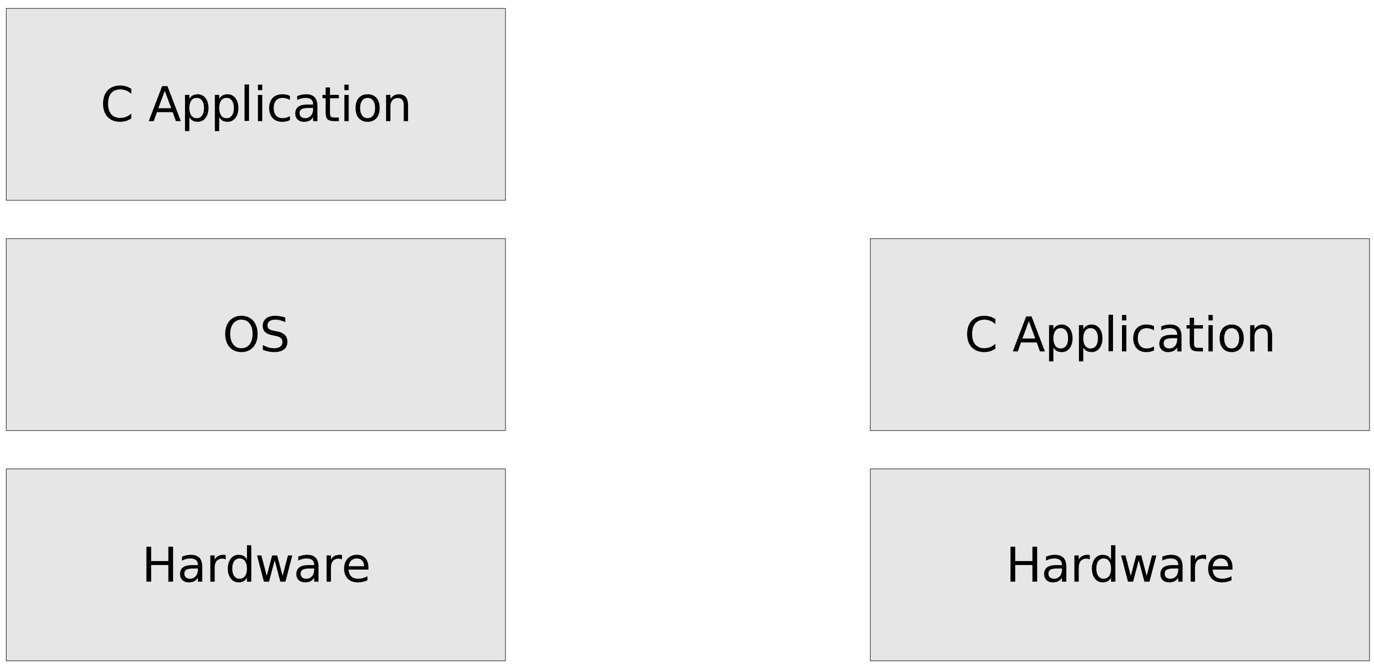
Embedded Programming with the GNU Toolchain

Vijay Kumar B.
vijaykumar@zilogic.com

What?



Conventional C Programs

Our Case

Why?

- Embedded Firmware Development
- RTOS development – eCOS, RTEMS, ...
- Bootloader development – U-Boot, ...
- Programming DSPs
- Testing Microprocessors cores implemented in ASICs / FPGAs

How?

- 3 Example Scenarios
- Hello Embedded World – Add 2 numbers in registers in assembly
- Add 2 numbers from memory in assembly
- Add 2 numbers from memory in C

Scenario I - Overview

- Cortex-M3 Processor
- Writing Assembly Programs
- Emulating Cortex-M3 with Qemu

ARMv7

- Latest revision of ARM architecture – ARMv7
- Cortex Processor – ARMv7 implementation
- Profiles
  - A Profile – GPOS and applications
  - R Profile – optimized for realtime systems
  - M Profile – optimized for low cost embedded systems

Cortex-M3 Features

- Thumb-2 Instruction Set
- Bit Banding
- Integrated Peripherals
  - NVIC
  - Memory Protection Unit (MPU)
  - Debug Peripherals

CM3 SoCs

- SoC vendors license CM3 from ARM
- SoC vendors use it as building block
- Licensees
  - TI – Stellaris processors
  - Atmel – ATSAM3U series
  - STMicroelectronics – STM32
  - NXP - LPC1700

LM3S811

- Cortex-M3 core
- Memory
  - 64KB Flash
  - 8KB RAM
- Peripherals
  - 10 bit ADCs
  - I2C, SPI, UARTs, PWM
  - 32 GPIOs

Registers

- Load Store Architecture
- Data processing instructions – register operands
- Large register file – 16 32-bit registers

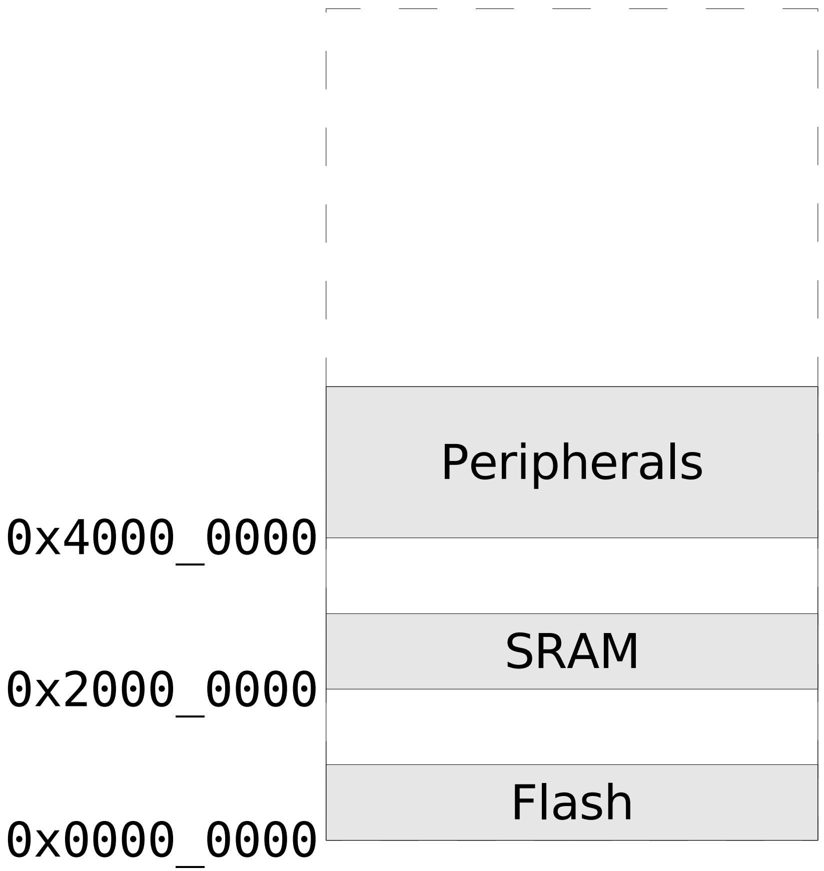
Registers (Contd.)

