Embedded Systems Design

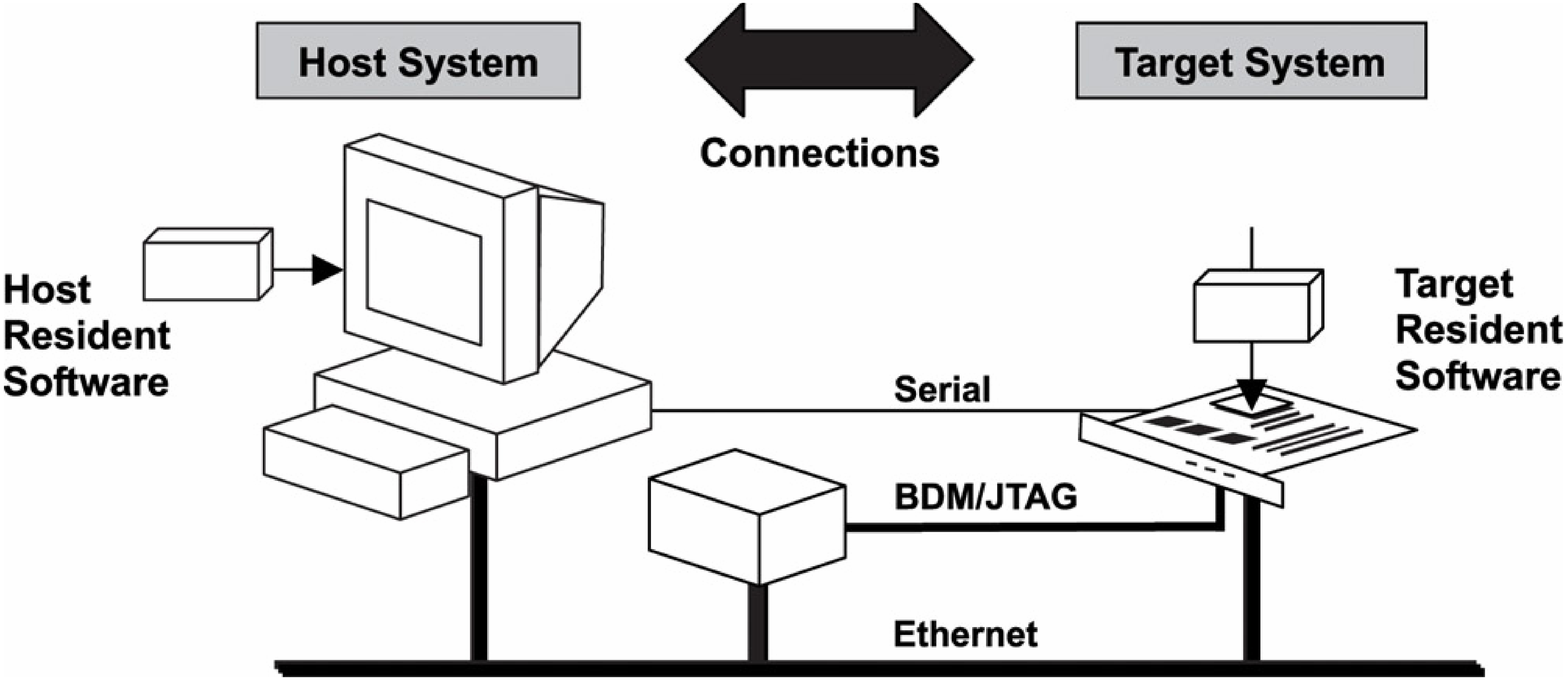
Lecture 11

Yongsoo Joo

KESL
Kookmin Univ. Embedded System Lab

Cross-Platform Development

- SW for an embedded systems is developed on one platform but runs on another



Cross-Platform Development

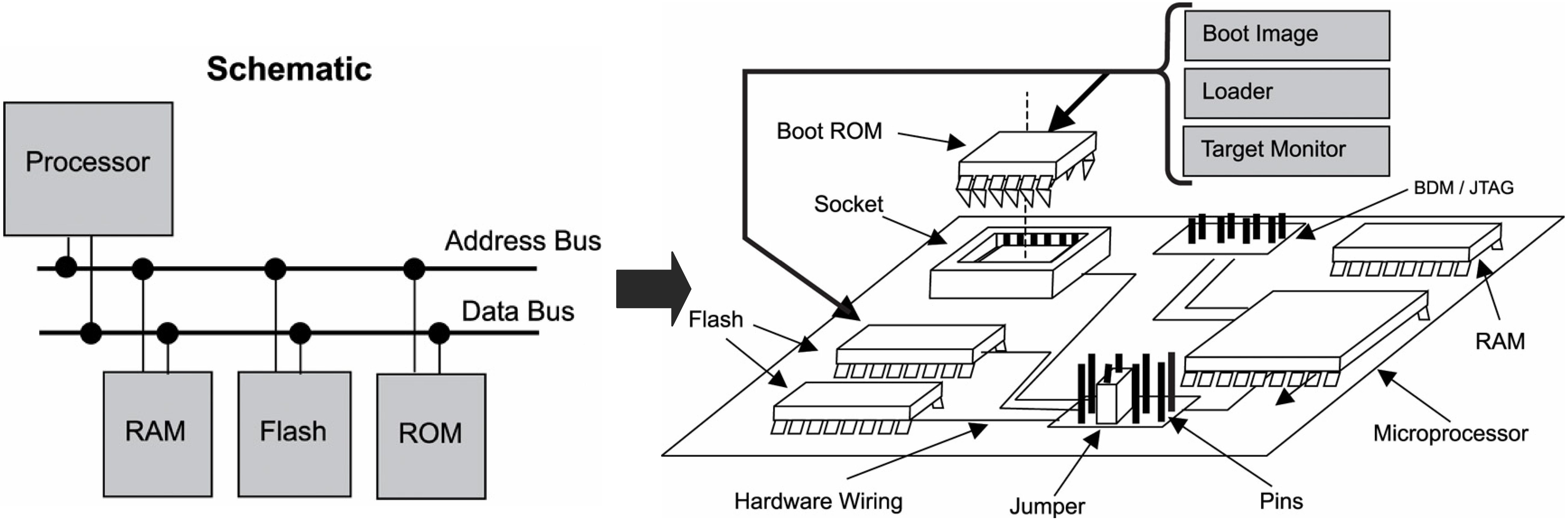
- Components
  - Host system
    - The system on which the embedded SW is developed
  - Target system
    - The embedded system under development
  - Cross toolchain
    - A set of tools to create executable code for the target system on the host system
  - Connection btw. host and target
    - JTAG/BDM/serial/ethernet

Host System

- Offer essential development tools (**toolchain**)
  - Editor
  - Cross compiler & assembler
  - Linker
  - Source-level debugger
  - Produces executable binary image that will run on target system
- Example host systems
  - PCs, workstations, laptops

