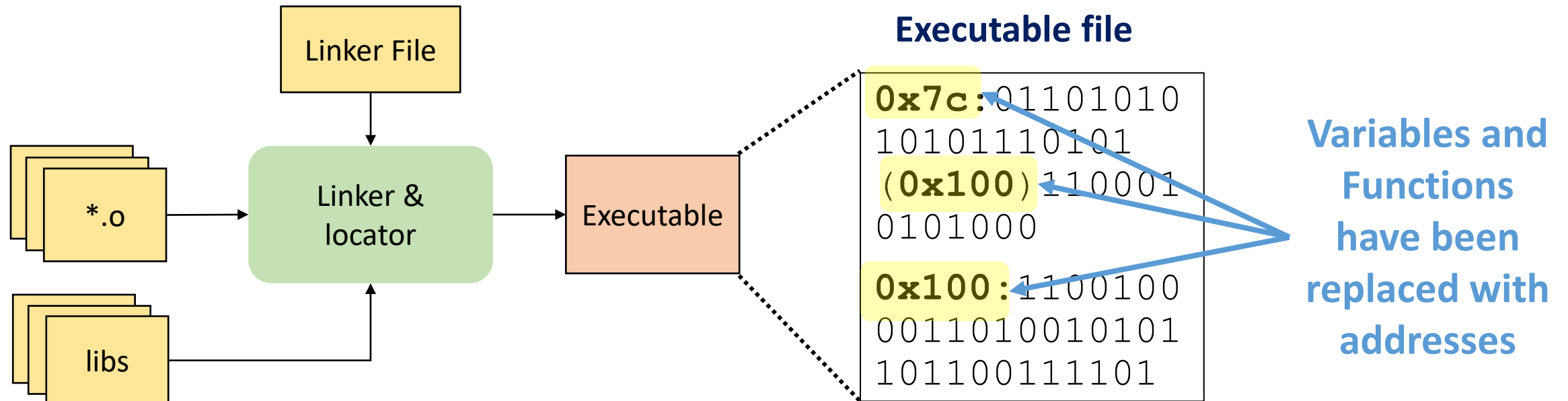
Relocatable & Executable files are NOT human readable



# Linking Object Files [S9]

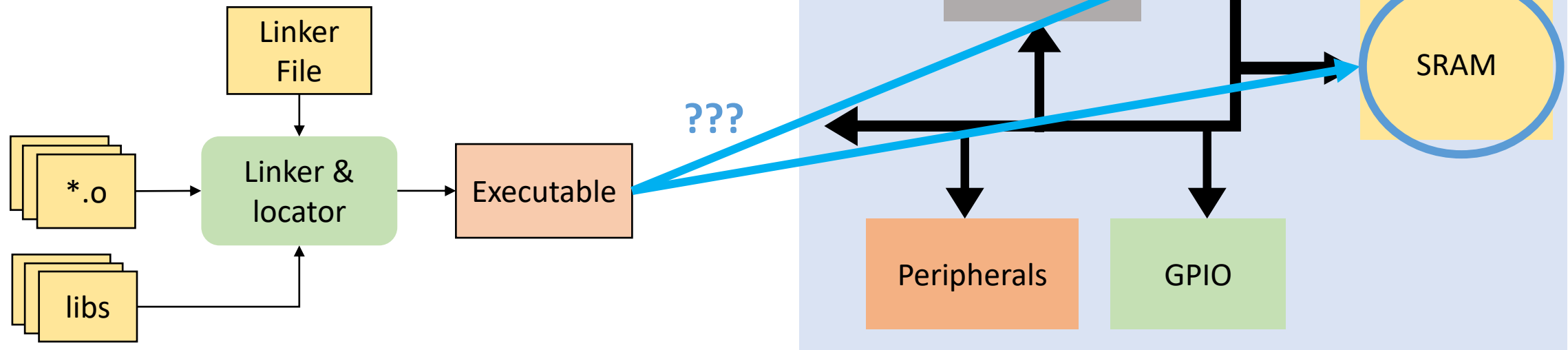
After locating, symbols are removed and direct **addresses** get assigned into the object code

The processor understands **machine code** (binary encoded instructions). These must have direct references to memory (addresses, not symbols)