R0	R8
R1	R9
R2	R10
R3	R11
R4	R12
R5	R13 (SP)
R6	R14 (LR)
R7	R15 (PC)

PSR	PRIMASK
	FAULTMASK
CONTROL	BASEPRI

- R0 – R12
  - General Purpose
- R13
  - Stack Pointer
- R14
  - Link Register
- R15
  - Program Counter

Memory Map



- CM3 has a fixed memory map
- Easy to port software
- 4GB Address Space
- LM3S811
  - 64KB Flash
  - 8KB SRAM

Reset

- SP from address 0x0
- PC from address 0x4
- Address is mapped to Flash

0x0004	Reset Vec.
0x0000	Initial SP

Assembly

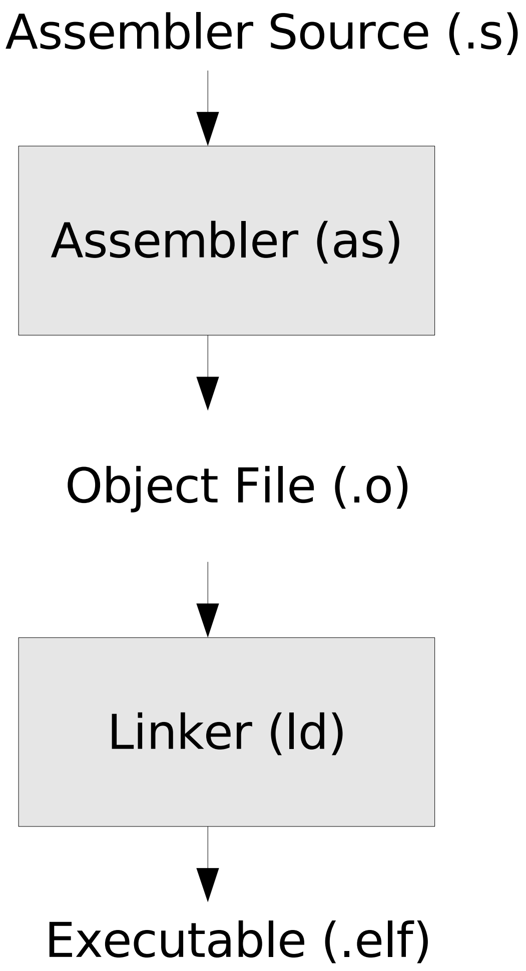
```
label:  instruction          @comment
```

- label: convenient way to refer to the memory location
- instruction: ARM instruction or assembler directive
- comment: starts with @

Hello Embedded World

```
            .thumb
            .syntax unified
sp:         .word 0x100
reset:      .word start+1

start:
            mov r0, #5
            mov r1, #4
            add r2, r1, r0

stop:       b   stop
```

Toolchain


Assembler Source (.s)
Assembler (as)
Object File (.o)
Linker (ld)
Executable (.elf)

Toolchain (Contd.)

```
$ arm-none-eabi-as -mcpu=cortex-m3 -o add.o add.s
```

- Cross toolchain prefix - `arm-none-eabi-`
- `-mcpu=cortex-m3` Specifies the CPU
- `-o` Specifies the output file

Toolchain (Contd.)

```
$ arm-none-eabi-ld -Ttext=0x0 -o add.elf add.o
```

- Cross toolchain prefix - `arm-none-eabi-`
- `-Ttext=0x0` Addresses should be assigned to instructions starting from 0.
- `-o` Specifies the output file

Toolchain (Contd.)

```
$ arm-none-eabi-nm add.elf
...
00000004 t reset
00000000 t sp
00000008 t start
00000014 t stop
```

- List symbols from object file
- Verify initial SP and reset vector are located at required address

Toolchain (Contd.)

- ELF file format contains meta information for OS
- Binary format contains consecutive bytes starting from an address
- Convenient for flashing tools

Toolchain (Contd.)

```
$ arm-none-eabi-objcopy -O binary add.elf add.bin
```

- objcopy – converts between different executable file formats
- -O specifies that output file format

Qemu

- Open source machine emulator - the processor and the peripherals
- Architectures – i386, ARM, MIPS, SPARC ...
- Used by various open source projects
  - OLPC
  - OpenMoko
  - Linux Kernel Testing

Emulating in Qemu

```
$ qemu-system-arm -M lm3s811evb -kernel add.bin
```

- `-M lm3s811evb` specifies the machine to be emulated
- `-kernel` specifies data to be loaded in Flash from address 0x0
- monitor interface – control and status
- can be used to view the registers

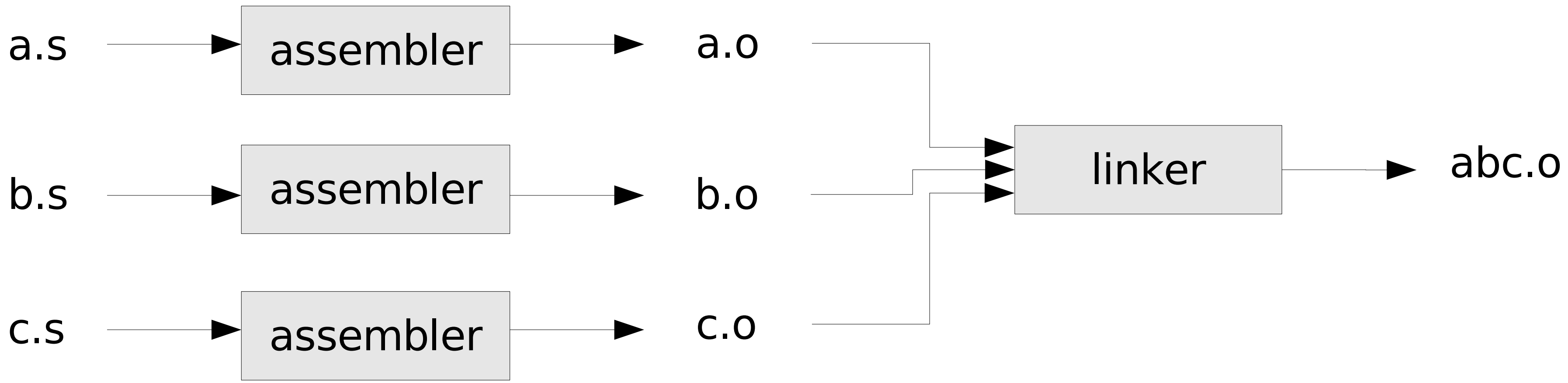
Review

- Writing simple assembly programs
- Building and linking them using GNU Toolchain
- Emulating Cortex-M3 processor using Qemu

Scenario II - Overview

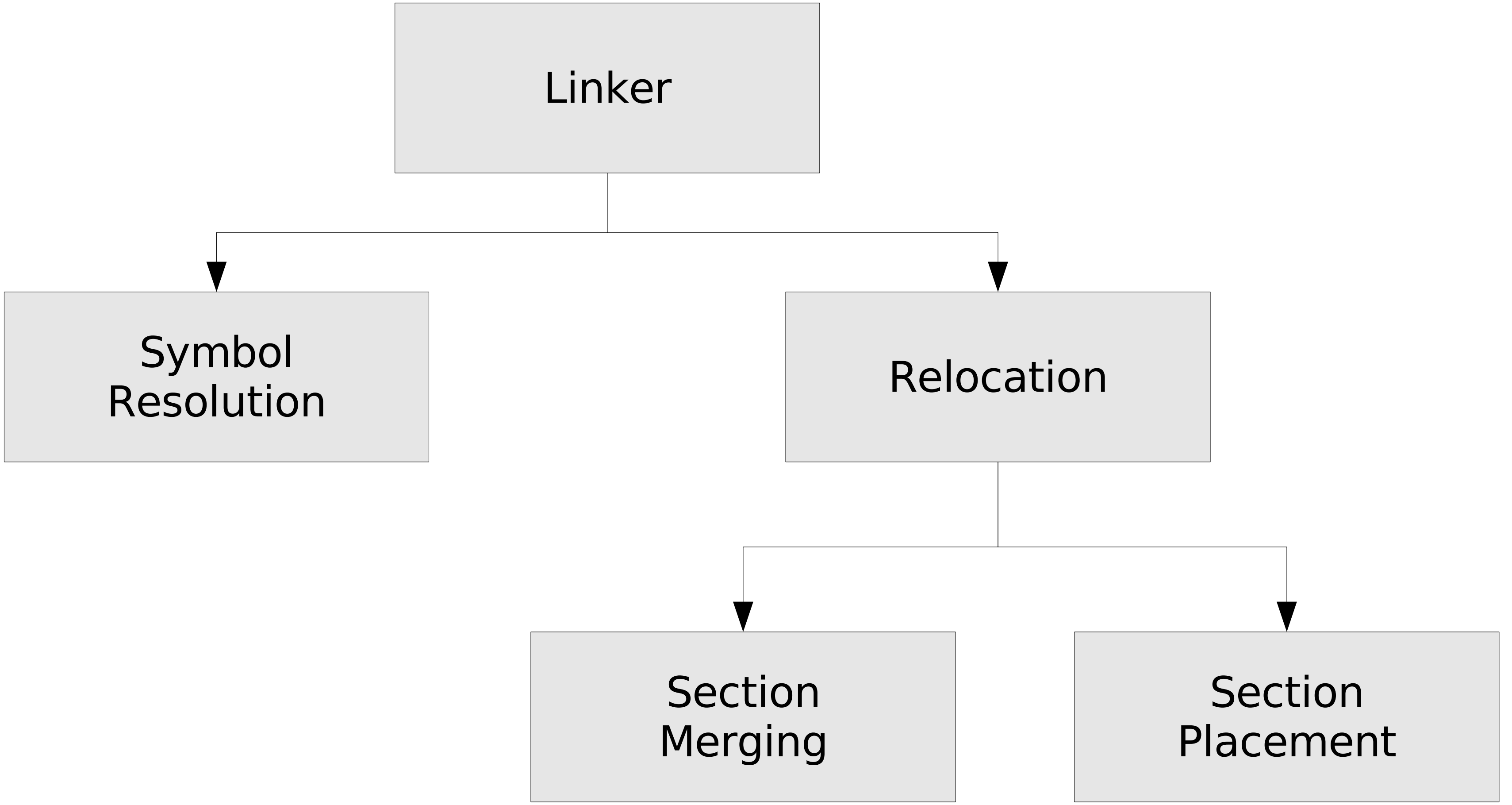
- Role of Linker
- Linker Scripts
- Placing data in RAM

