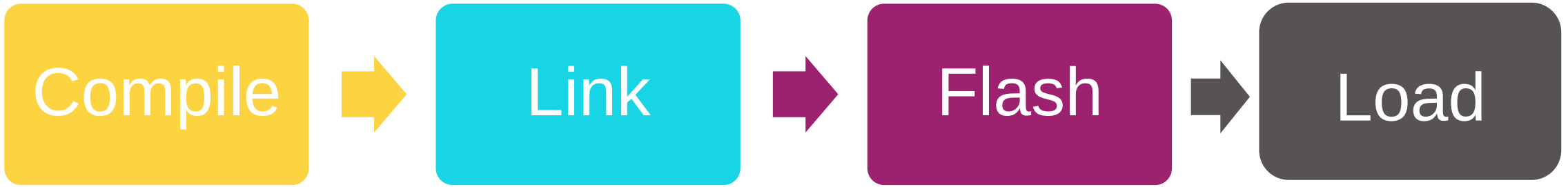
Journey from source to binary

Prakash Balasubramanian

06/April/2024

Source to binary - Flow



C/C++ to OBJ
ASM to OBJ

OBJ to ELF

ELF programmed to Microcontroller memories

ELF sections loaded from flash to selective memories

Site of action

Host PC with cross-compiler for target CPU (e.g. ARM Cortex-M0)

Compilation, Linking and ELF generation

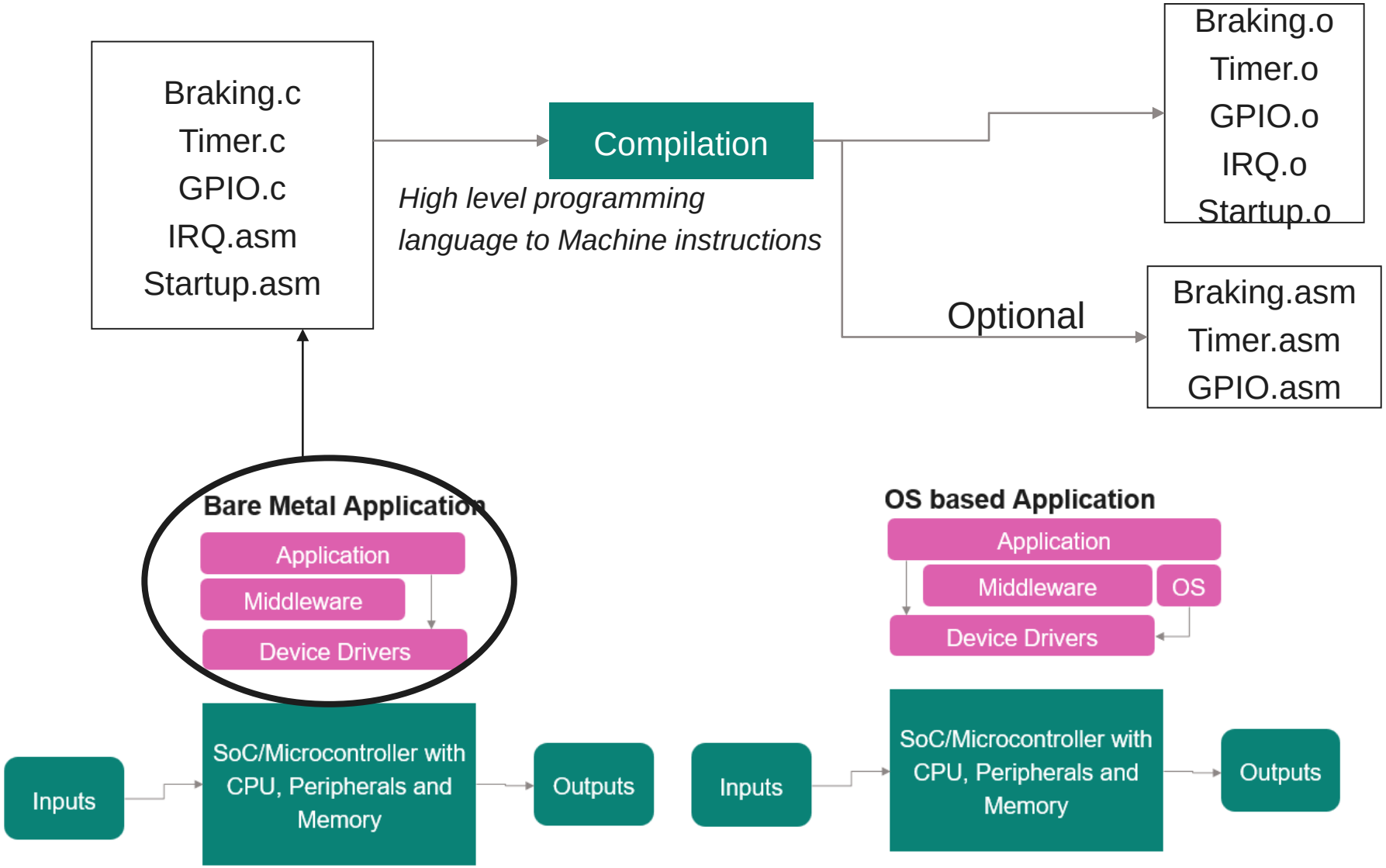
This debugger box helps in programming ELF sections of your application to Non-Volatile memories of the microcontroller