# Linker Files [S10]

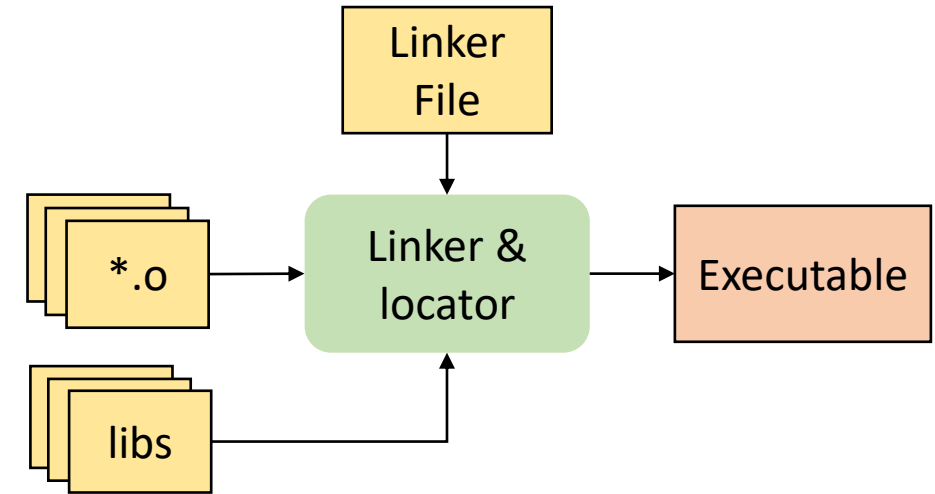
- Details on how to map compiled data into physical memory regions





# Linker Scripts Details [S11]

- Code sections to memory regions map
- Start and Sizes of memory regions
- Access attributes of memory regions
- Report checking for over-allocation
- Entry points of the program



## Example Memory Regions:

RAM/SRAM  
FLASH (MAIN)  
EEPROM  
VECTORS  
BOOTLOADER

## Example code/data sections:

.bss  
.data  
.text  
.isr\_vectors  
.heap

Entry Point Example:  
`ENTRY(Reset_Handler)`

# Example Linker Script Contents [S12a]

## MEMORY

```
{  
    MAIN          (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA     (RW) : origin = 0x20000000, length = 0x00010000  
}
```

## Physical Memory Regions

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text : > MAIN  
    .const : > MAIN  
    .cinit : > MAIN  
    .pinit : > MAIN  
    .data : > SRAM_DATA  
    .bss : > SRAM_DATA  
    .heap : > SRAM_DATA  
    .stack : > SRAM_DATA (HIGH)  
}
```

## Compiled Memory Sections

# Example Linker Script Contents [S12b]

```
MEMORY
{
    MAIN          (RX) : origin = 0x00000000, length = 0x00040000
    SRAM_DATA     (RW) : origin = 0x20000000, length = 0x00010000
}
```

## Physical Memory Regions

**Each “code” section output from compilation is then mapped into memory regions**

## SECTIONS

```
{
    .intvecs: > 0x00000000
    .text   : > MAIN
    .const  : > MAIN
    .cinit  : > MAIN
    .pinit  : > MAIN
    .data   : > SRAM_DATA
    .bss    : > SRAM_DATA
    .heap   : > SRAM_DATA
    .stack  : > SRAM_DATA (HIGH)
}
```

## Compiled Memory Sections

# Example Linker Script Contents [S12c]

## MEMORY

```
{  
    MAIN (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

## Physical Memory Regions

**Specifies the location the compiled region should map into physical memory**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text : > MAIN  
    .const : > MAIN  
    .cinit : > MAIN  
    .pinit : > MAIN  
    .data : > SRAM_DATA  
    .bss : > SRAM_DATA  
    .heap : > SRAM_DATA  
    .stack : > SRAM_DATA (HIGH)  
}
```

## Compiled Memory Sections

# Example Linker Script Contents [S12c]

## MEMORY

```
{  
    MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

### Physical Memory Regions

**Specifies the location the compiled region should map into physical memory**



**This is the “relocating” that the locator does**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text   : > MAIN  
    .const  : > MAIN  
    .cinit  : > MAIN  
    .pinit  : > MAIN  
    .data   : > SRAM_DATA  
    .bss    : > SRAM_DATA  
    .heap   : > SRAM_DATA  
    .stack  : > SRAM_DATA (HIGH)  
}
```

### Compiled Memory Sections

# Example Linker Script Contents [S12d]

## MEMORY

```
{  
    MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

### Physical Memory Regions

**Specifies the location the compiled region should map into physical memory**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text : > MAIN  
    .const : > MAIN  
    .cinit : > MAIN  
    .pinit : > MAIN  
    .data : > SRAM_DATA  
    .bss : > SRAM_DATA  
    .heap : > SRAM_DATA  
    .stack : > SRAM_DATA (HIGH)  
}
```

### Compiled Memory Sections

# Example Linker Script Contents [S12e]

## MEMORY