Linker

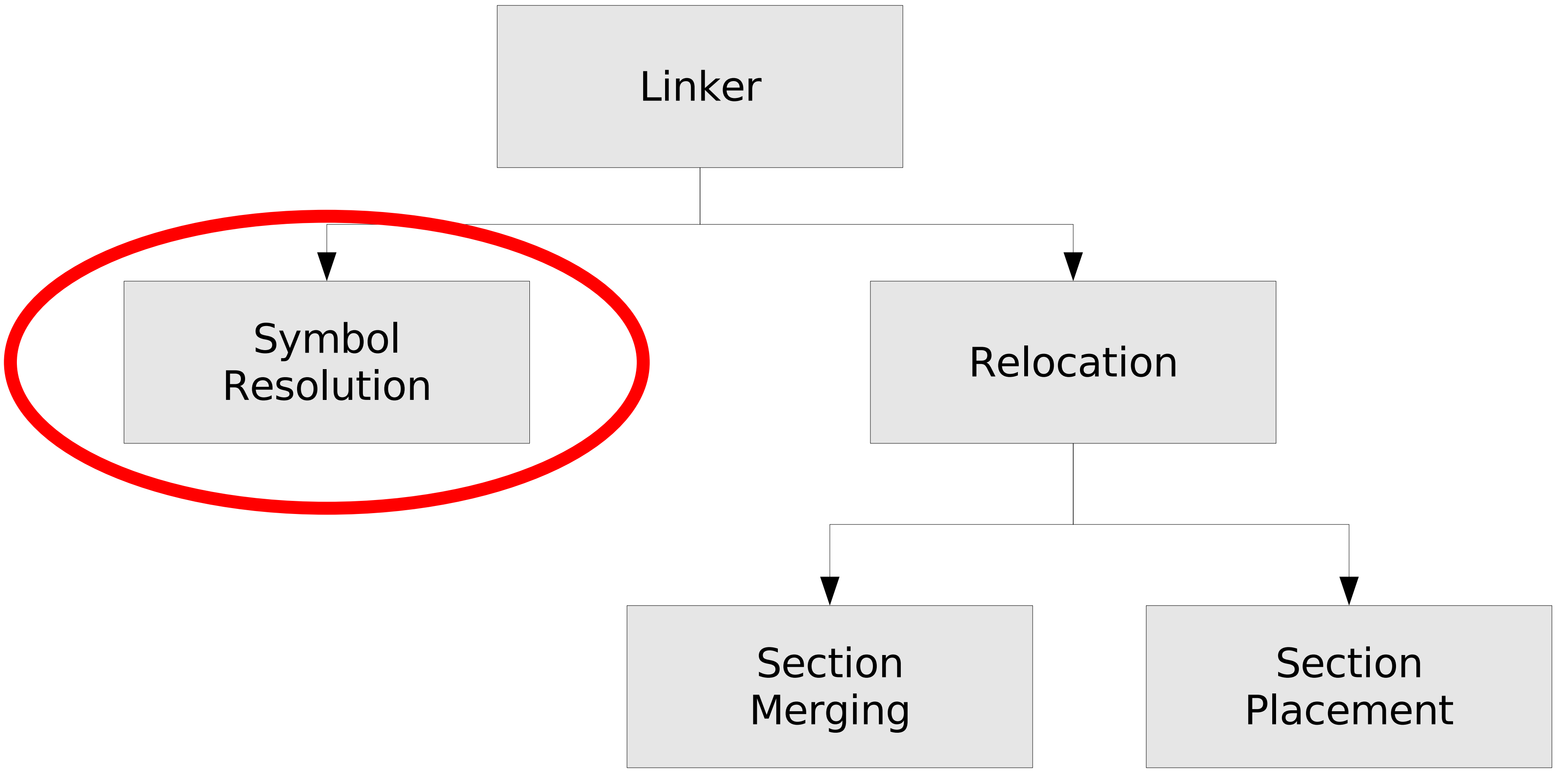


- In multi-file programs – combines multiple object files to form executable

Linker (Contd.)