When you power-up the CPU

The startup software

1. Copies data sections of your application from flash to memory,
2. Clears BSS addresses
3. Optionally copies time critical program sections from flash to SRAM
4. Passes control to your main()

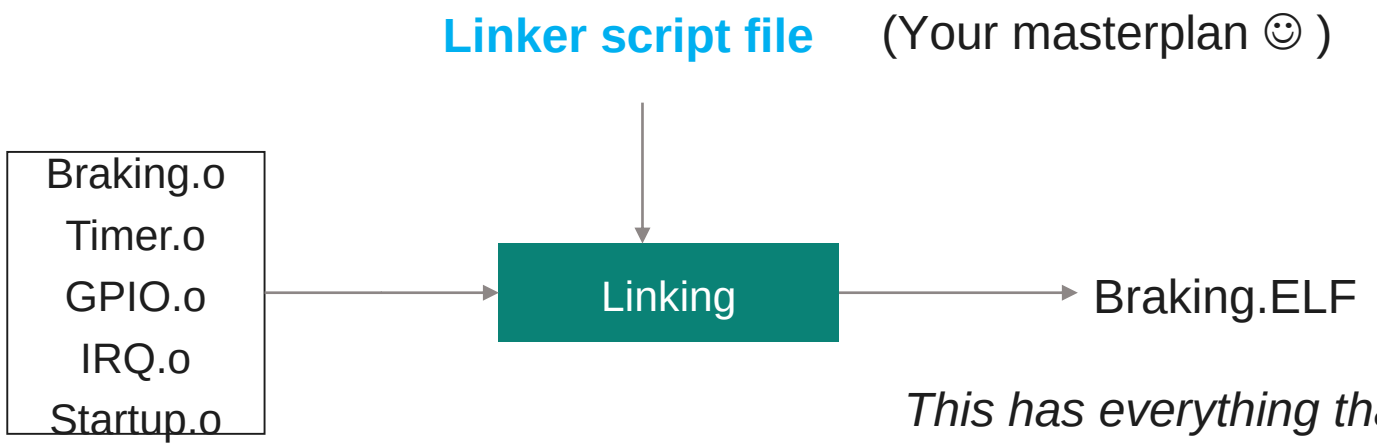
Source code compilation



The instructions and data in these object files are more or less ready for flashing.

BUT where must they be stored? What should be their address in memory?

Object linking



This has everything that your program loader needs to know!

Before Linking

func1()
func2()
func3()
Data1
Data2

Linking

After Linking

func1() 0x20000010 – 0x20001FE
func2() 0x20000200 – 0x200005CB
func3() 0x200005D0 – 0x200005F0
Data1 0x80000000
Data2[8] 0x80000004 – 0x80000023

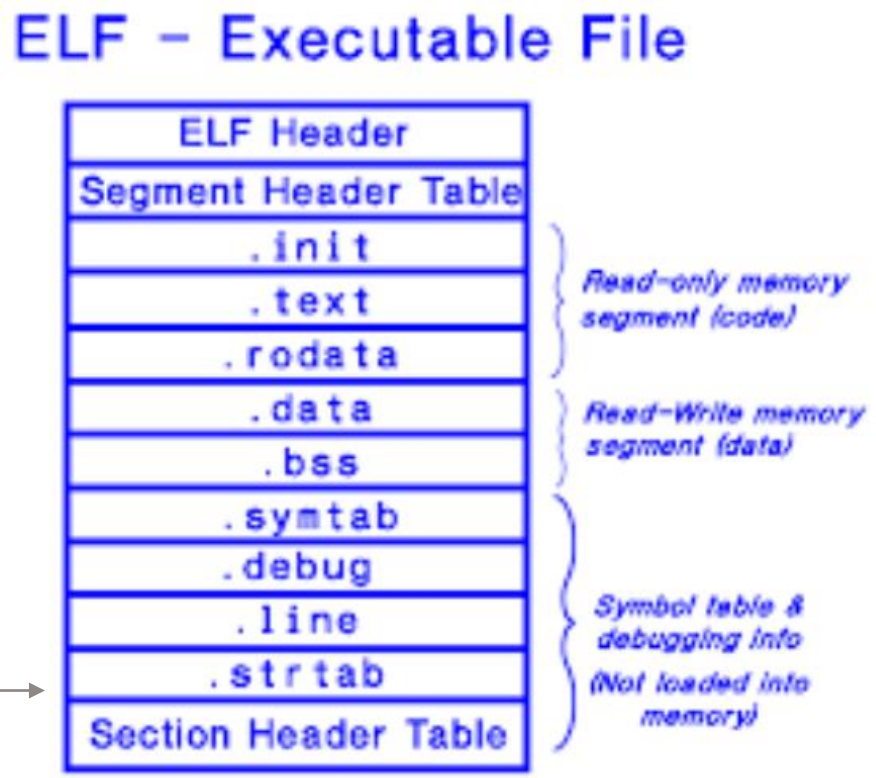
You specified the range in the linker script!

