1658 BTL

Bootloading, Application Mapping and Loading Techniques on PIC32

Objective

- Considerations while designing a bootloader on the PIC32
- Mapping the application into different memory regions
- Run-time library loading technique and its applications

MICROCHIP MASTERs 2012

Agenda

- **Bootloader Basics**
- **Designing Bootloader on the PIC32**
  - Memory architecture, Bootloader placement
  - Handling device configuration registers, interrupts
  - Flash controller registers
- **Application Mapping**
  - Lab 1: Understanding the linker scripts
  - Lab 2: Application and Bootloader mapping
  - Merging the Hex Files
- **Run-Time Library Loading Technique**
  - Concept
  - Lab 3: RTLL usage
- **Microchip PIC32 Bootloader Solutions**

Bootloader Basics

- Piece of code which runs at start-up soon after reset

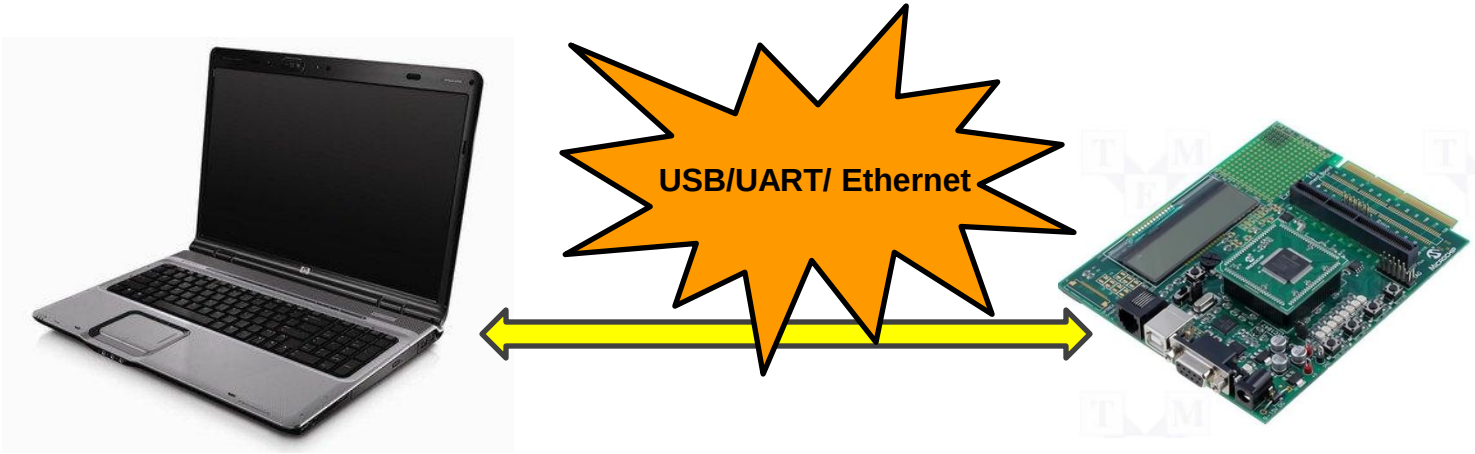
- Limited functionality:
  - Firmware upgrades
  - Application integrity verification
  - Run the application

Why Bootloader?

Programming without Bootloader

- Technical personnel required
- Expensive programmers
- Additional software

Programming with Bootloader



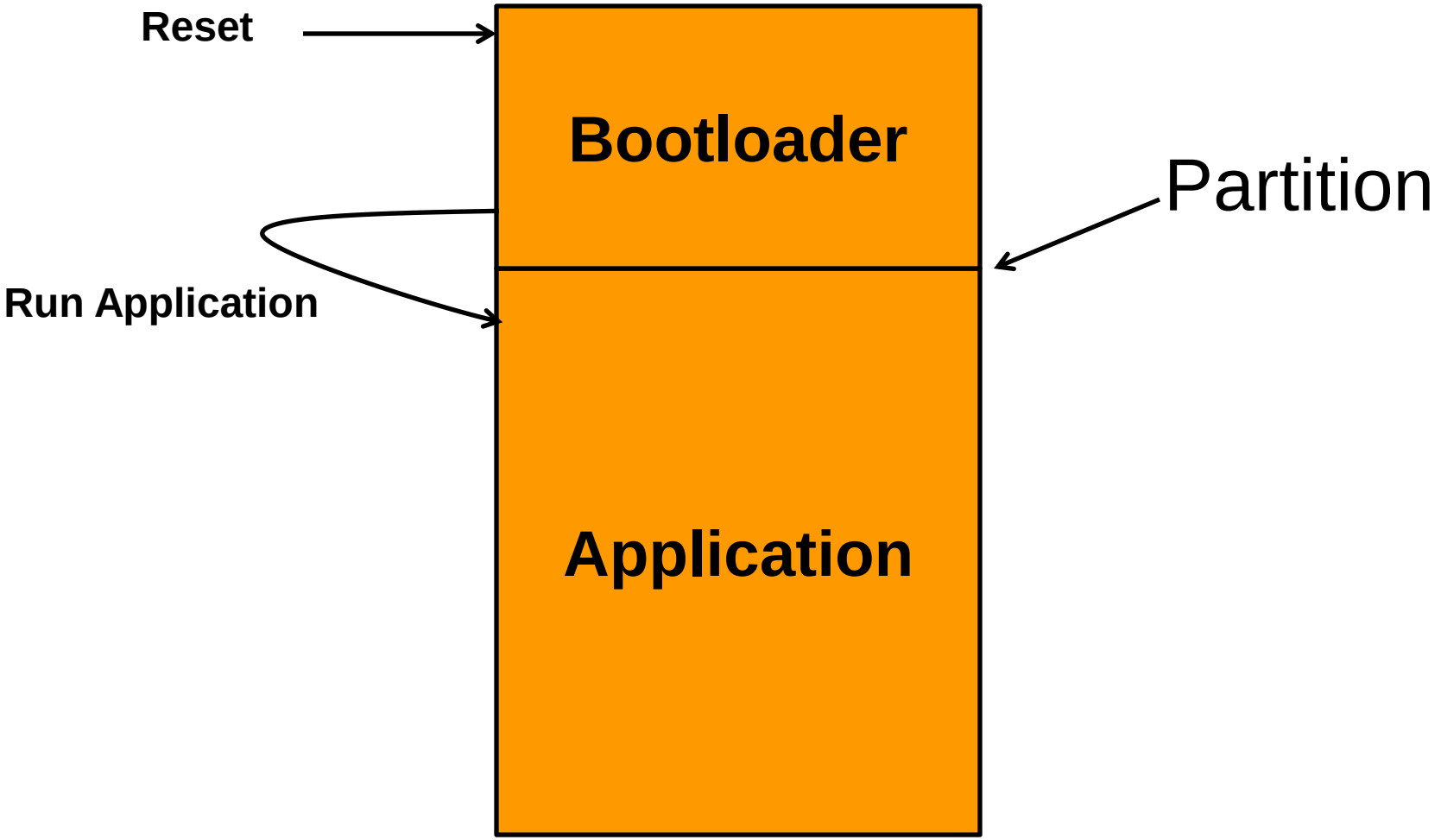
- Firmware upgrade - DIY
- No additional hardware
- Simple PC host application
- Avoids expensive recalls