Target System

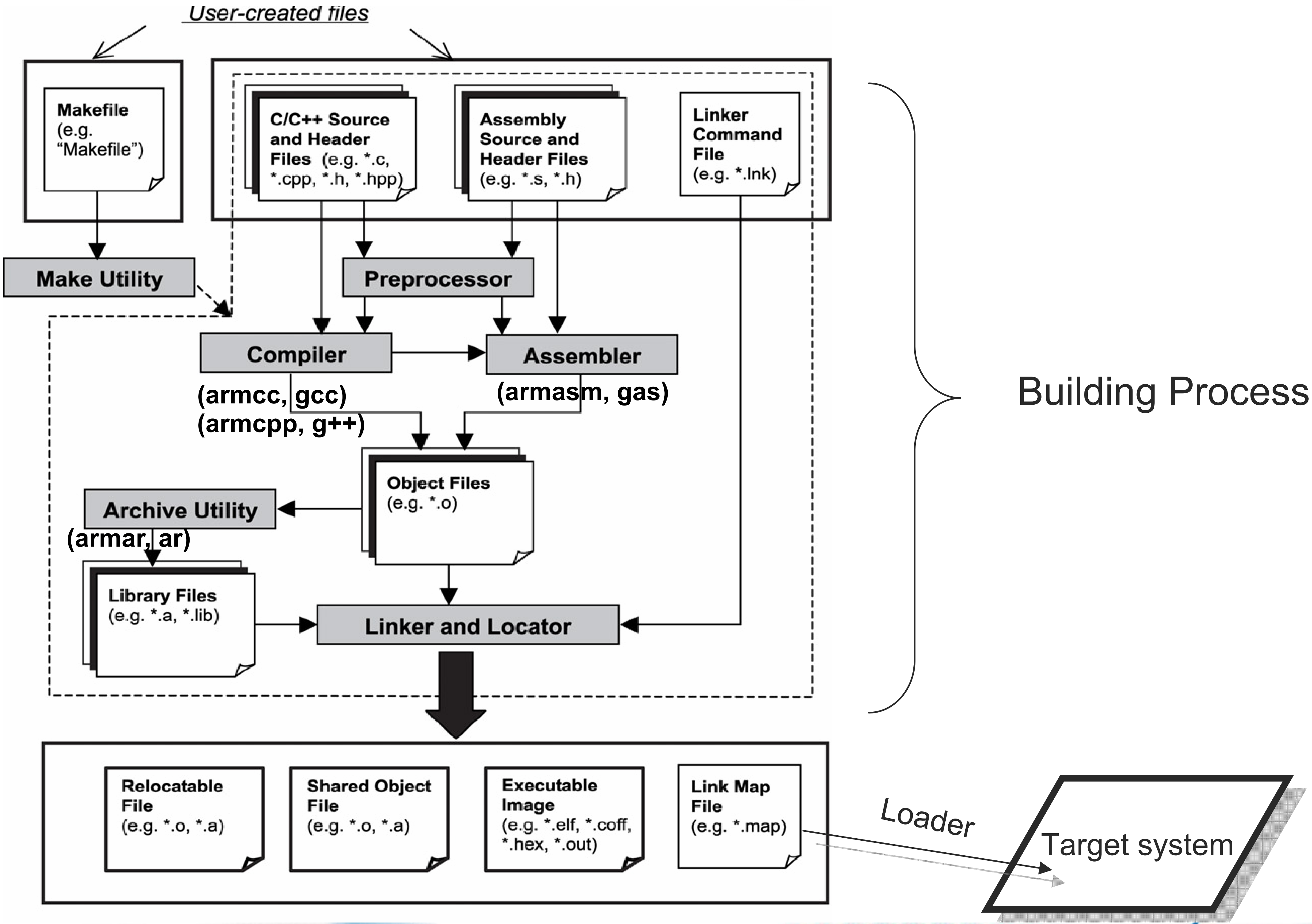
- Actual HW for the embedded system under development
- Need to understand the target system
  - How to store the program image on the target board
  - How to develop and debug the system iteratively



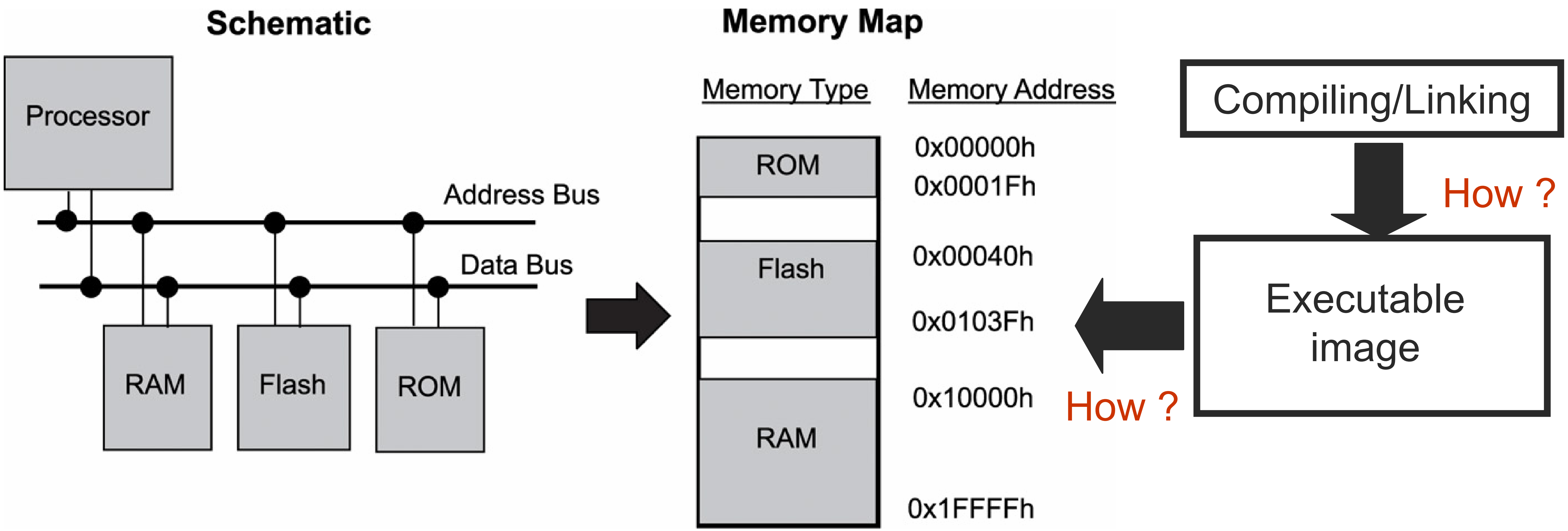
Target System vs. Final System

- Target system
  - Requires repeated downloading of the codes during the development phase
  - Reprogramming EEPROM or flash memory every time the code changes is time consuming
  - Developers prefer to transfer the code image directly into the RAM of the target system over one of serial/ethernet/JTAG/BDM interfaces
- Final system (= product)
  - Code image is permanently stored into the nonvolatile storage such as EEPROM or flash

Toolchain for Building Embedded SW


User-created files
Makefile (e.g. "Makefile")
C/C++ Source and Header Files (e.g. *.c, *.cpp, *.h, *.hpp)
Assembly Source and Header Files (e.g. *.s, *.h)
Linker Command File (e.g. *.lnk)
Make Utility
Preprocessor
Compiler
Assembler
(armcc, gcc)
(armcpp, g++)
(armasm, gas)
Object Files (e.g. *.o)
Archive Utility
(armar, ar)
Library Files (e.g. *.a, *.lib)
Linker and Locator
Relocatable File (e.g. *.o, *.a)
Shared Object File (e.g. *.o, *.a)
Executable Image (e.g. *.elf, *.coff, *.hex, *.out)
Link Map File (e.g. *.map)
Building Process
Loader
Target system