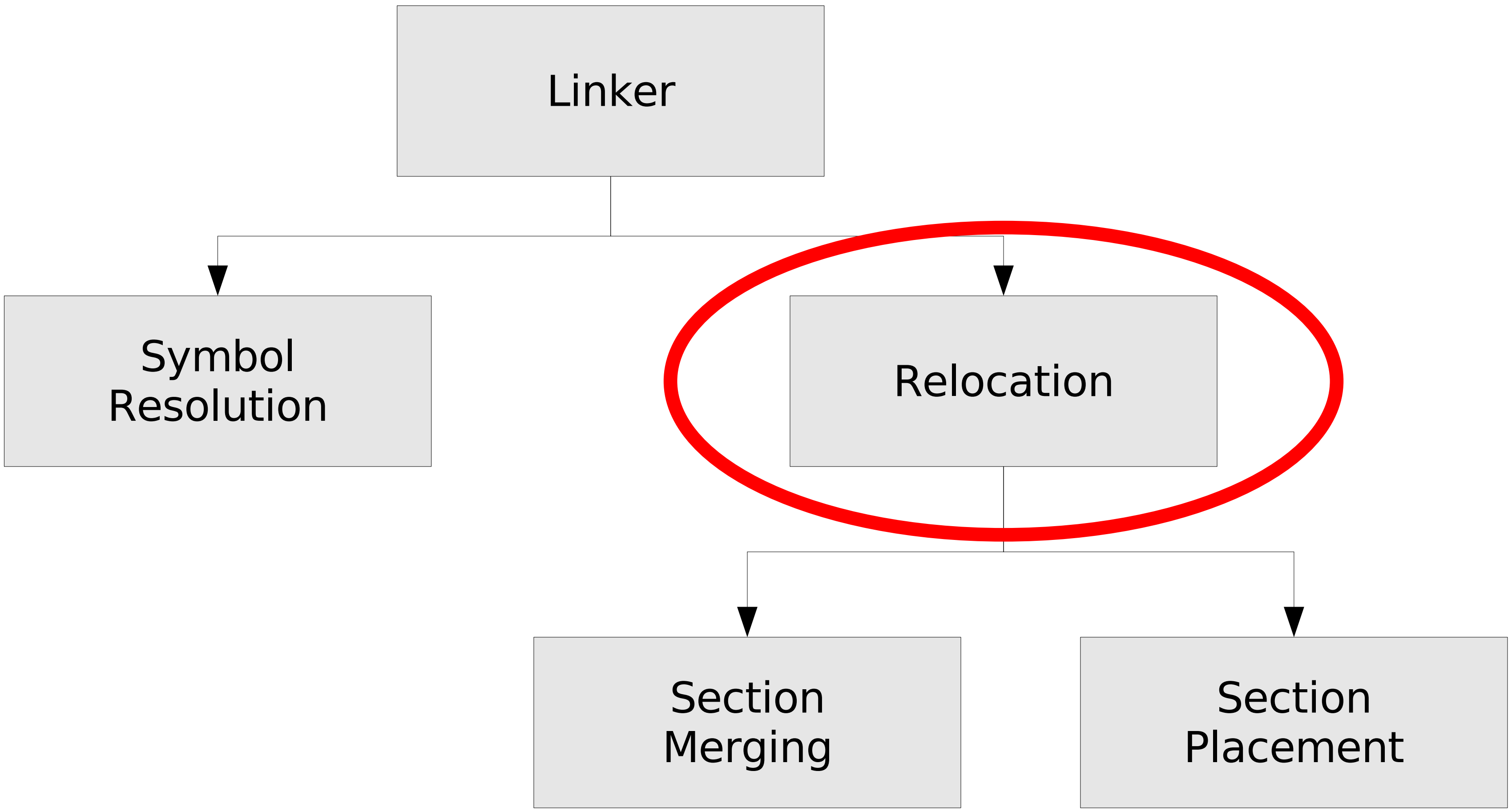
Linker (Contd.)



Symbol Resolution

- Functions are defined in one file
- Referenced in another file
- References are marked unresolved by the compiler
- Linker patches the references

Linker


Linker
Symbol Resolution
Relocation
Section Merging
Section Placement

Relocation

- Code generated assuming it starts from address X
- Code should start from address Y
- Change addresses assigned to labels
- Patch label references

Sections

- Placing related bytes at a particular location.
- Example:
  - instructions in Flash
  - data in RAM
- Related bytes are grouped together using sections
- Placement of sections can be specified

Sections (Contd.)

- Most programs have atleast two sections, .text and .data
- Data or instructions can be placed in a section using directives
- Directives
  - .text
  - .data
  - .section

Sections (Contd.)

```
        .data
arr:    .word 10, 20, 30, 40, 50
len:    .word 5
        .text
start:  mov r1, #10
        mov r2, #20
        .data
result: .skip 4
        .text
        add r3, r2, r1
        sub r3, r2, r1
```

.data section

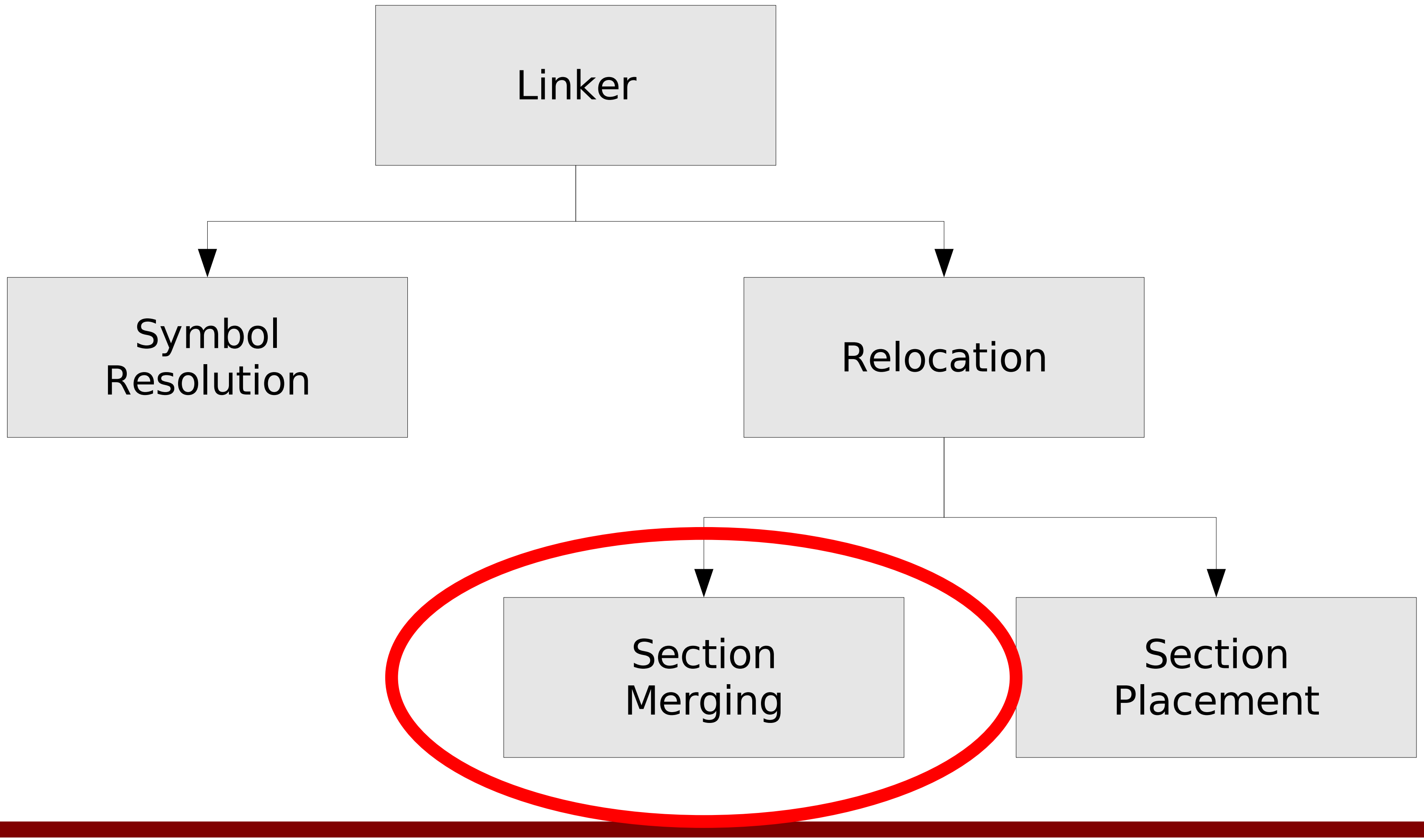
```
0000_0000 arr:    .word 10, 20, 30, 40, 50
0000_0014 len:    .word 5
0000_0018 result: .skip 4
```

.text section

```
0000_0000 start:  mov r1, #10
0000_0004         mov r2, #20
0000_0008         add r3, r2, r1
0000_000C         sub r3, r2, r1
```

- Source – sections can be interleaved
- Bytes of a section – contiguous addresses