Real World Examples

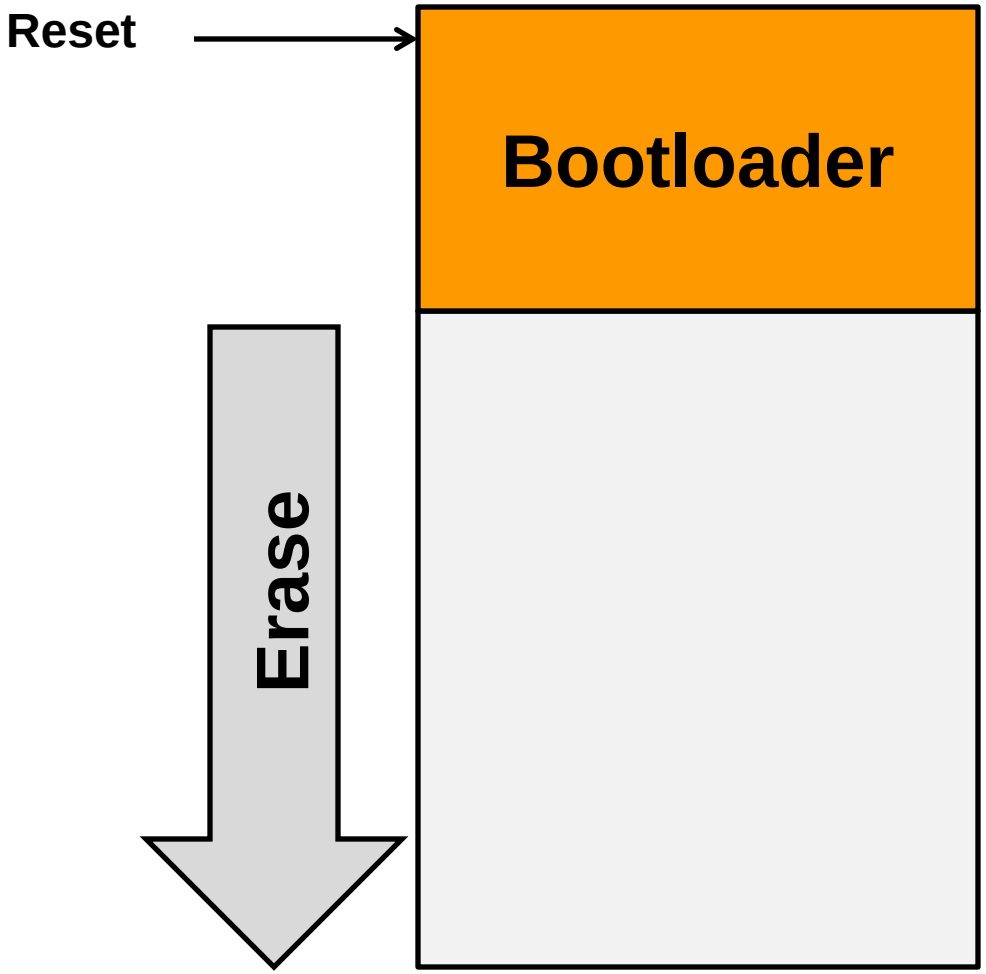
- Firmware upgrades to:
  - MP3 Players
  - Mobile Phones
  - Set-top boxes
  - Satellites
- In PC, OS installation using Bootable CD ROM