Compiling & Linking

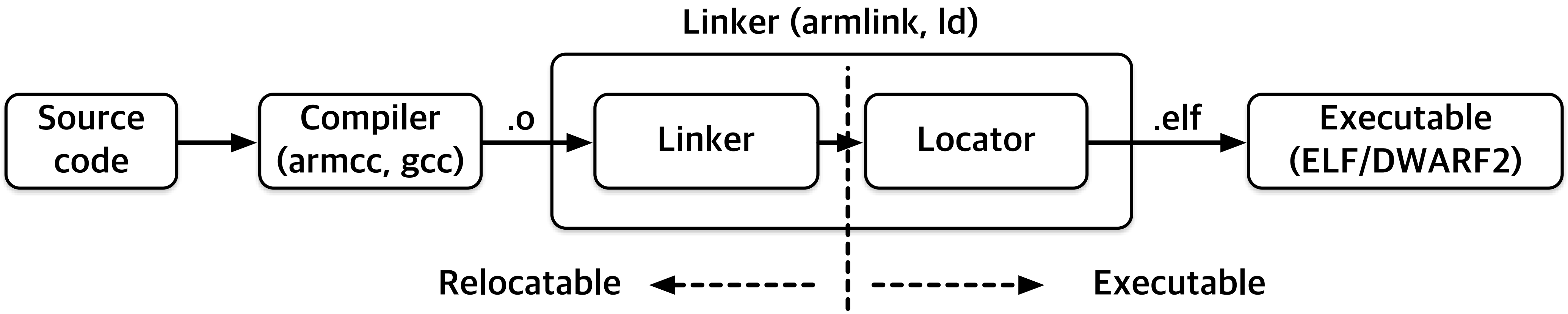


Compiling

- Output from compilation is an object file
- An object file contains
  - File size, binary code & data size, and source file name
  - Machine-specific binary instructions and data
  - Symbol table and symbol relocation table
  - Debug information
- Two common object file format
  - COFF: common object file format
  - ELF: executable linkable format

The ELF Format

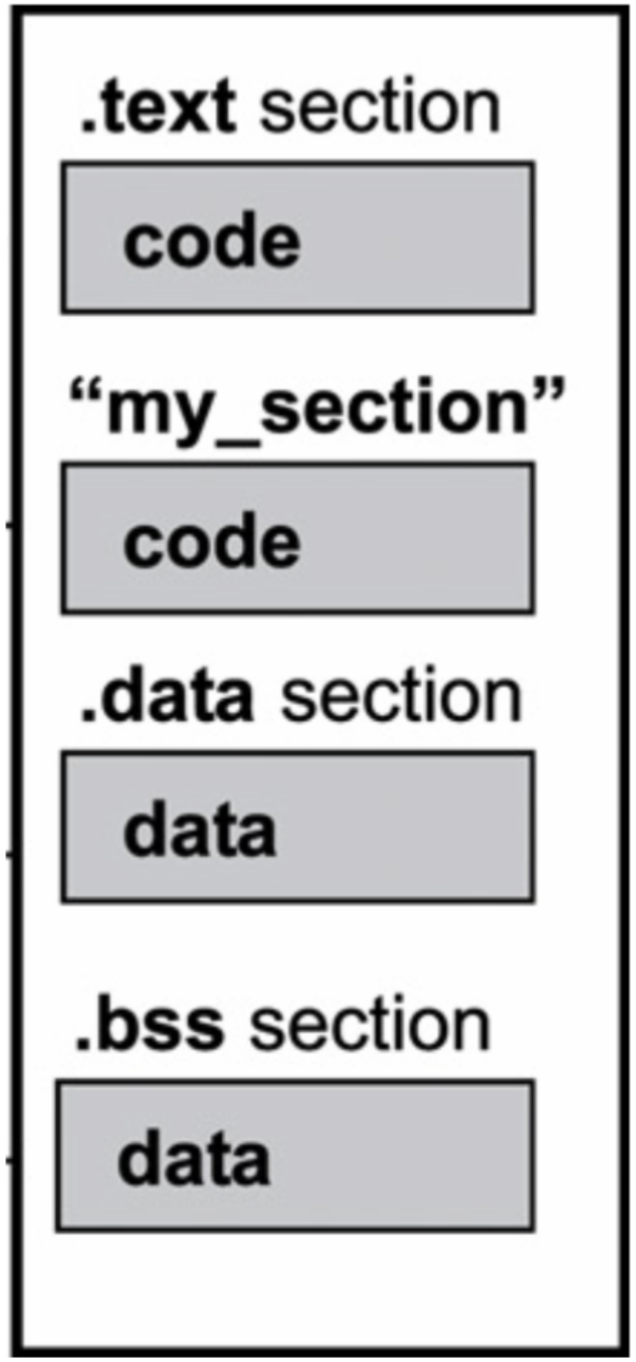
- ELF
  - Executable and linkable format
  - Provides two views: linking and execution
- Type of object files
  - Relocatable file
  - Executable file
  - Shared object file



Linking View	Execution View
ELF Header	ELF Header
Program Header Table (Optional)	Program Header Table
Section 1	Segment 1
....	
Section n	Segment 2
....	
....	....
Section Header Table	Section Header Table (Optional)