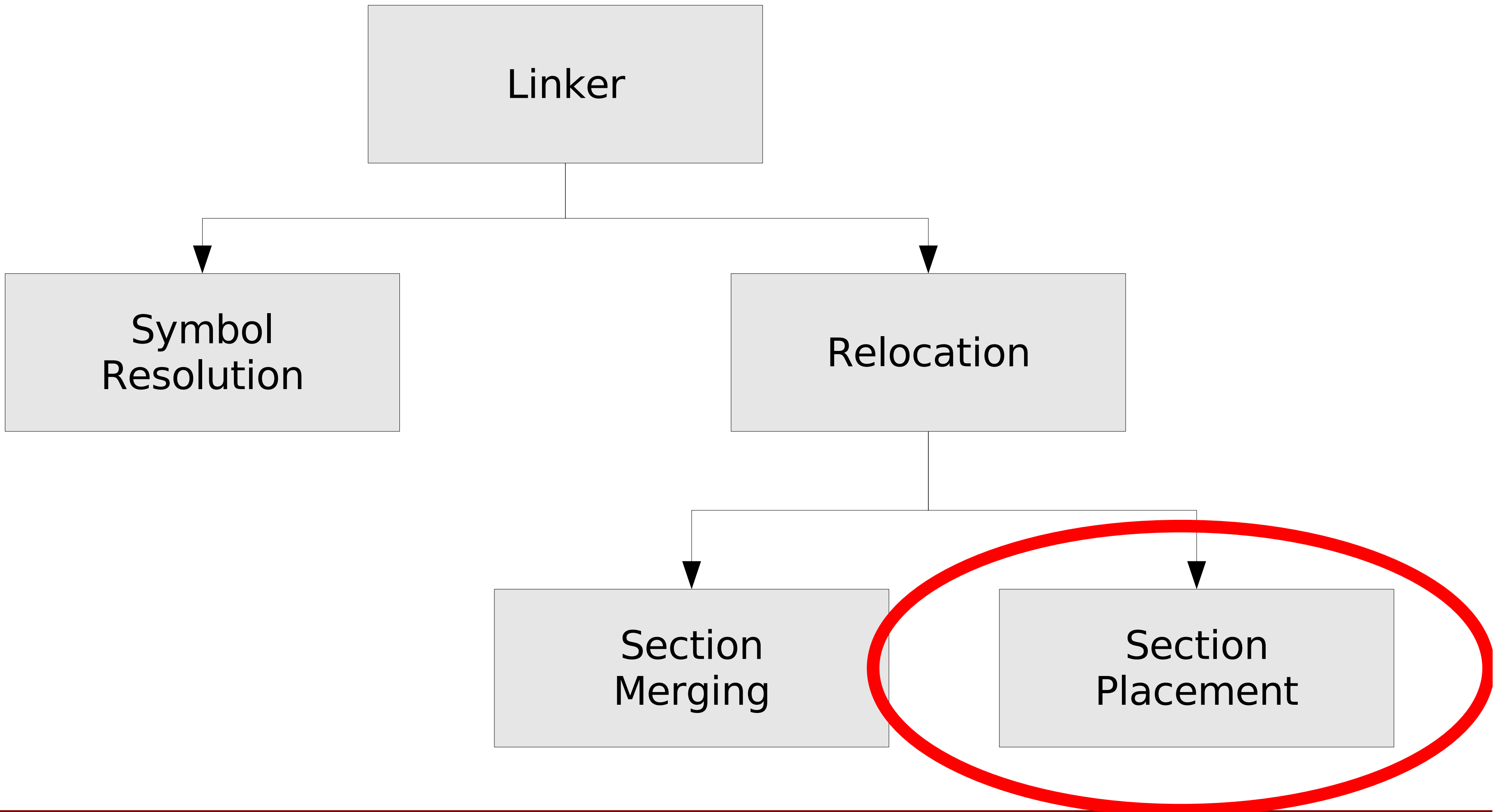
Linker


Linker
Symbol Resolution
Relocation
Section Merging
Section Placement

Section Merging

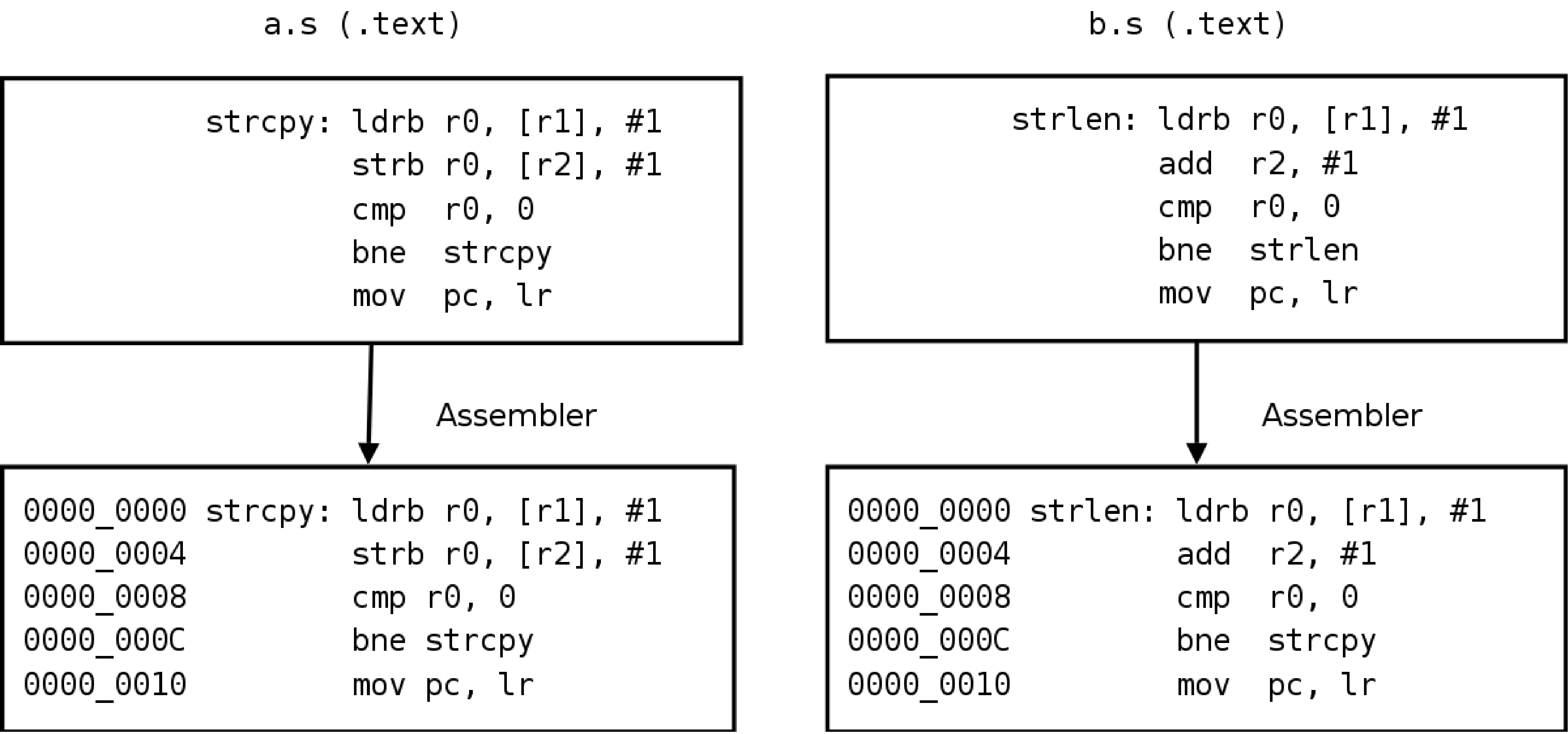
- Linker merges sections in the input files into sections in the output file
- Default merging – sections of same name
- Symbols get new addresses, and references are patched
- Section merging can be controlled by linker script files