```
{  
    MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

### Physical Memory Regions

**Specifies the start address and length of the region for the memory map (in bytes)**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text   : > MAIN  
    .const  : > MAIN  
    .cinit  : > MAIN  
    .pinit  : > MAIN  
    .data   : > SRAM_DATA  
    .bss    : > SRAM_DATA  
    .heap   : > SRAM_DATA  
    .stack  : > SRAM_DATA (HIGH)  
}
```

### Compiled Memory Sections

# Example Linker Script Contents [S12e]

- Linker file can calculate memory segments
  - Can throw an errors if memory space is invalid

```
HEAP_SIZE  = DEFINED(__heap_size__)  ? __heap_size__  : 0x0400;
STACK_SIZE = DEFINED(__stack_size__) ? __stack_size__ : 0x0800;

__StackTop  = ORIGIN(SRAM_DATA) + LENGTH(SRAM_DATA);
__StackLimit = __StackTop - STACK_SIZE;
ASSERT(__StackLimit >= __HeapLimit, "Region SRAM_DATA overflowed!")
```



# Example Linker Script Contents [S12c]

## MEMORY

```
{  
    MAIN (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

## Physical Memory Regions

**Specifies the location the compiled region should map into physical memory**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text : > MAIN  
    .const : > MAIN  
    .cinit : > MAIN  
    .pinit : > MAIN  
    .data : > SRAM_DATA  
    .bss : > SRAM_DATA  
    .heap : > SRAM_DATA  
    .stack : > SRAM_DATA (HIGH)  
}
```

## Compiled Memory Sections

# Example Linker Script Contents [S12f]

## MEMORY

```
{  
    MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

### Physical Memory Regions

**Specifies the access properties of the region**

## SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text   : > MAIN  
    .const  : > MAIN  
    .cinit  : > MAIN  
    .pinit  : > MAIN  
    .data   : > SRAM_DATA  
    .bss    : > SRAM_DATA  
    .heap   : > SRAM_DATA  
    .stack  : > SRAM_DATA (HIGH)  
}
```

### Compiled Memory Sections

# Memory Segments [S13a]

MEMORY

```
{  
    MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
    SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

SECTIONS

```
{  
    .intvecs: > 0x00000000  
    .text : > MAIN  
    .const : > MAIN  
    .cinit : > MAIN  
    .pinit : > MAIN  
    .data : > SRAM_DATA  
    .bss : > SRAM_DATA  
    .heap : > SRAM_DATA  
    .stack : > SRAM_DATA (HIGH)  
}
```

Data Memory

Start Address

(unused)

End Address

Code Memory

Start Address

(unused)

End Address

# Memory Segments [S13b]

MEMORY

```
{  
  MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
  SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

SECTIONS

```
{  
  .intvecs: > 0x00000000  
  .text : > MAIN  
  .const : > MAIN  
  .cinit : > MAIN  
  .pinit : > MAIN  
  .data : > SRAM_DATA  
  .bss : > SRAM_DATA  
  .heap : > SRAM_DATA  
  .stack : > SRAM_DATA (HIGH)  
}
```

Data Memory  
(SRAM\_DATA)

Start Address  
(0x20000000)

(unused)

End Address  
(0x20010000)

Code Memory (MAIN)

Start Address  
(0x00000000)

(unused)

End Address  
(0x00040000)

# Memory Segments [S13c]

MEMORY

```
{  
  MAIN      (RX) : origin = 0x00000000, length = 0x00040000  
  SRAM_DATA (RW) : origin = 0x20000000, length = 0x00010000  
}
```

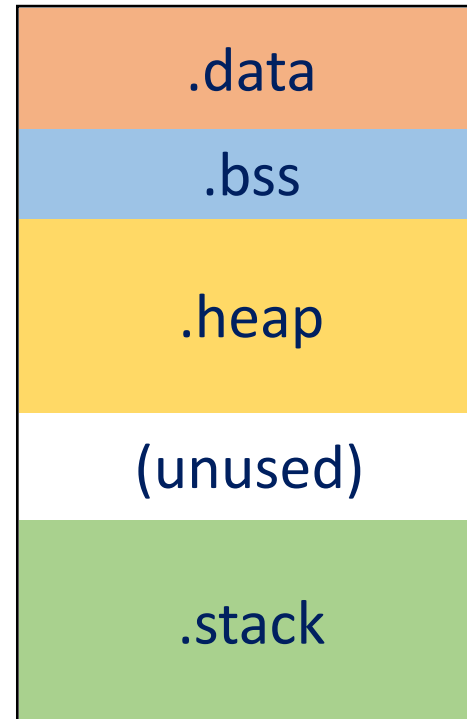
SECTIONS