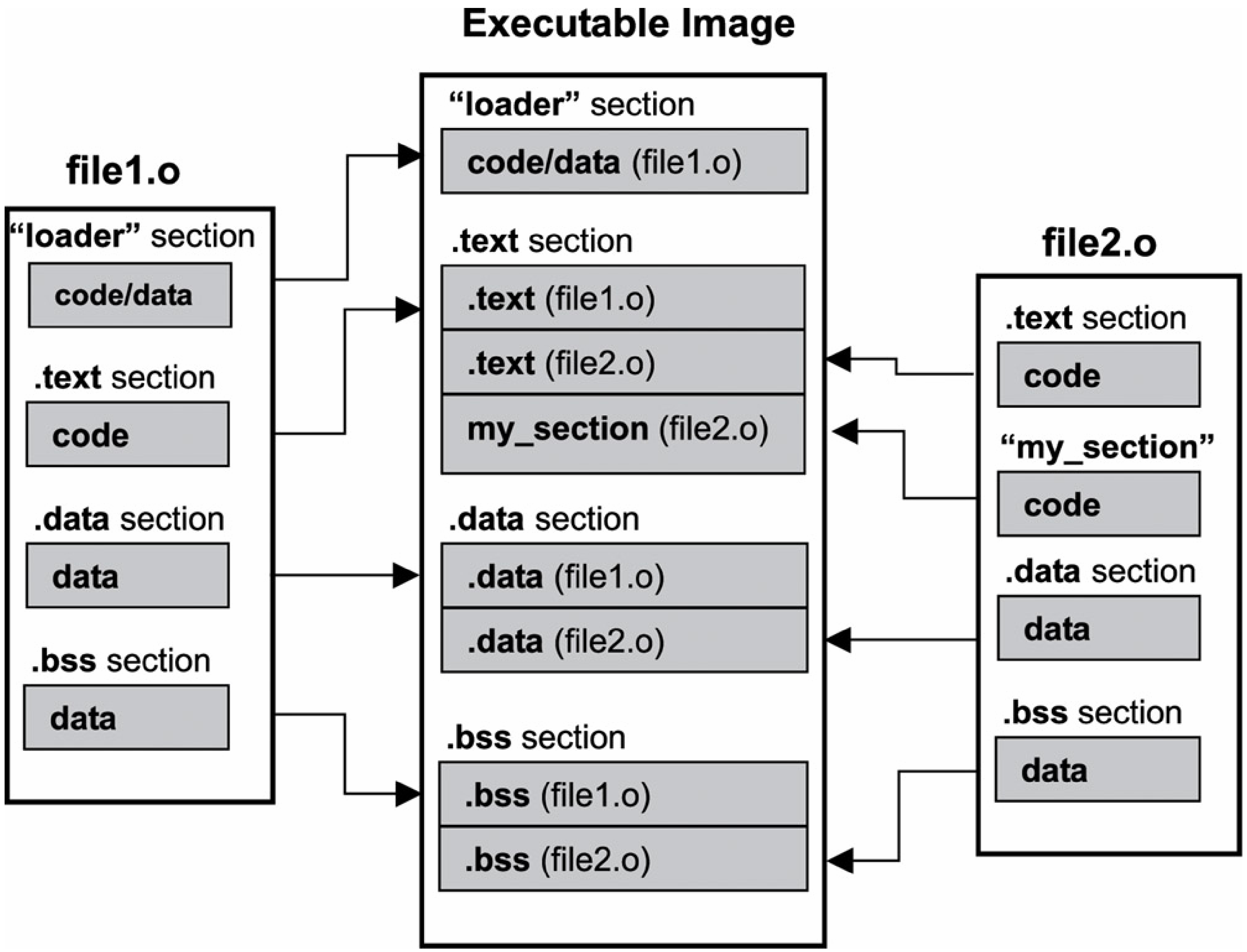
Compiling - Object File

- Object file contains header that describe the rest of the sections
- Blocks regrouped
  - Code blocks in “text” section
  - Initialized global variables in “data” section
  - Uninitialized global variables in “bss” section



Linking

- Linker combines several (relocatable) object (.o) files into a single (executable) object file (.elf)



Linker

- Merges text, data, and bss sections
- Unresolved symbol matching
  - Referred variable is undeclared in one file but is declared in another file
  - Linker will replace with reference to the actual variable
- Normally, the address in the executable image is the absolute address (physical address)

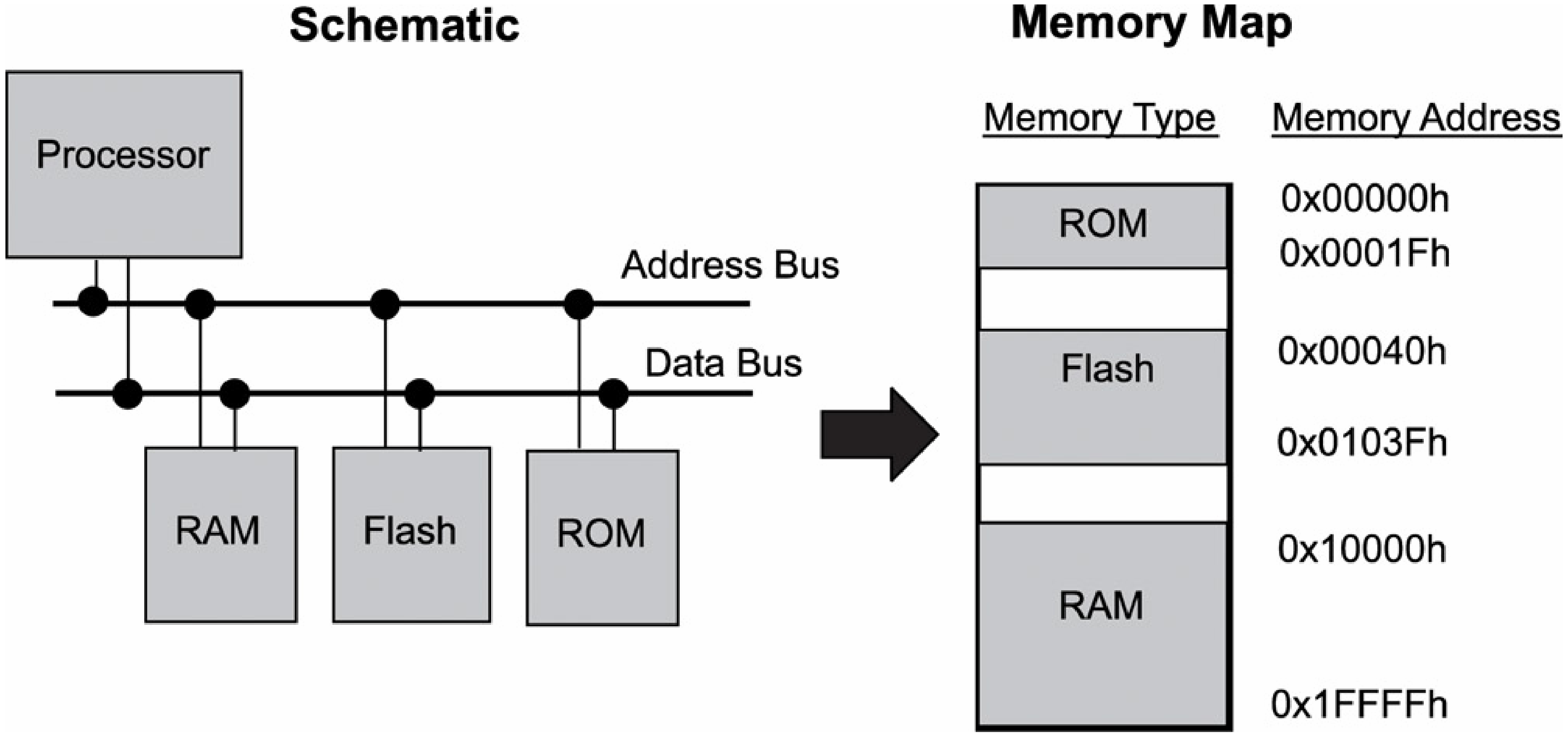
Linker Script

- Set of linker directives that controls how the linker combines the sections and allocates the segments into the target system
- Two common directives supported by most linkers
  - MEMORY
    - Describes the target system's memory map
  - SECTION
    - Specifies how the sections are to be merged and at what location they are to be placed