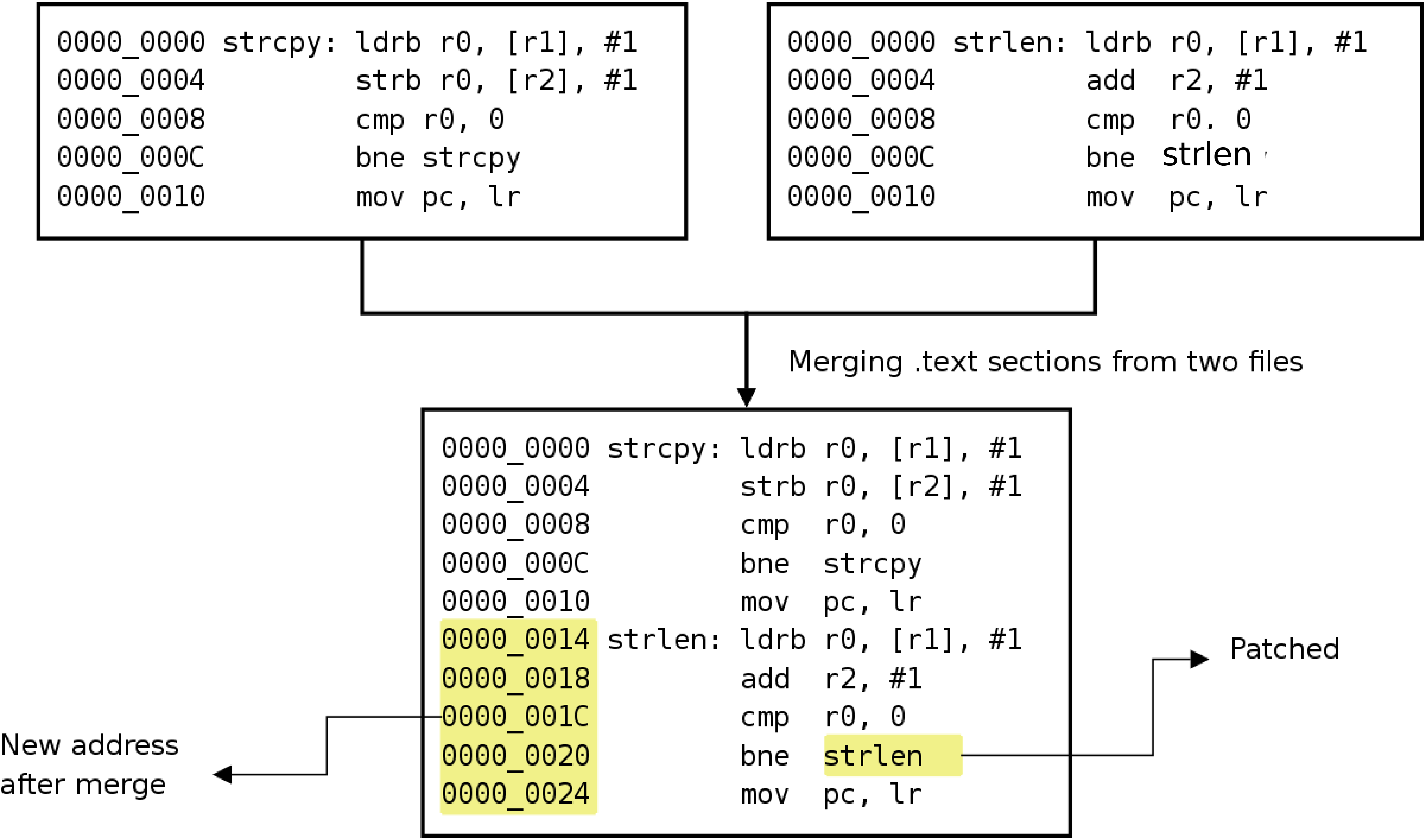
Linker


Linker
Symbol Resolution
Relocation
Section Merging
Section Placement

Section Placement

- Bytes in each section is given addresses starting from 0x0
- Labels get addresses relative to the start of section
- Linker places section at a particular address
- Labels get new address, label references are patched


a.s (.text)
strcpy: ldrb r0, [r1], #1
strb r0, [r2], #1
cmp r0, 0
bne strcpy
mov pc, lr
Assembler
0000_0000 strcpy: ldrb r0, [r1], #1
0000_0004 strb r0, [r2], #1
0000_0008 cmp r0, 0
0000_000C bne strcpy
0000_0010 mov pc, lr
b.s (.text)
strlen: ldrb r0, [r1], #1
add r2, #1
cmp r0, 0
bne strlen
mov pc, lr
Assembler
0000_0000 strlen: ldrb r0, [r1], #1
0000_0004 add r2, #1
0000_0008 cmp r0, 0
0000_000C bne strcpy
0000_0010 mov pc, lr



0000_0000 strcpy: ldrb r0, [r1], #1
0000_0004 strb r0, [r2], #1
0000_0008 cmp r0, 0
0000_000C bne strcpy
0000_0010 mov pc, lr
0000_0000 strlen: ldrb r0, [r1], #1
0000_0004 add r2, #1
0000_0008 cmp r0. 0
0000_000C bne strlen
0000_0010 mov pc, lr
Merging .text sections from two files
0000_0000 strcpy: ldrb r0, [r1], #1
0000_0004 strb r0, [r2], #1
0000_0008 cmp r0, 0
0000_000C bne strcpy
0000_0010 mov pc, lr
0000_0014 strlen: ldrb r0, [r1], #1
0000_0018 add r2, #1
0000_001C cmp r0, 0
0000_0020 bne strlen
0000_0024 mov pc, lr
New address after merge
Patched

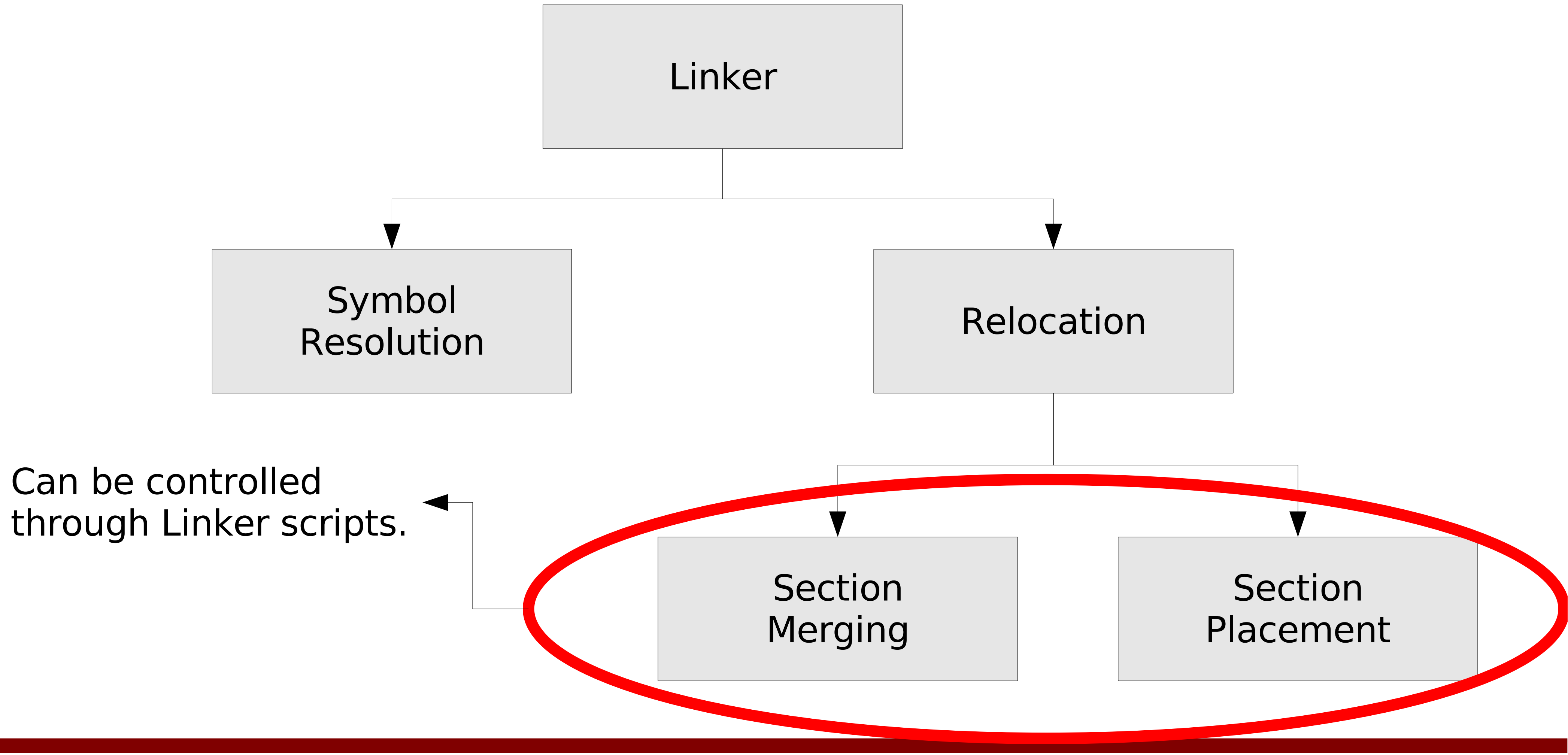
```
0000_0000 strcpy: ldrb r0, [r1], #1
0000_0004         strb r0, [r2], #1
0000_0008         cmp  r0, 0
0000_000C         bne  strcpy
0000_0010         mov  pc, lr
0000_0014 strlen: ldrb r0, [r1], #1
0000_0018         add  r2, #1
0000_001C         cmp  r0, 0
0000_0020         bne  strlen
0000_0024         mov  pc, lr
```