Concept

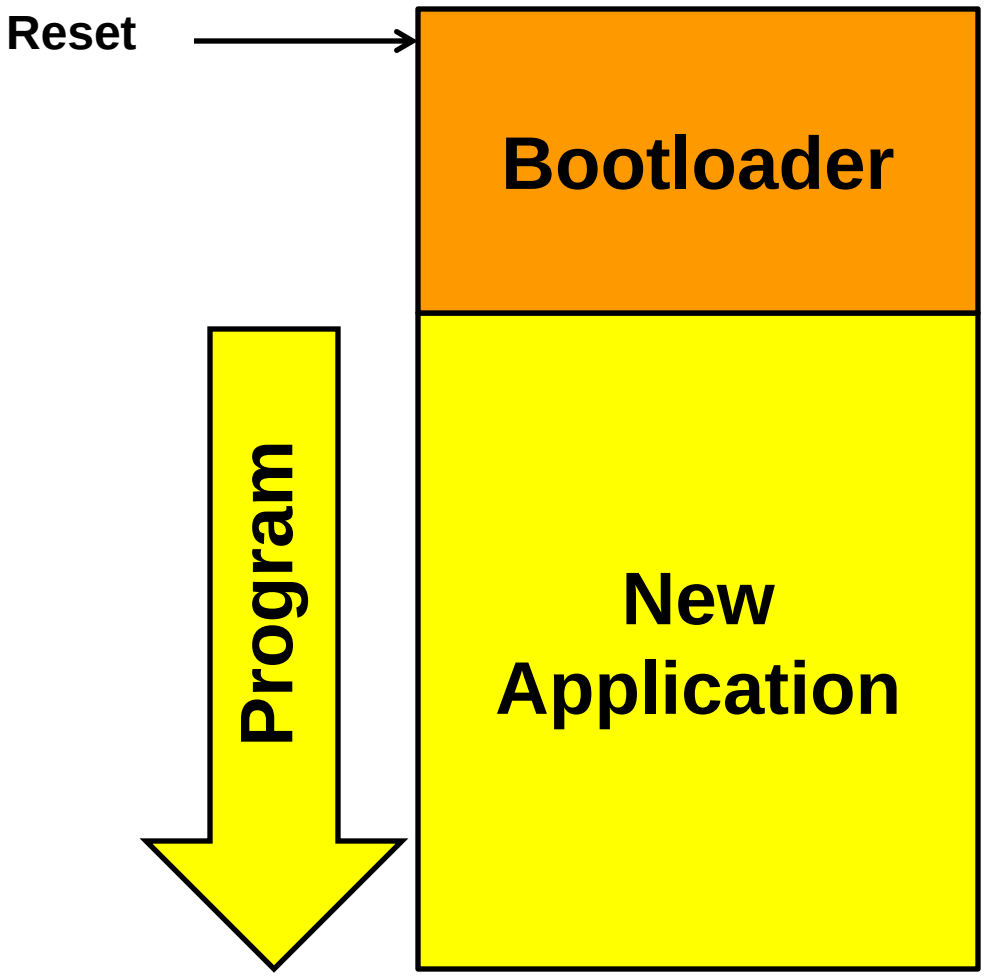
Concept



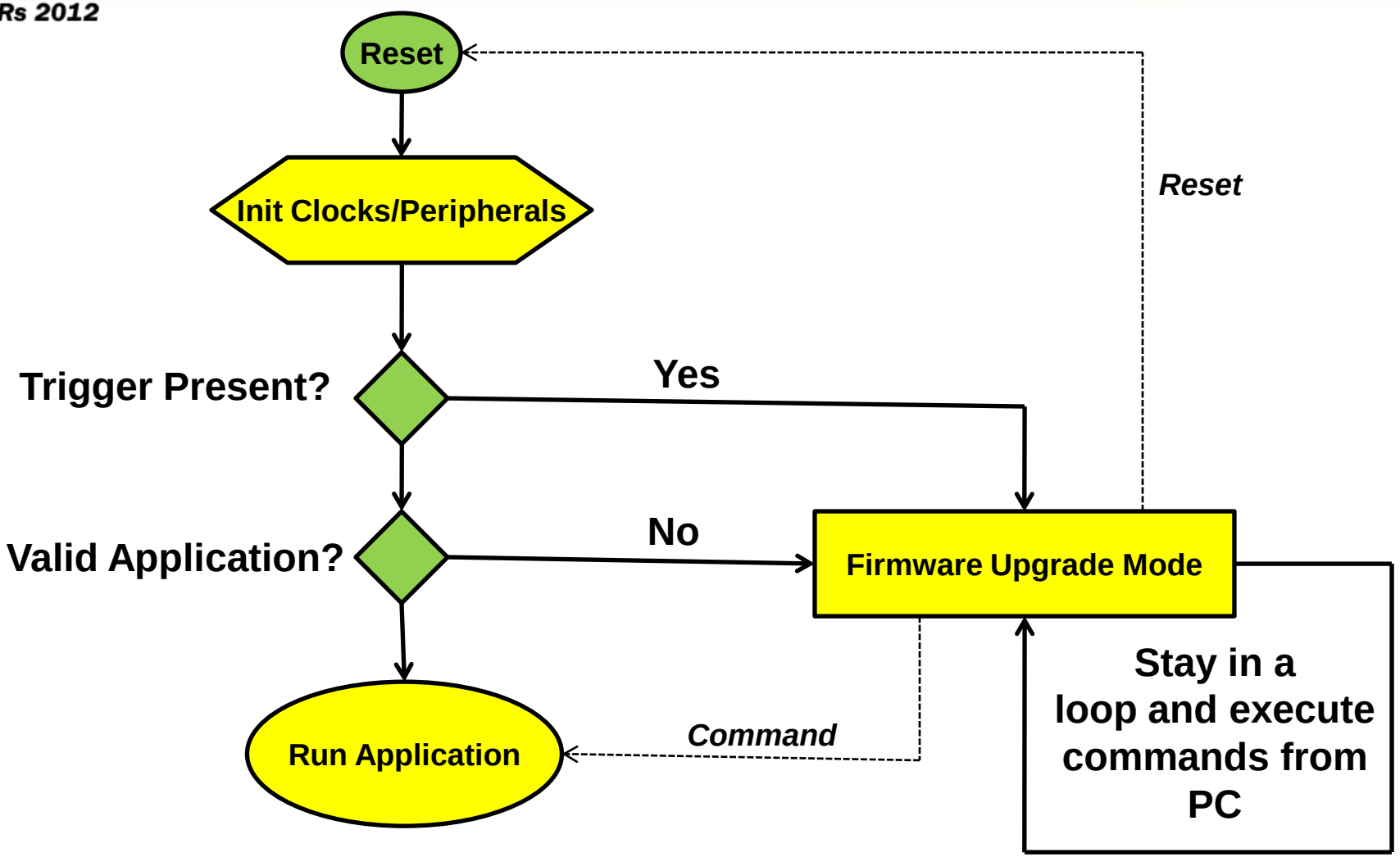
Concept



Concept



Flow Chart



Entering FW Upgrade Mode

Triggers

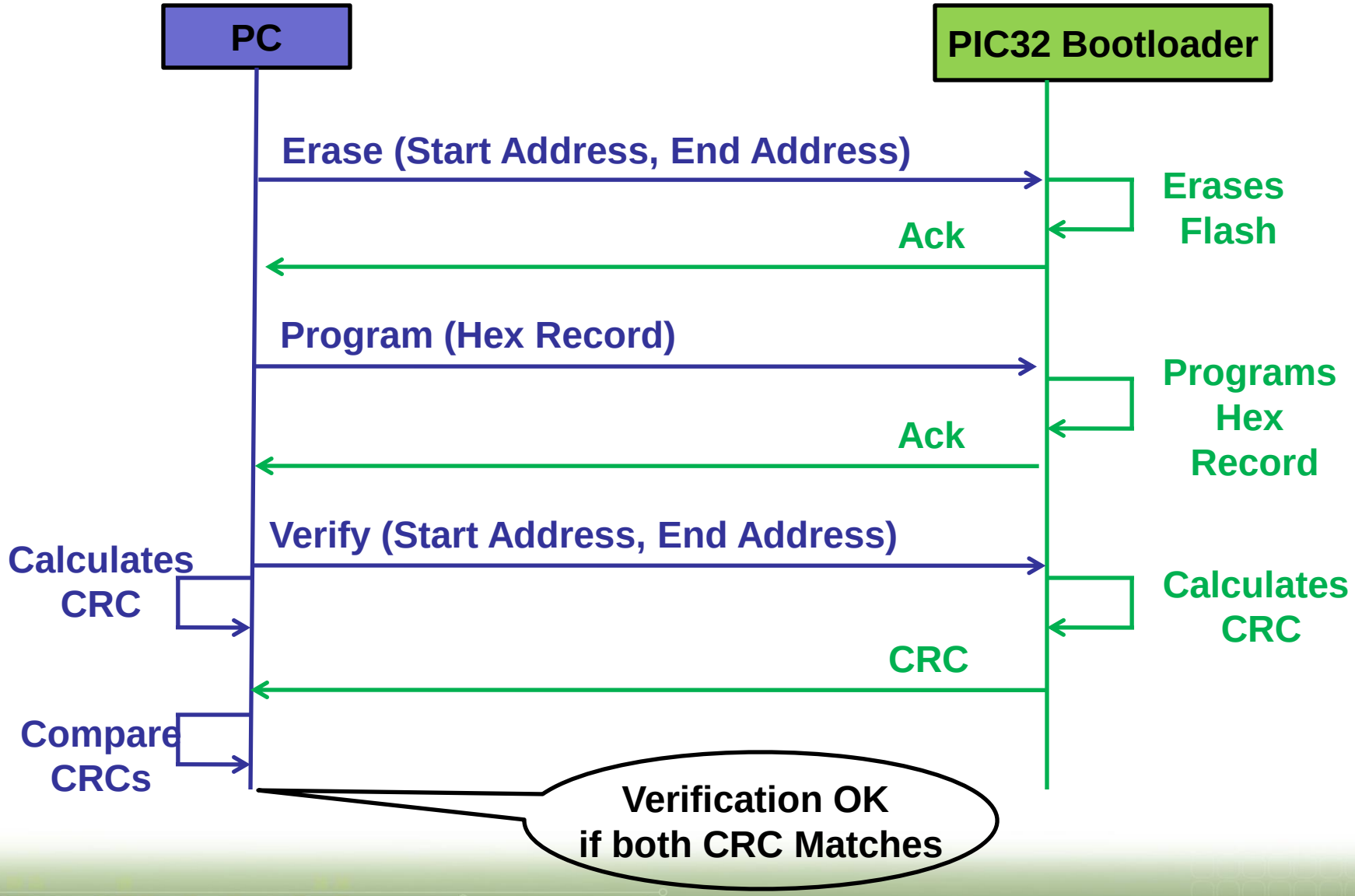
- I/O pin status check
- Wait for PC command with time-out