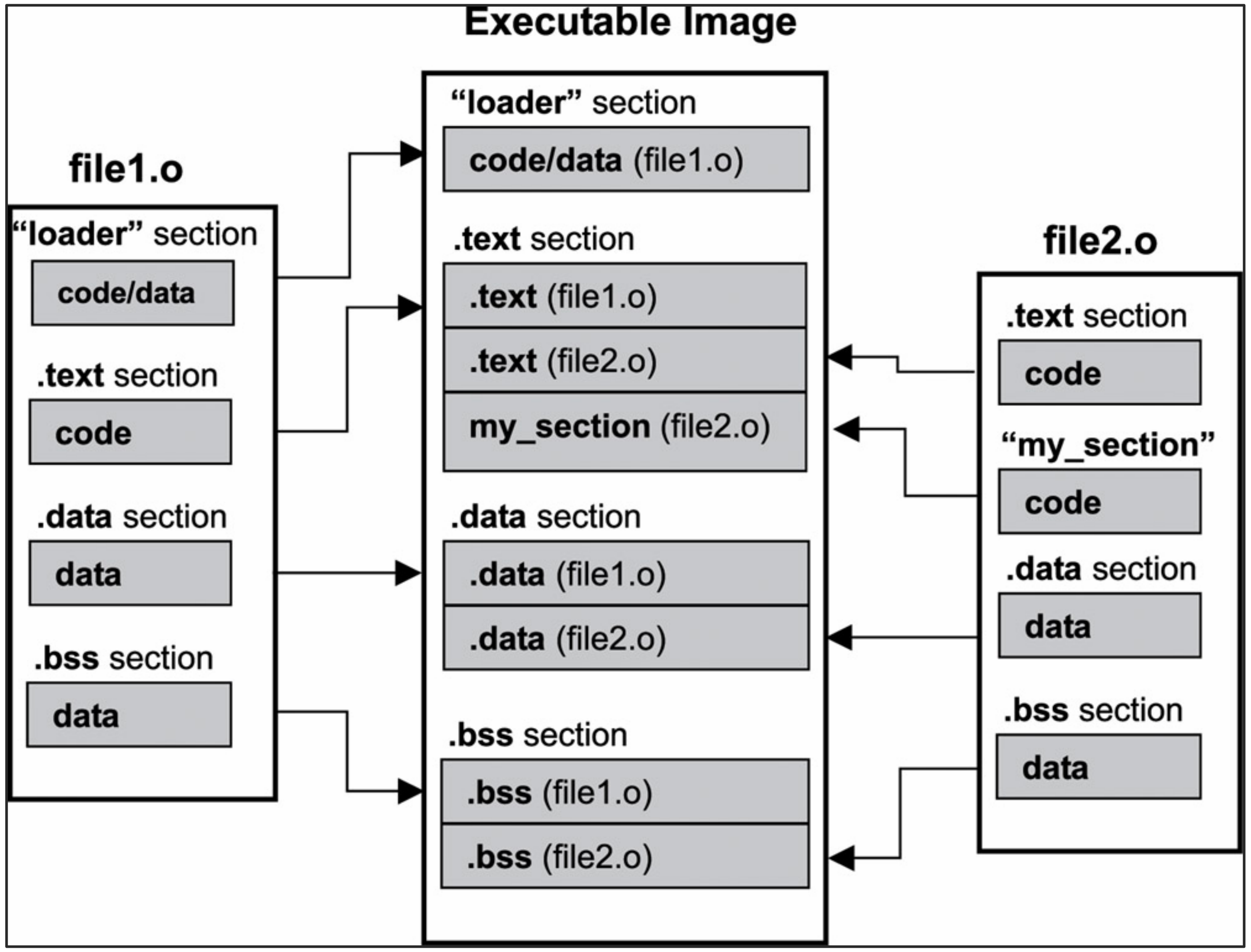
Linker Script Example

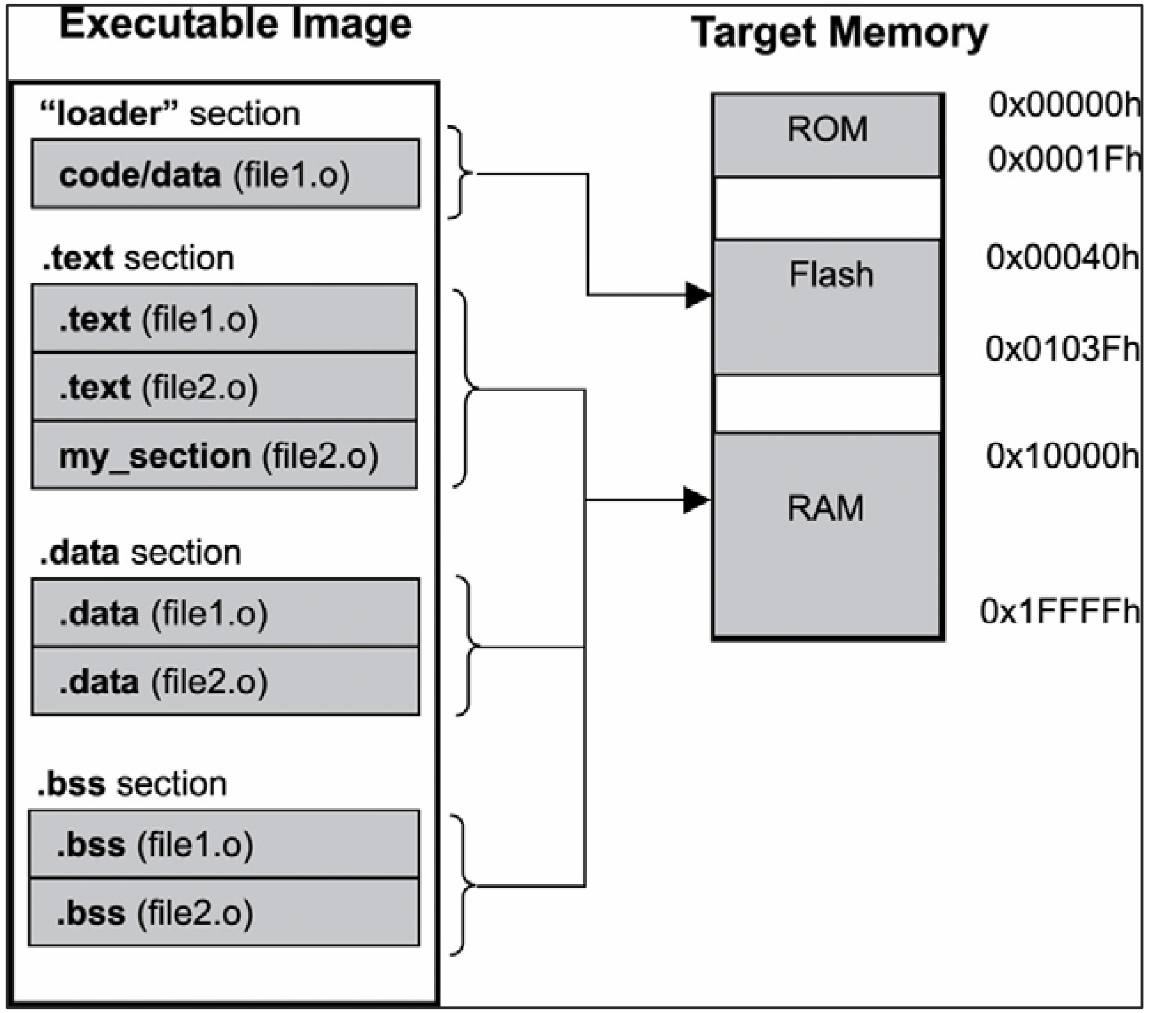
```
MEMORY {
        ROM: origin = 0x0000h, length = 0x0020h
        FLASH: origin = 0x0040h, length = 0x1000h
        RAM: origin = 0x1000h, length = 0x10000h
}

SECTION {
        .text :
        {
                my_section *(.text)
        }
        loader : > FLASH
        GROUP ALIGN (4) :
        {
                .text,
                .data : {}
                .bss : {}
        } >RAM
 }
```