Placing .text section at 0x2000_0000

```
2000_0000 strcpy: ldrb r0, [r1], #1
2000_0004         strb r0, [r2], #1
2000_0008         cmp  r0, 0
2000_000C         bne  strcpy
2000_0010         mov  pc, lr
2000_0014 strlen: ldrb r0, [r1], #1
2000_0018         add  r2, #1
2000_001C         cmp  r0, 0
2000_0020         bne  strlen
2000_0024         mov  pc, lr
```

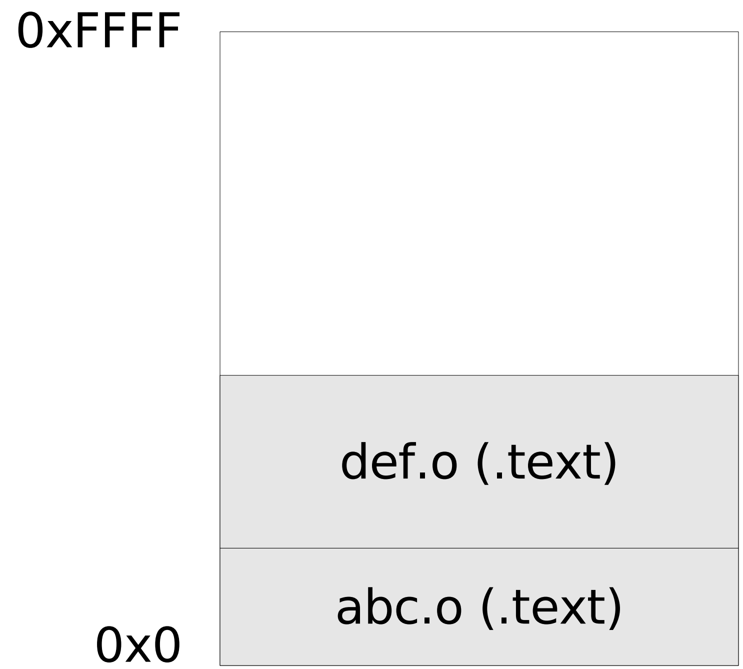
Patched

Linker Script



Simple Linker Script

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        abc.o (.text);
        def.o (.text);
    } > FLASH
}
```



Simple Linker Script

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        abc.o (.text);
        def.o (.text);
    } > FLASH
}
```

Section Merging

Simple Linker Script

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        abc.o (.text);
        def.o (.text);
    } > FLASH
}
```

Section Placement

Making it Generic

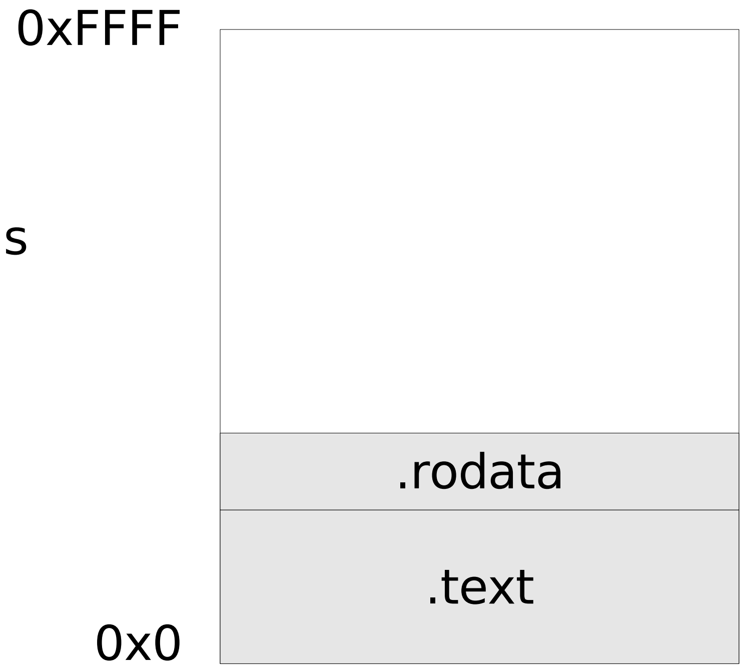
```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        * (.text);
    } > FLASH
}
```

Wildcards to represent .text form all input files

Multiple Sections

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        * (.text);
    } > FLASH

    .rodata : {
        * (.rodata);
    } > FLASH
}
```

Dealing with
mutiple sections



Data in RAM

- Add two numbers from memory
- Assembly source
- Linker Script

RAM is Volatile!

- RAM is volatile
- Data cannot be made available in RAM at power-up
- All code and data should be in Flash at power-up
- Startup code – copies data from Flash to RAM

RAM is Volatile! (Contd.)

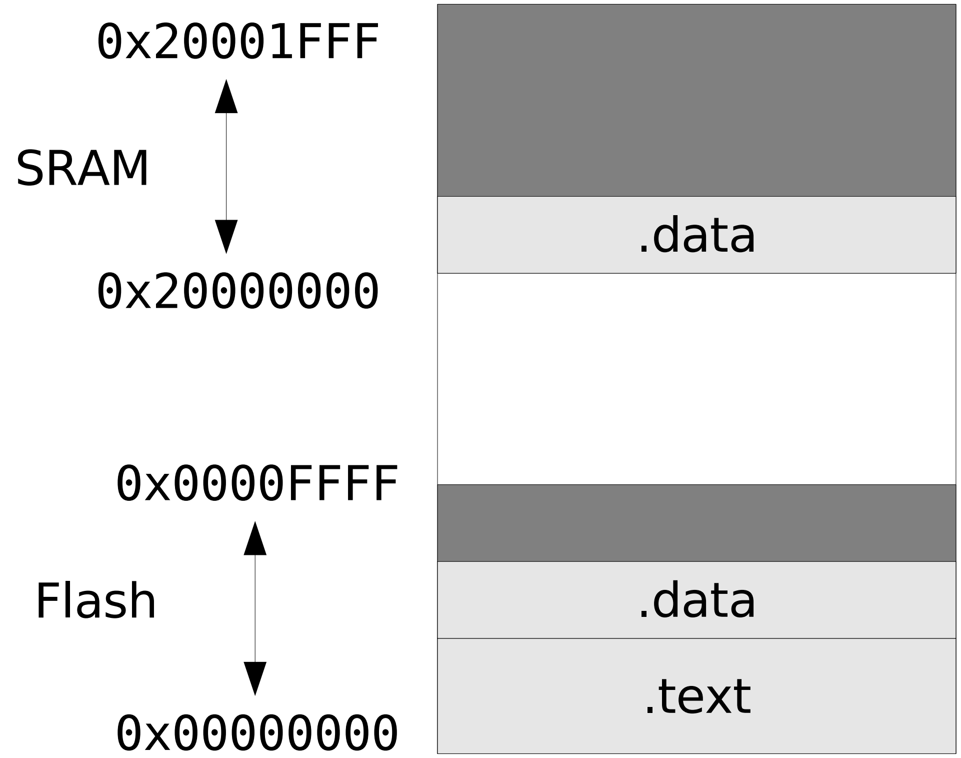
- .data section should be present in Flash at power-up
- Section has two addresses
  - load address (aka LMA)
  - run-time address (aka VMA)
- So far only run-time address – actual address assigned to labels
- Load address defaults to run-time address