- No application/application corrupted

Bootloader Commands

Bootloader Commands

- **Mandatory Firmware Upgrade Commands:**
  - Erase Flash
  - Program Flash
  - Verify Flash

Command Sequence



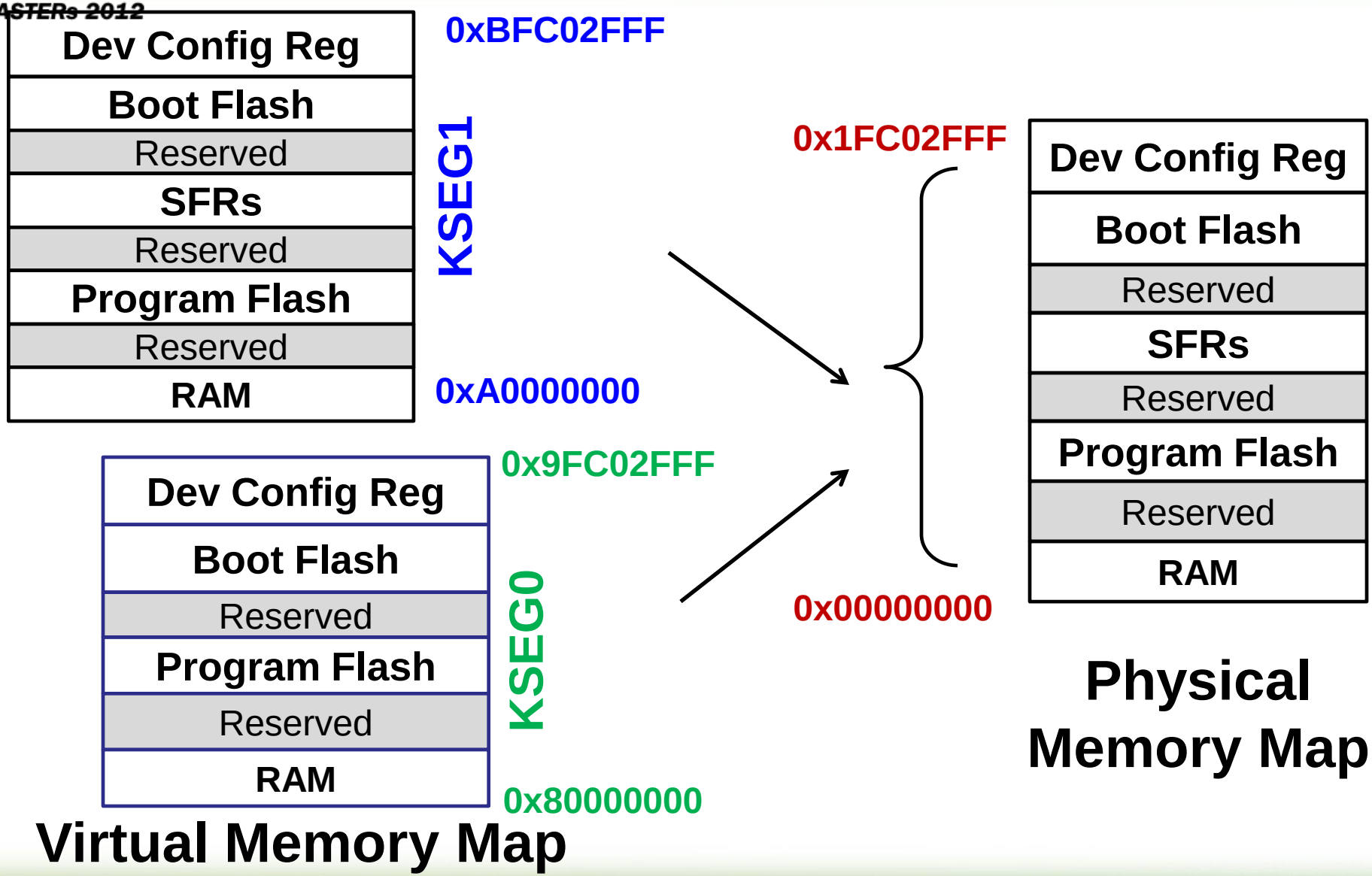
Designing Bootloader on PIC32

What should one know?