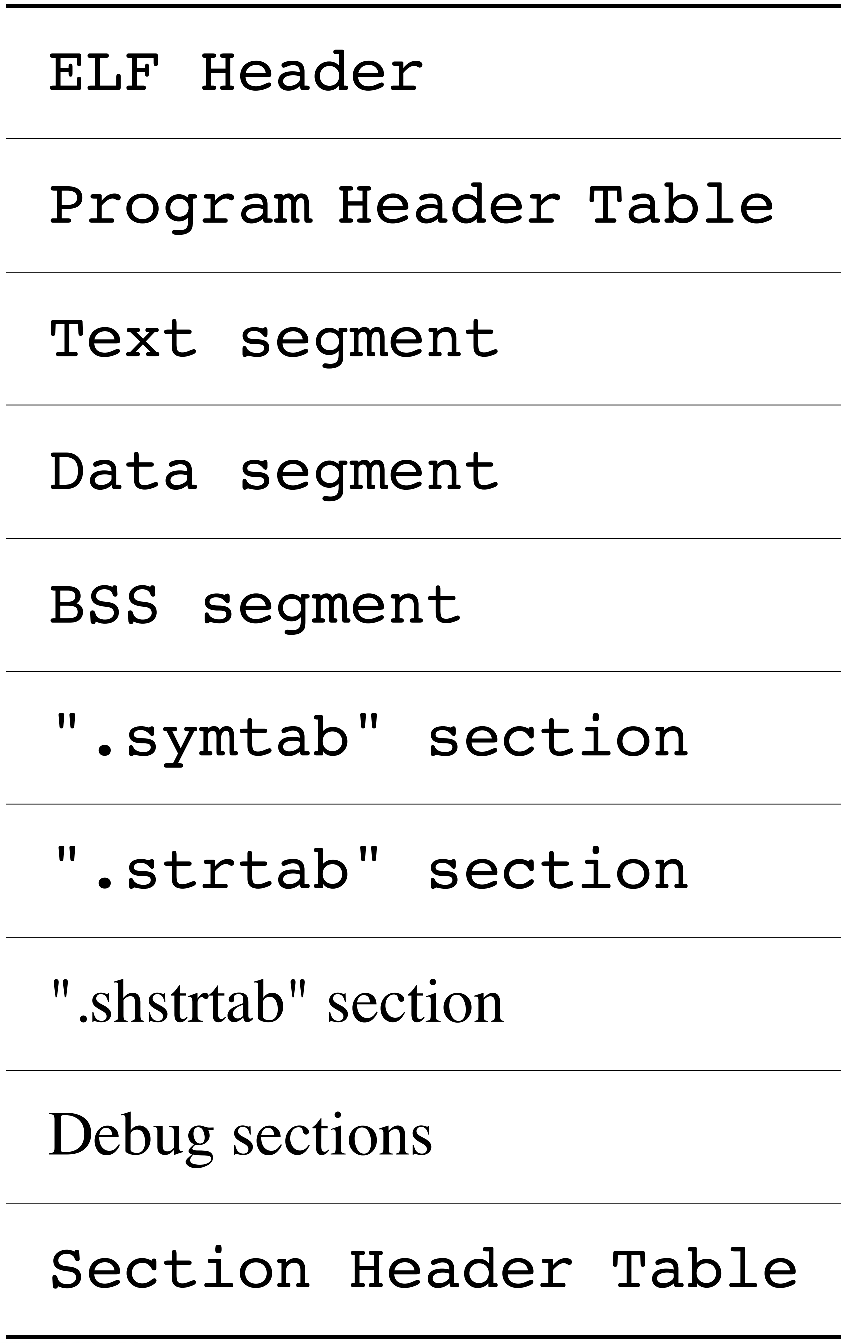
Linker Script Example





Anatomy of Executable Object File

- ARM executable object file
  - Header
    - Describe each section
  - Segment (a group of sections having the same attribute)
    - Text: code for the executable
    - Data: initialized read-write data for the executable
    - BSS: uninitialized data
  - Section
    - Symbol table (.symtab): mapping between address to variable or function name (useful for debugging or reverse engineering)