Linker Script Revisited

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        * (.text);
    } > FLASH

    .data : {
        * (.data);
    } > SRAM
}
```

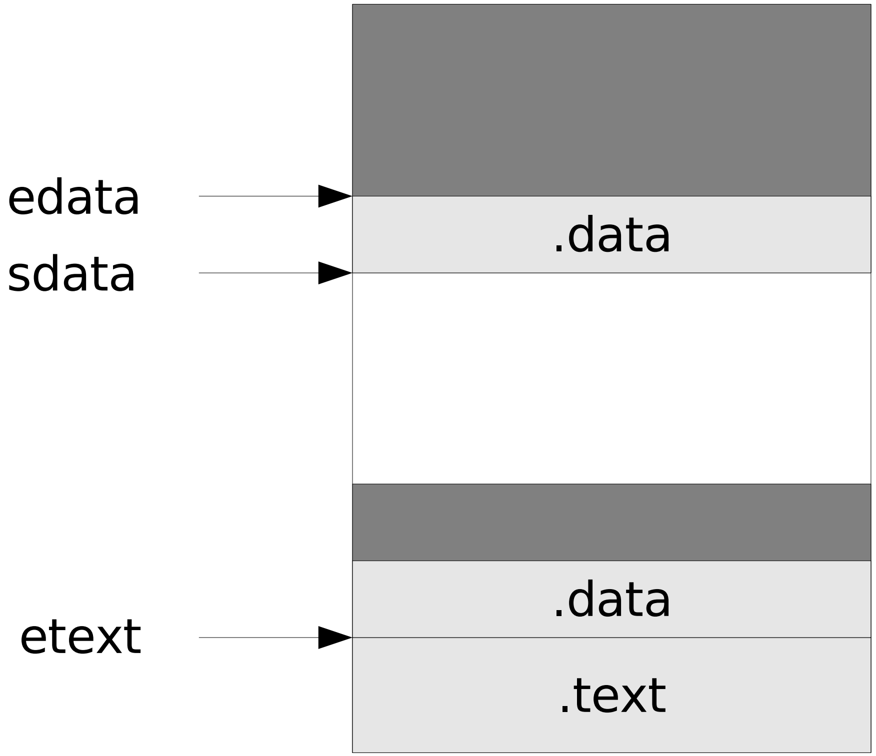
Linker Script Revisited

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        * (.text);
    } > FLASH

    .data : {
        * (.data);
    } > SRAM AT> FLASH
}
```



Linker Script Revisited

```
MEMORY {
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x10000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x2000
}

SECTIONS {
    .text : {
        * (.text);
        etext = .;
    } > FLASH

    .data : {
        sdata = .;
        * (.data);
        edata = .;
    } > SRAM AT> FLASH
}
```



Data in RAM

- Copy .data from Flash to RAM

```
start:
        ldr r0, =sdata              @ Load the address of sdata
        ldr r1, =edata              @ Load the address of edata
        ldr r2, =etext              @ Load the address of etext

copy:   ldrb r3, [r2]               @ Load the value from Flash
        strb r3, [r0]               @ Store the value in RAM

        add  r2, r2, #1             @ Increment Flash pointer
        add  r0, r0, #1             @ Increment RAM pointer

        cmp  r0, r1                 @ Check if end of data
        bne  copy                   @ Branch if not end of data
```

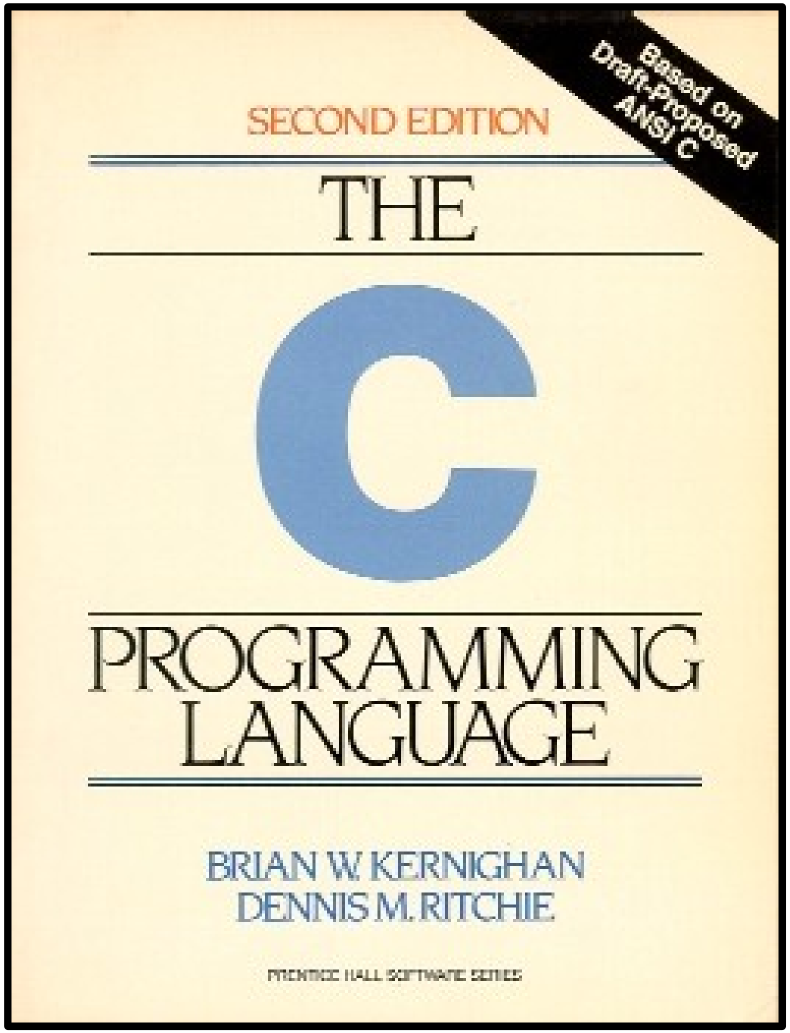
Review

- Linker Script can control
  - Section Merging
  - Section Placement
- .data placed in RAM, .text in Flash
- RAM is volatile
- at load time .data is in Flash
- at startup .data is copied from Flash to RAM

Scenario III - Overview

- C Environment Requirements
- C Sections
- C Source Code
- Linker Script

Doing it in C



- Environment has to be setup
  - Stack pointer
  - Non-initalized global variables, initialized to zero
  - Initialized global variables must have their initial value

C Sections

- Sections created by GCC
  - .text – for functions
  - .data – for initialized global data
  - .bss – for uninitialized global data
  - .rodata – for strings and global variables defined as const

Credits

- Cash Register – Nikola Smolenski
- Cerebral Cortex - www.toosmarttostart.samhsa.gov
- Reset Button – flattop341
  http://www.flickr.com/photos/flattop341/224175619/
- Church Relocation – Fletcher6
  http://commons.wikimedia.org/wiki/File:Salem_Church_Relocation.JP
- Rope Image - Markus BÃ¤rlocher
  http://commons.wikimedia.org/wiki/File:Schotstek_links.jpg

Further Reading

- Embedded Programming using the GNU Toolchain - http://www.bravegnu.org/gnu-eprog/
- GNU Linker Manual
- GNU Assembler Manual