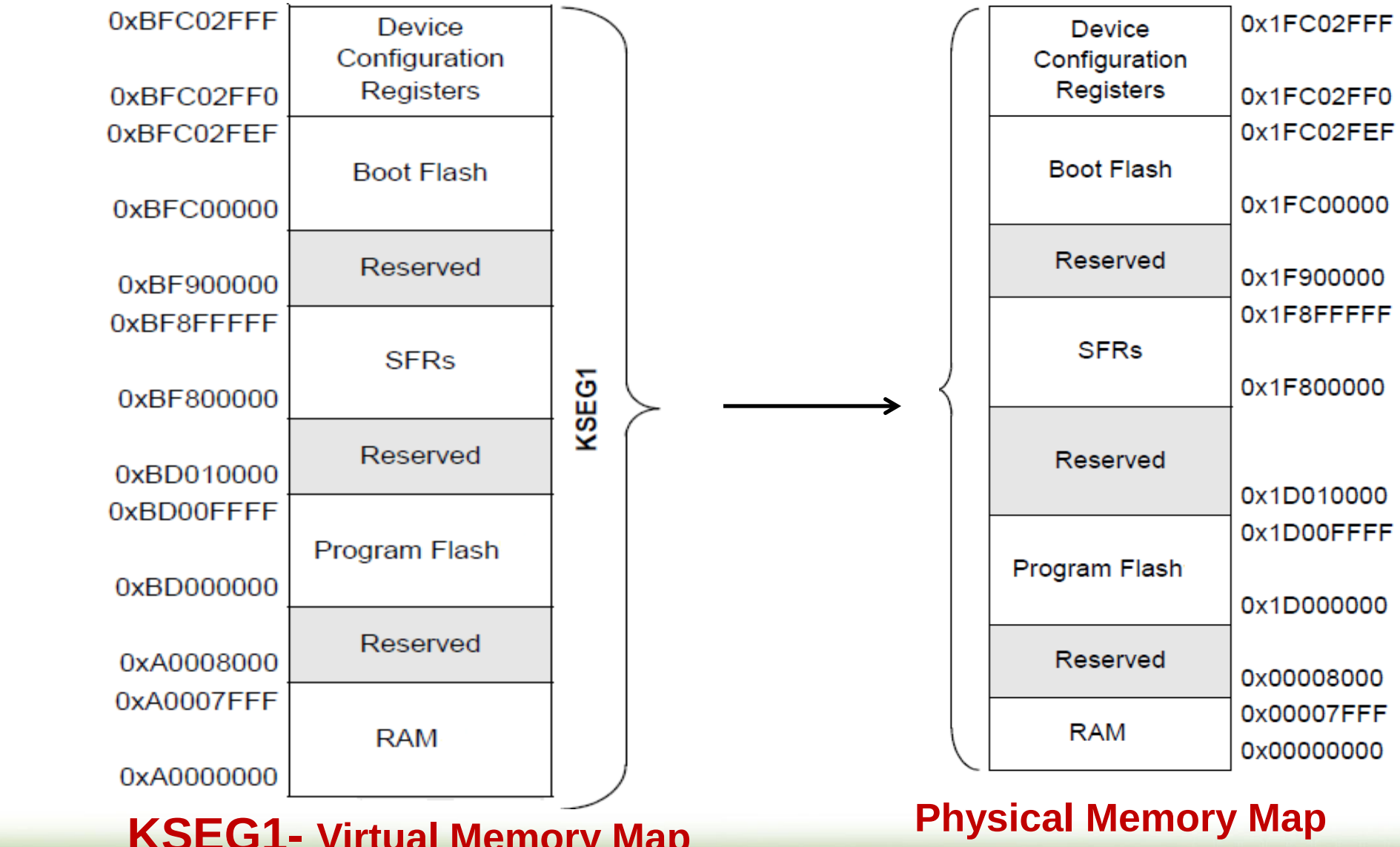
- PIC32 memory architecture
- Partitioning the Flash
- Handling the interrupts
- Handling device configuration registers
- Flash controller registers
- Linker script basics
- Editing the linker scripts