ELF Header

Program Header Table

Text segment

Data segment

BSS segment

".symtab" section

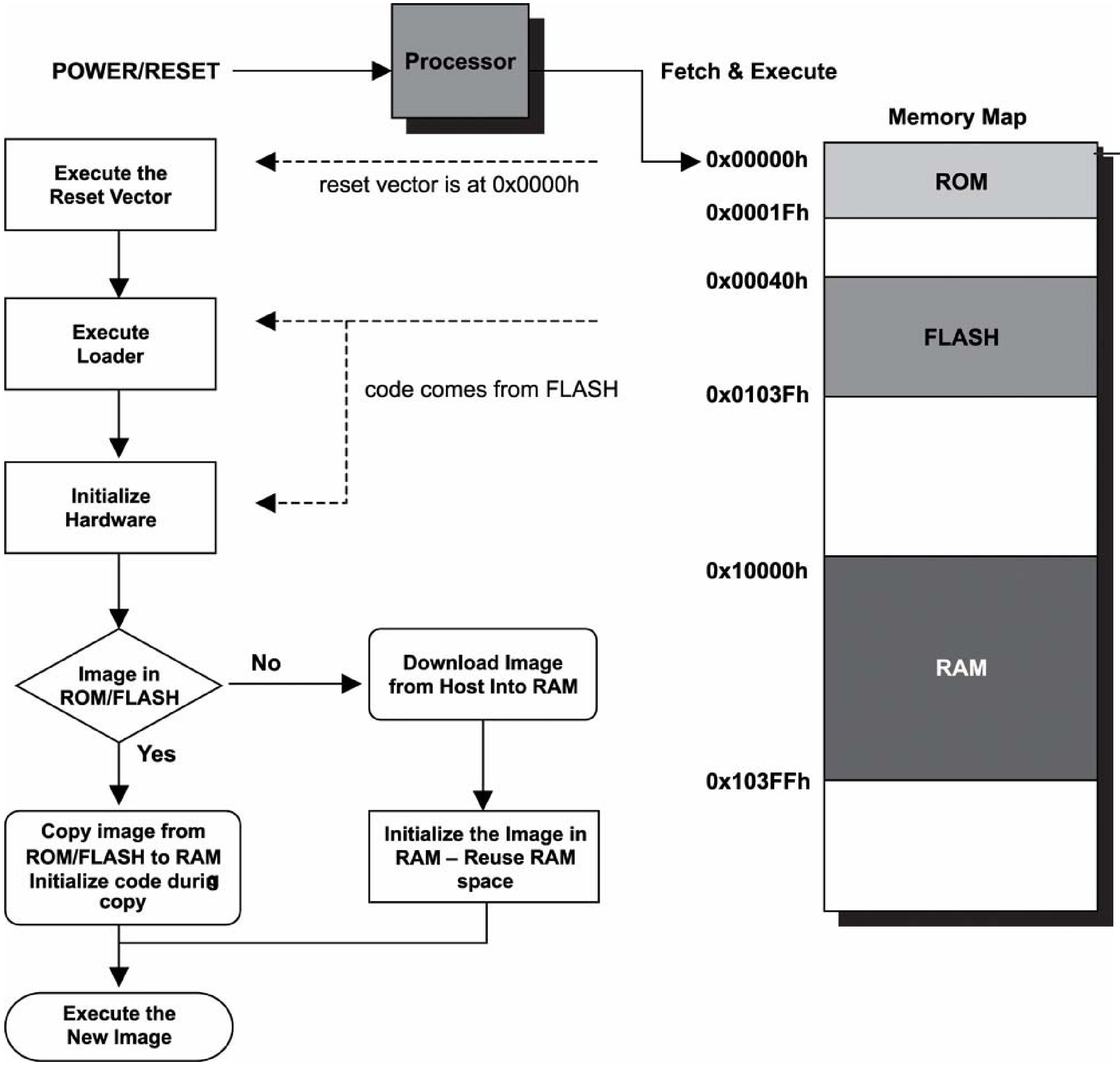
".strtab" section

".shstrtab" section

Debug sections

Section Header Table

Target Boot Scenarios



ldr pc, [pc, #64] ; 00000040 <Reset_Addr>

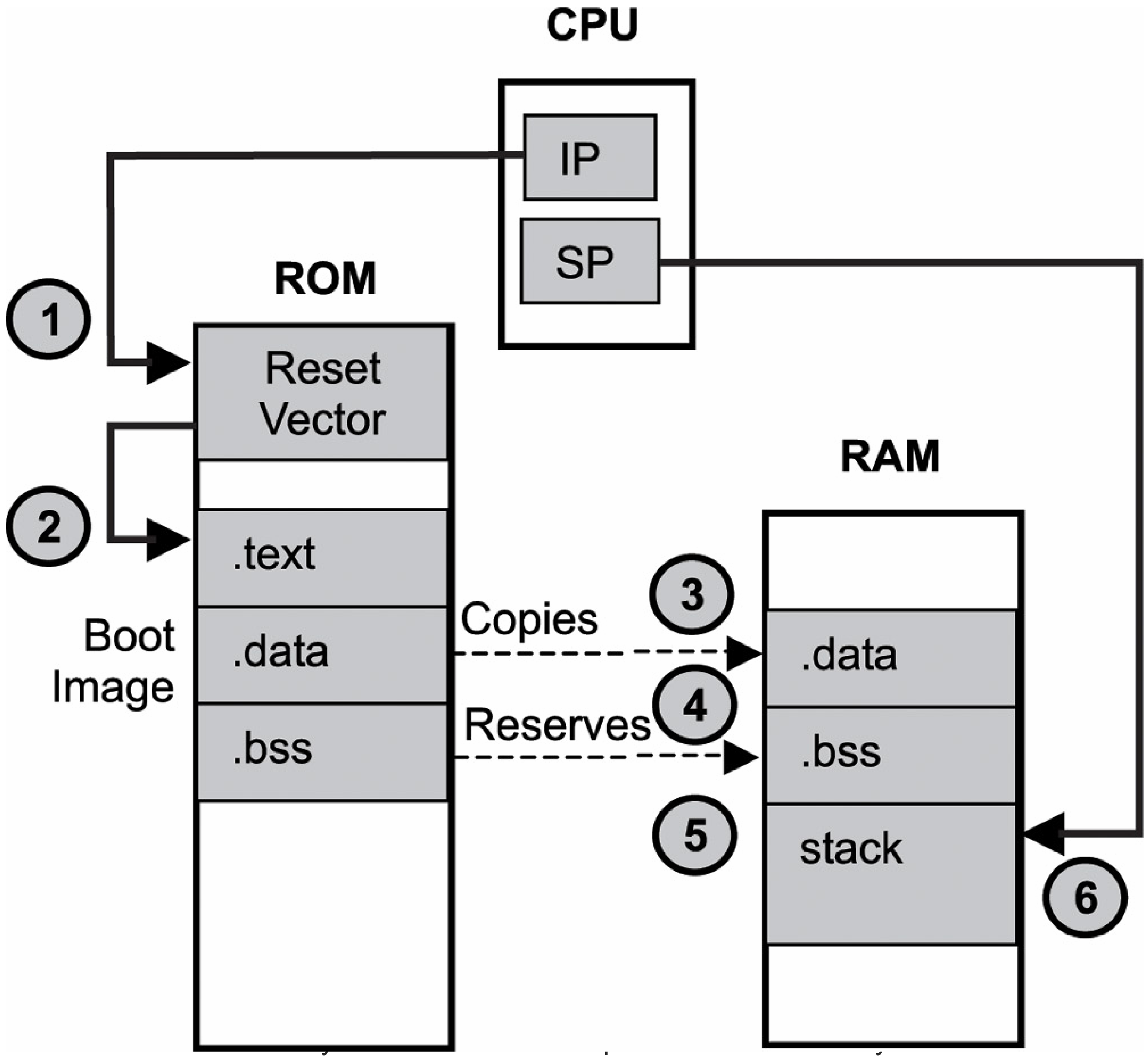
Embedded processors are built-in a small boot code

Target Boot Scenarios

- The application image can begin executing after the boot loader completes its works
- Various boot scenarios
  - XIP (eXecute-In-Place)
    - Executing directly from ROM, using RAM for data
  - Code shadowing
    - Executing from RAM after image transfer from ROM
  - Development phase
    - Executing from RAM after image transfer from the host machine
  - Demand paging
    - Code pages fetched on demand by the embedded OS
    - Example: smartphones with Android OS