```
{  
  .intvecs: > 0x00000000  
  .text : > MAIN  
  .const : > MAIN  
  .cinit : > MAIN  
  .pinit : > MAIN  
  .data : > SRAM_DATA  
  .bss : > SRAM_DATA  
  .heap : > SRAM_DATA  
  .stack : > SRAM_DATA (HIGH)  
}
```

**Data Memory  
(SRAM\_DATA)**

Start Address  
(0x20000000)

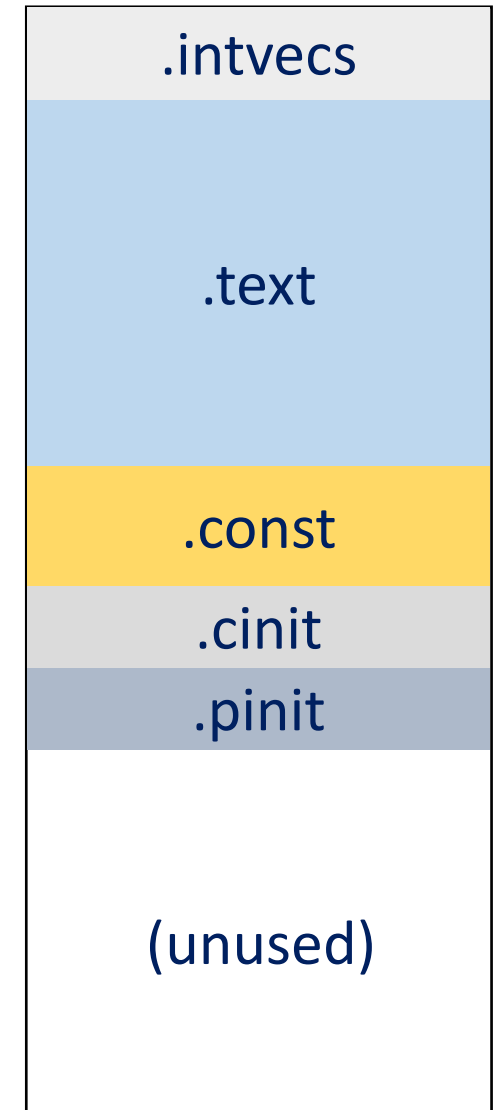
End Address  
(0x20010000)



**Code Memory (MAIN)**

Start Address  
(0x00000000)

End Address  
(0x00040000)



# Linker Flags [S14a]

Option & Format	Purpose
<code>-map [NAME]</code>	Outputs a memory map file [NAME] from the result of linking
<code>-T [NAME]</code>	Specifies a linker script name [NAME]
<code>-o [NAME]</code>	Place the output in the filename [NAME]
<code>-O&lt;#&gt;</code>	The level of optimizations from [#=0-3] (-O0, -O1, -O2, -O3)
<code>-Os</code>	Optimize for memory size
<code>-z stacksize=[SIZE]</code>	The amount of stack space to reserve
<code>-shared</code>	Produce a shared library (dynamic linking library)
<code>-l [LIB]</code>	Link with library
<code>-L [DIR]</code>	Include the following library path
<code>-Wl, &lt;OPTION&gt;</code>	Pass option to linker from compiler
<code>-Xlinker &lt;OPTION&gt;</code>	Pass option to linker from compiler

# Passing Flags to Linker from Compiler [S14b]

- You can pass arguments from the compiler to the linker

```
$ gcc <other-options-here> -Xlinker -Map=main.map
```

```
$ gcc <other-options-here> -Xlinker -T=mk125z_lnk.ld
```

```
$ gcc <other-options-here> -Wl,option
```

```
$ gcc <other-options-here> -Wl,-Map,main.map
```

```
$ gcc <other-options-here> -Wl,-Map=main.map
```

# Executable File Formats [S15]

- Executable and Linkable Format (ELF)
- Common Object File Format (COFF)
- Intel Hex Record
- Motorola S Record (SREC)
- ARM Image Format (AIF)

```
:10010000214601360121470136007EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Intel Hex Record Example File<sup>[3]</sup>

```
00000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 |.ELF.....|
00000010 02 00 3e 00 01 00 00 00 c5 48 40 00 00 00 00 |..>.....H@.....|
```

ELF File Example<sup>[4]</sup>