PIC32 Memory Architecture

PIC32 Memory Architecture



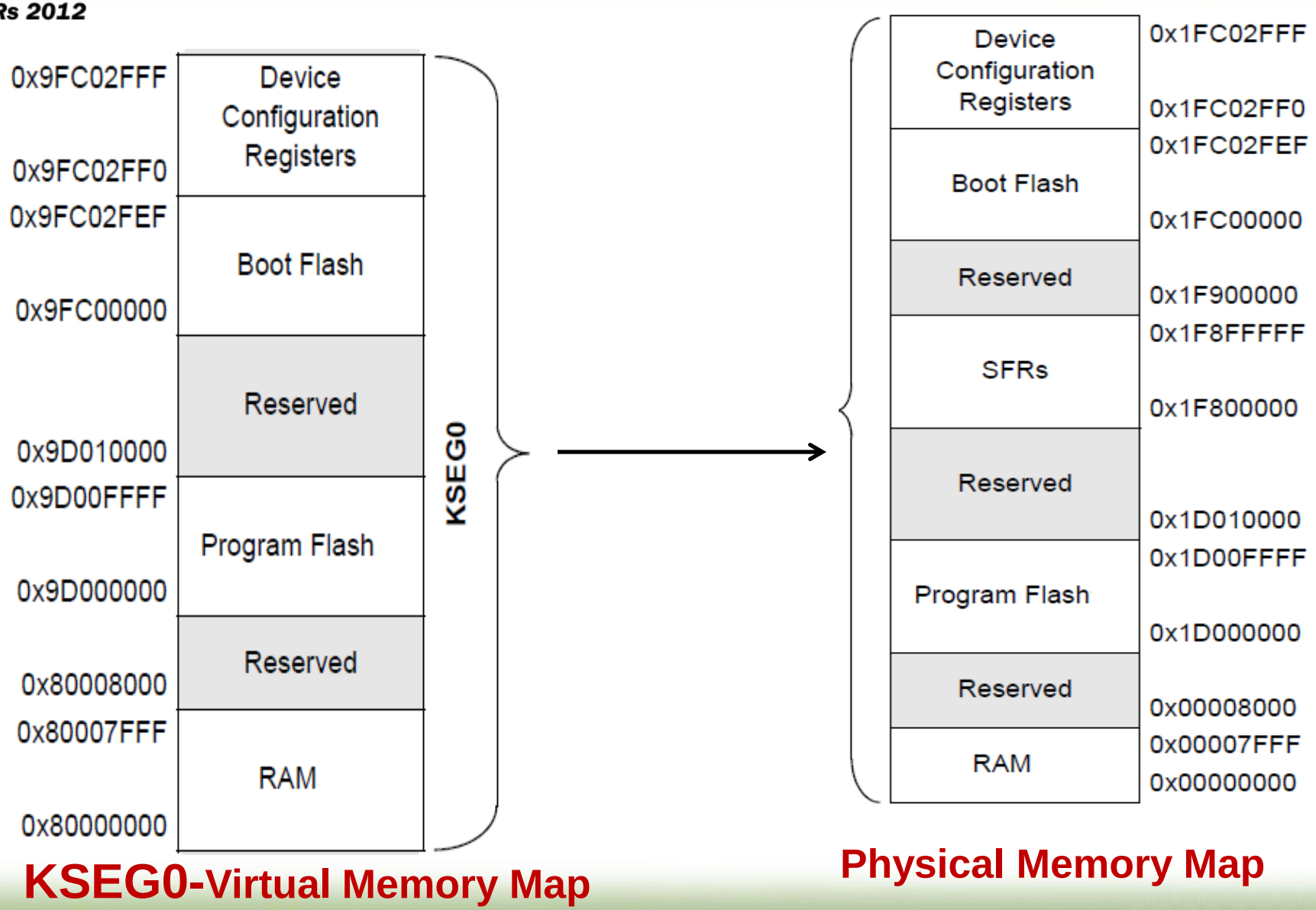
KSEG1 Memory Map



KSEG1- Virtual Memory Map

Physical Memory Map

KSEG0 Memory Map



KSEG0-Virtual Memory Map

Physical Memory Map

PIC32 Memory Architecture

- KSEG0 is cacheable
- KSEG1 is non-cacheable
- Executable code in KSEG0/KSEG1 address space
- Physical Memory access - Bus Masters only
- SFRs not accessible in KSEG0