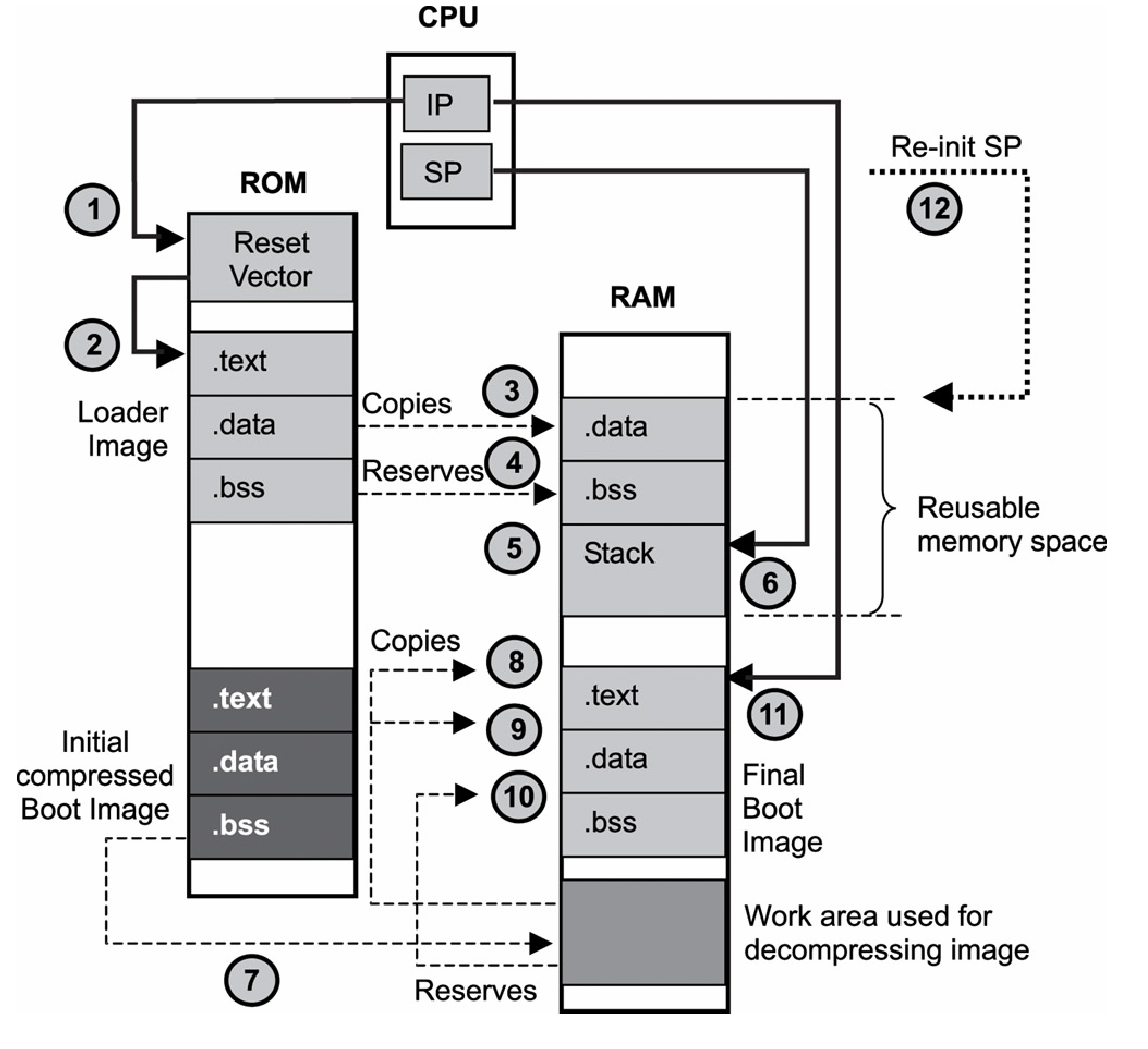
XIP (eXecute-In-Place)

- Executing directly from ROM, using RAM for data



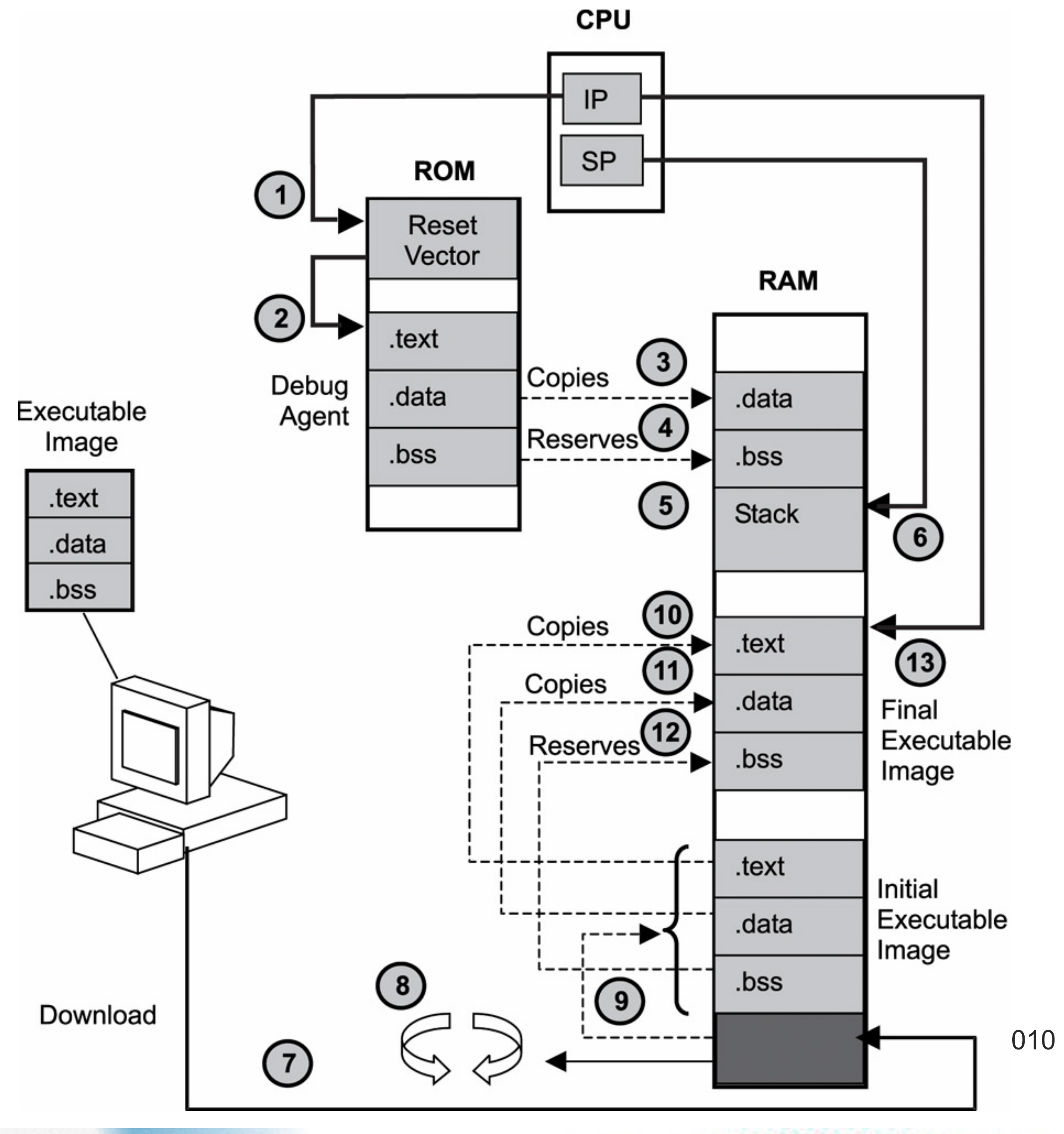
Code Shadowing

- Executing from RAM after image transfer from ROM



Development Phase

- Executing from RAM after image transfer from the host machine



CPU
IP
SP
ROM
Reset Vector
.text
.data
.bss
Debug Agent
RAM
.data
.bss
Stack
.text
.data
.bss
.text
.data
.bss
Copies
Reserves
Copies
Copies
Reserves
Executable Image
.text
.data
.bss
Download
Final Executable Image
Initial Executable Image
1
2
3
4
5
6
7
8
9
10
11
12
13
010