The linker has only followed your instructions faithfully.

What is an ELF anyways?

This bloke is the ELF!

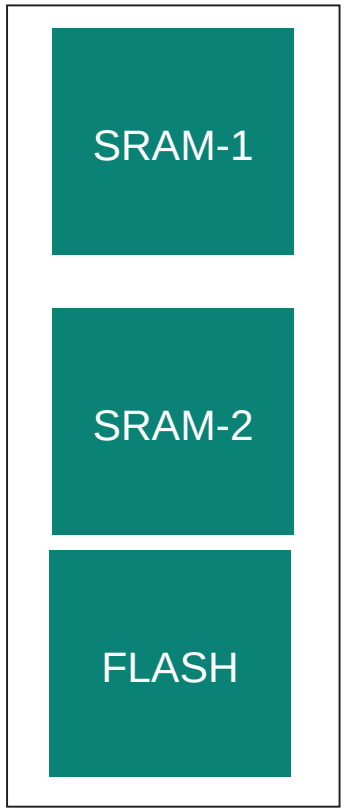
Executable and Linkable Format



This is also an ELF!

Next few slides deal with ELF creation!

Linker script – Big picture



Define MEMORY regions – i.e. Define start address and length of each system memory block

Assign your program to “Code”, “RODATA”, “DATA” and “BSS” INPUT SECTIONS respectively

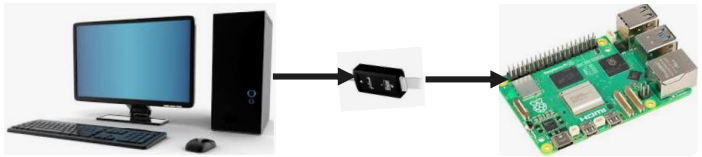
Group several INPUT Sections with a common requirement to an OUTPUT section

Assign OUTPUT Sections to MEMORY sections

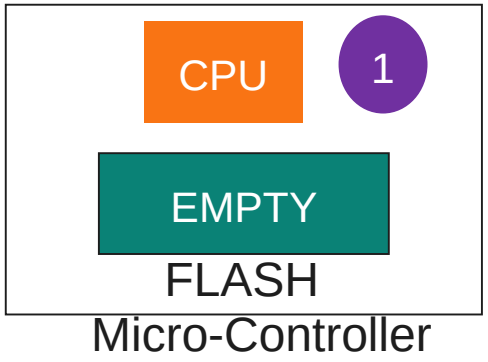
For each output section, define its:

- **Load region**
- **Run region**

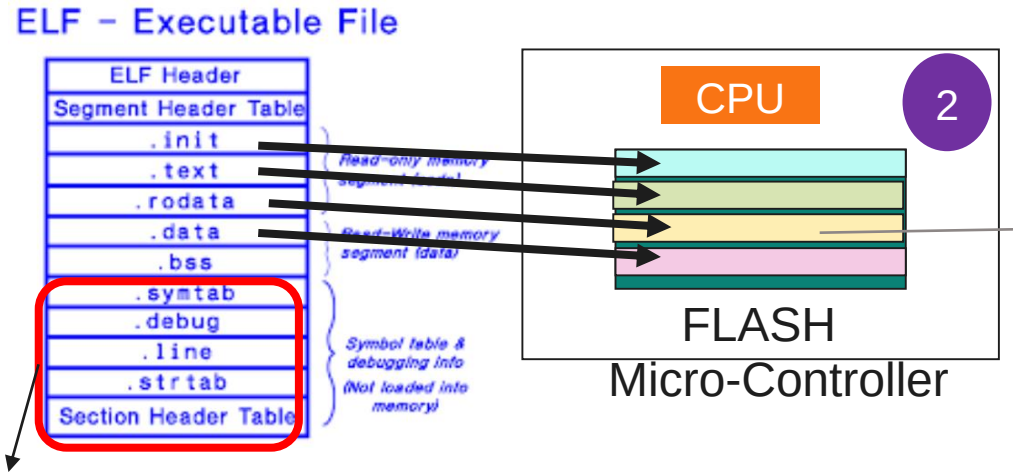
Putting them all together



Your brand new development board



After Flashing your application

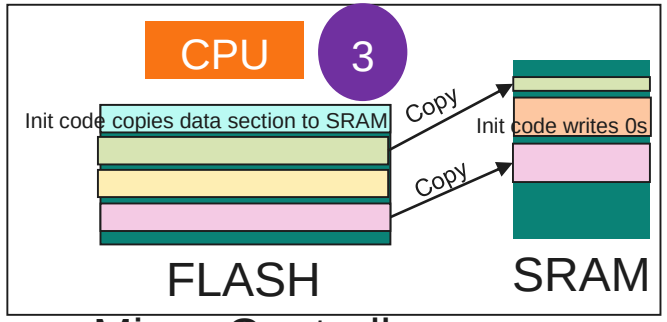


Your application now resides in load region

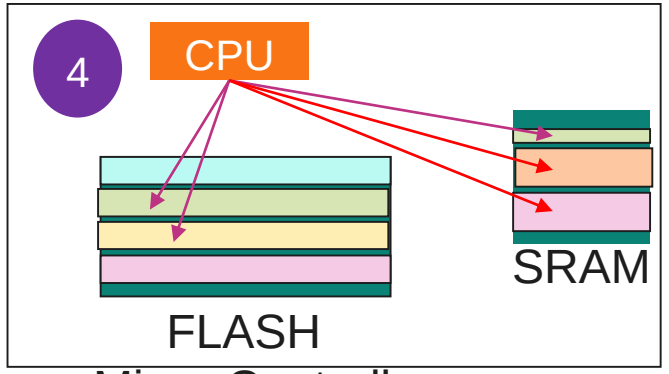
Not flashed.

These are needed only for debugging

At run time (After powering up)



At run time (After program loading)