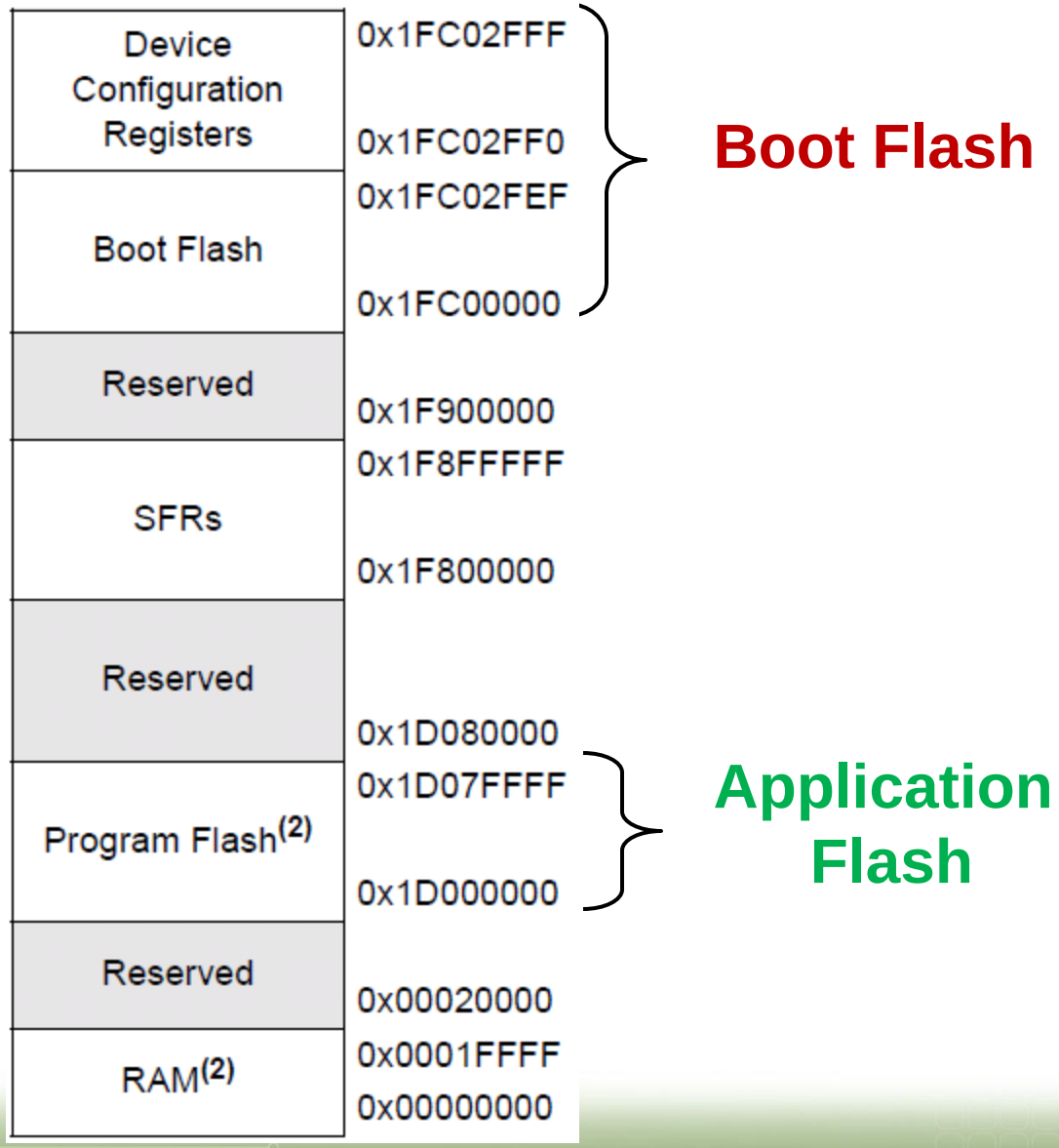
Partitioning the Flash

Flash Partitioning

Option 1

Bootloader

Application



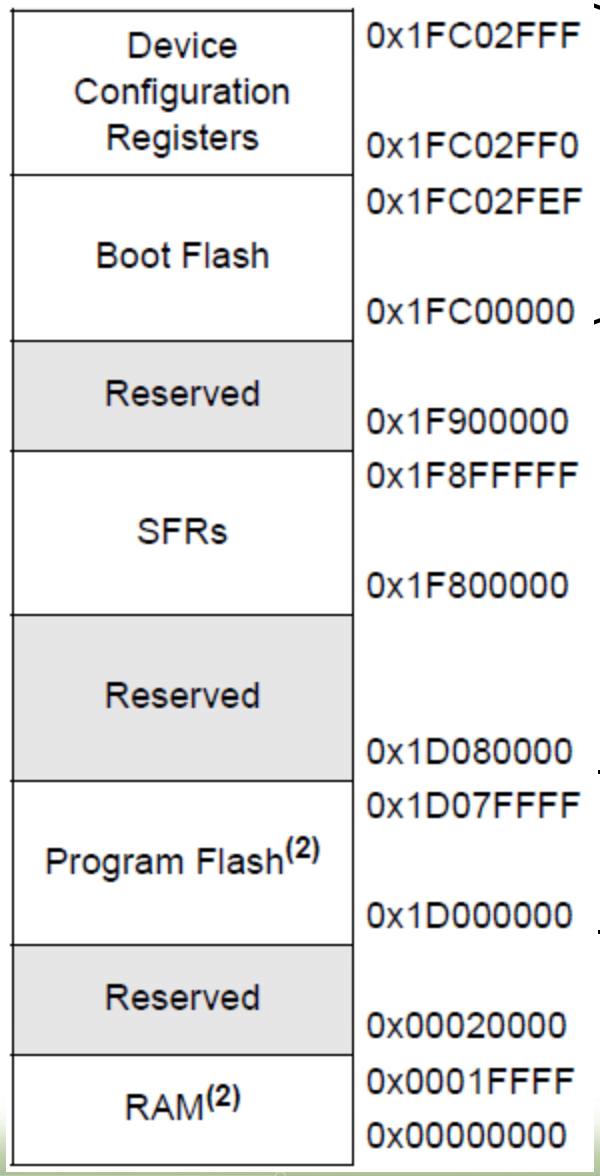
Flash Partitioning

Option 2

Bootloader Part-1 →

Bootloader Part-2

Application →



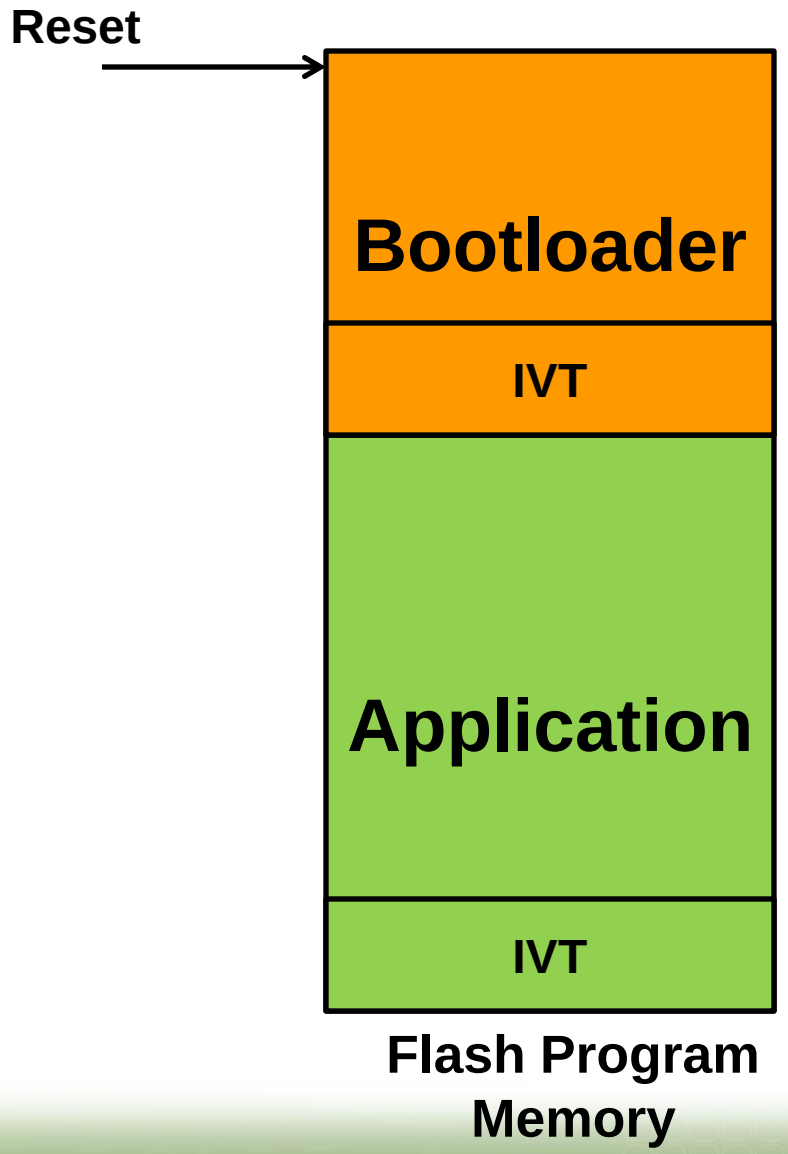
Boot Flash

Application Flash

Handling the Interrupts

Interrupt Handling

- IVT re-mappable at run time
- Bootloader and application can have separate IVTs
- Easy to design