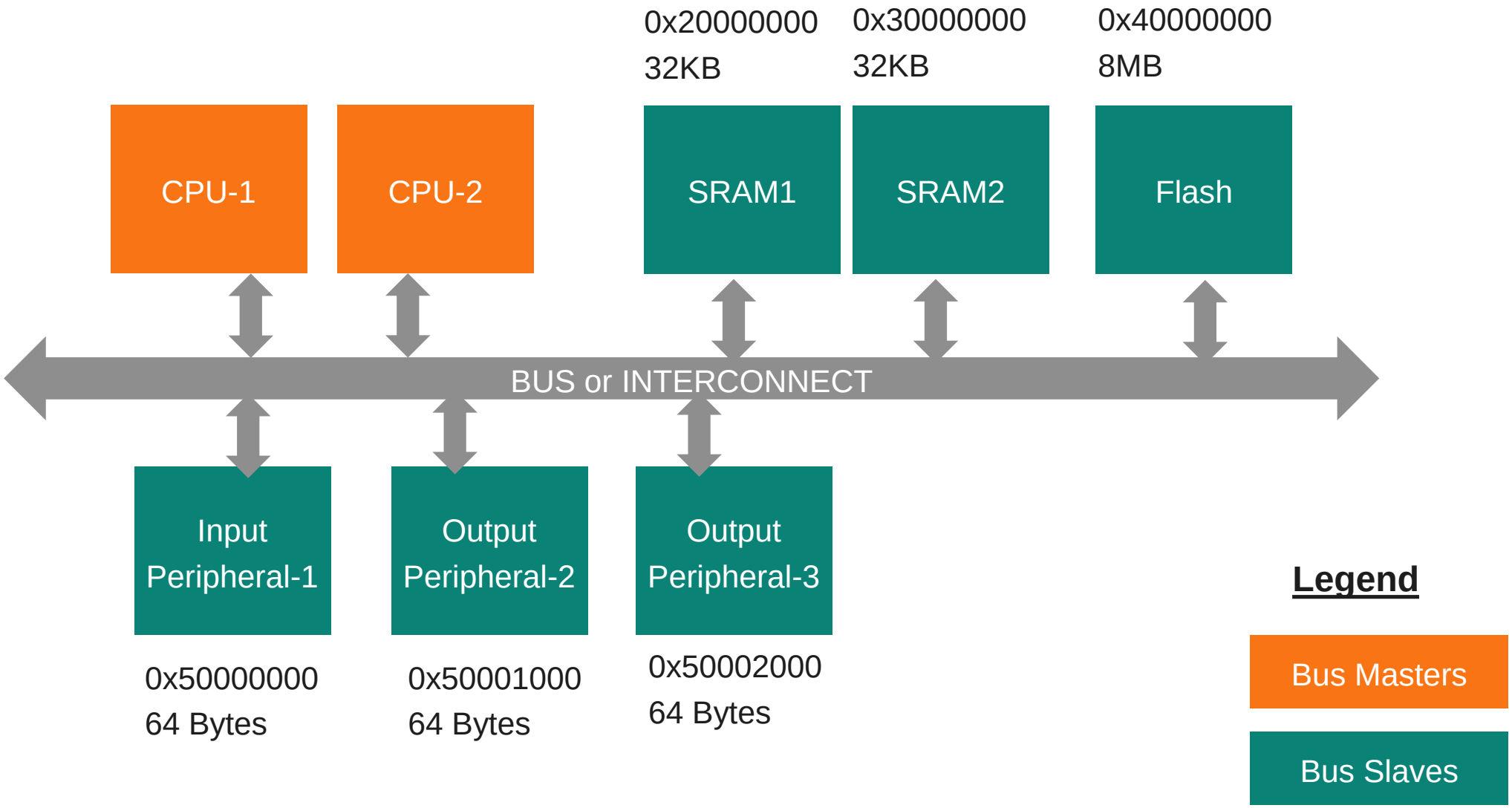


CPU fetches TEXT and RODATA from flash

CPU accesses DATA and BSS in SRAM

CPU may fetch instructions of time critical TEXT segment which has been coped to SRAM.

An exemplary Microcontroller



Example Linker Script (GNU syntax)

Define MEMORY regions

```
MEMORY
  {
     name [( attr )] : ORIGIN = origin , LENGTH = len
...
  }
```

```
MEMORY
{
  SRAM1(rwx): ORIGIN: 0x20000000, LENGTH = 32K

}
```

Example Linker Script (GNU syntax)

Define Sections

```
SECTIONS
{
  Output_Section1:
  {
     Object_File(Input section of the object file)
  } > REGION

 Output_Section2:
  {
    Object_File(Input section of the object file)
  } > REGION

}
```

```
SECTIONS
{
  .fasttext:
  {
     Adc.o(.text)
  } > SRAM1
  .data:
   {
      *(.data)
      *(.bss)
   } > SRAM2
 .text:
  {
    *(.text)
    *(.RODATA)
  } > FLASH
}
```

Assignment-1 (1 of 2)

- Here is a brief description of the microcontroller for which you are required to write a linker script. Please use GNU linker script format. You are **fully encouraged** to use internet for obtaining the linker script syntax and example linker scripts. You are also fully encouraged to discuss with your batchmates, but the work should be your own. Do NOT worry about the correctness of your answer.

- The exemplary microcontroller has 4MB of flash starting at 0x20000000, 128KB of SRAM-1 starting at 0x30000000 and 32KB of SRAM-2 starting at 0x40000000

- You may assume that the compiler generates the following input sections. Notice that there is a dot(.) prefixed to the names of the input sections.
  - **.text** for the code section
  - **.data** for the initialized data section
  - **.bss** for the uninitialized data section
  - **.rodata** for constant data section

- You may also assume that you found a way to instruct the compiler to create a special input section by name **.criticaltext**

Assignment-1 (2 of 2)

- Write a linker script fulfilling the following requirements
  - IMPORTANT: Learn and understand the concept of Wildcard notation in programming
  - Describe the memory regions of the microcontroller using MEMORY command
    - FLASH, SRAM1 and SRAM2
  - Using SECTIONS command
    - Create an output section by name **.flashsections** and assign the **.text and .rodata** input sections to it
    - Assign this **.flashsections** output section to FLASH memory region
    - Create an output section by name **.rwdata** and assign the **.data** and **.bss** input sections to it
    - Assign this **.rwdata** output section to SRAM1 memory region
    - Create an output section by name **.criticaltext** and assign the **.criticaltext** input section to it [Yes, the names of input and output sections are the same]
    - Assign this **.criticaltext** output section to SRAM2 memory region
  - For each of the input sections, find out how the start-address of the section and its length can be determined.
    - **HINT:** In the examples that you may find on the internet, you will find variables by name sbss, ebss, stext, etext etc.


Infineon