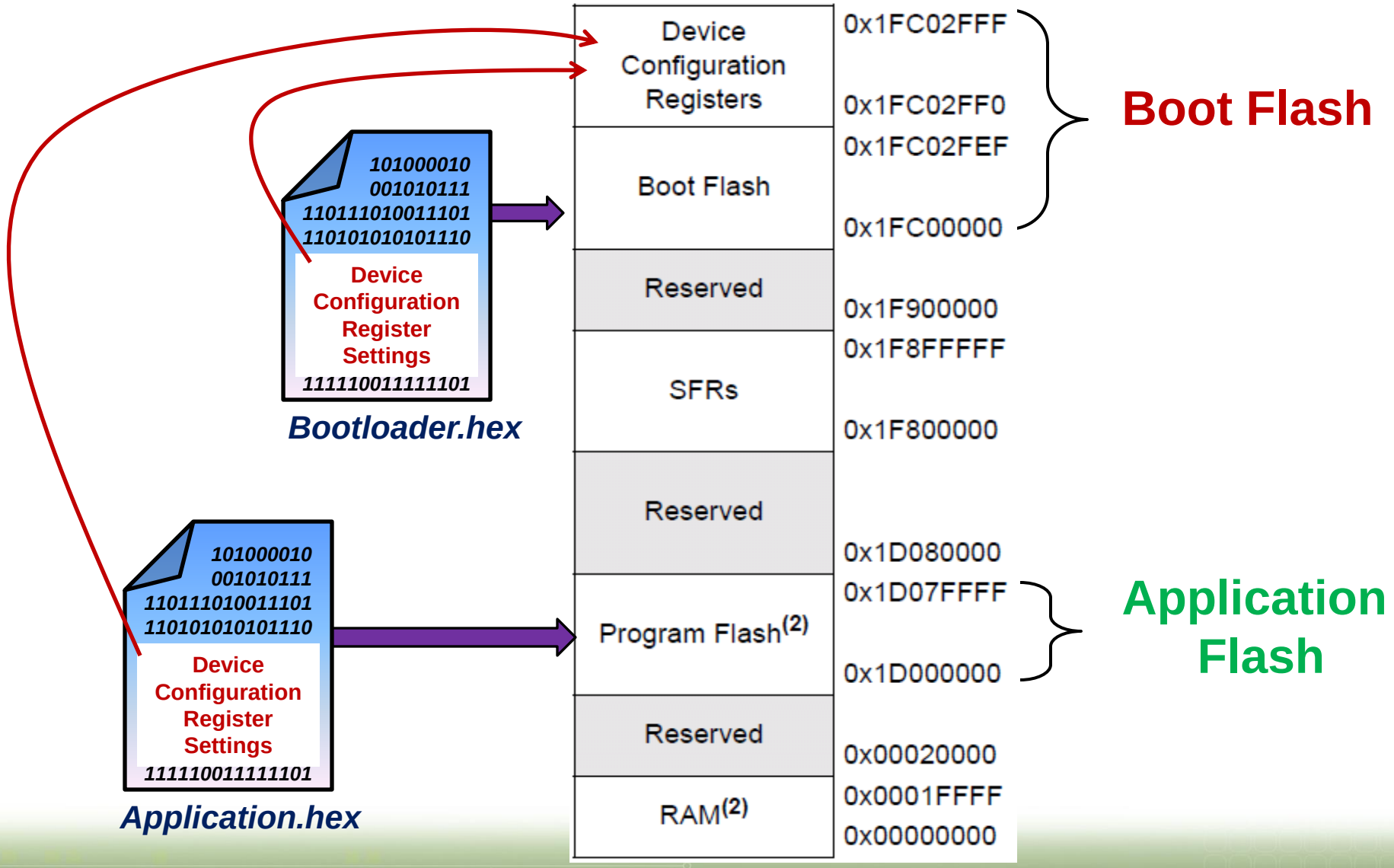
Write Protecting the Bootloader

Write Protecting

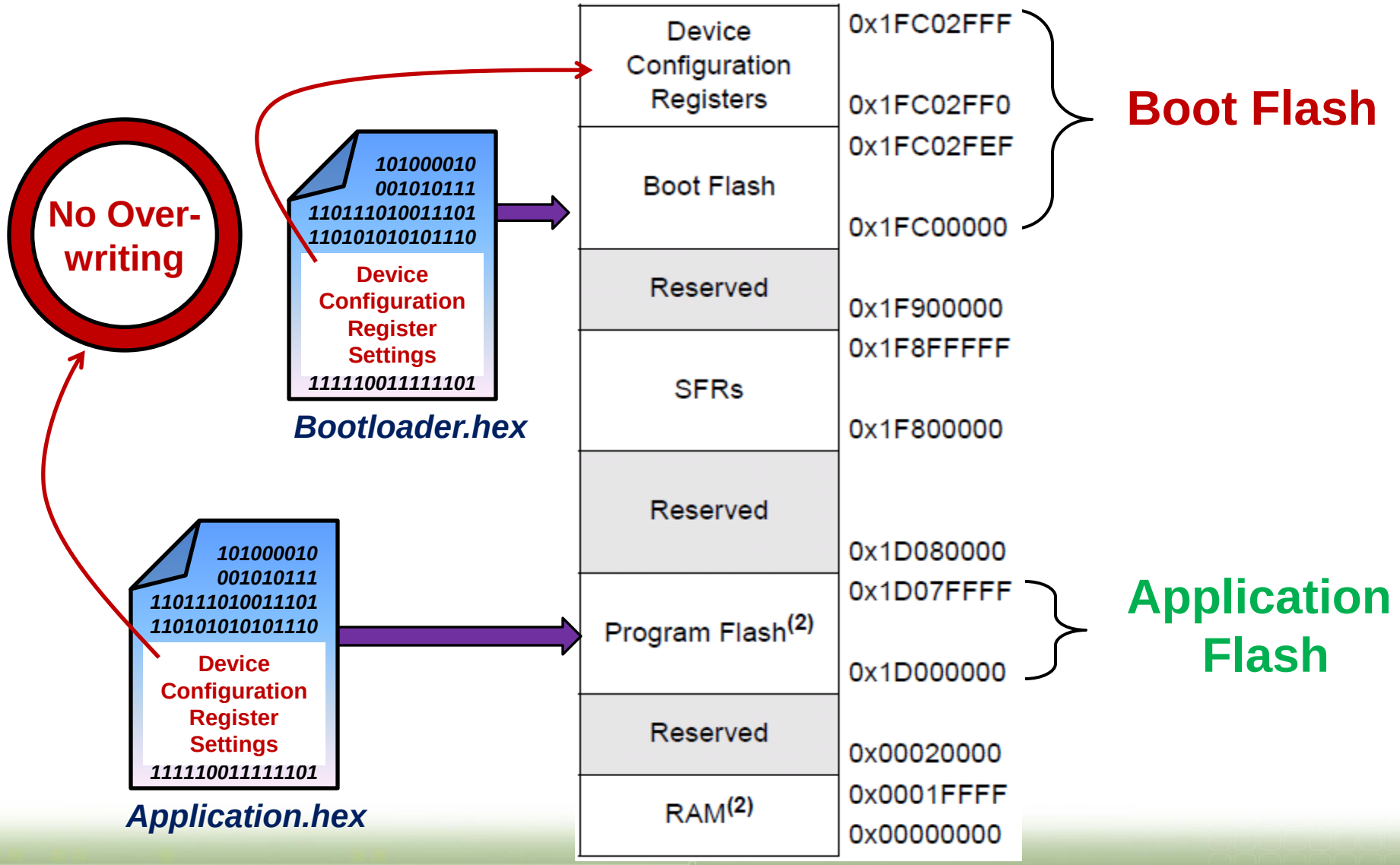
- BWP bit in CFG0 locks Boot Flash
- PWP bits in CFG0 locks selected pages of program Flash

Handling Device Configuration Registers

Handling Device Configuration Registers



Handling Device Configuration Registers



Handling Device Configuration Registers

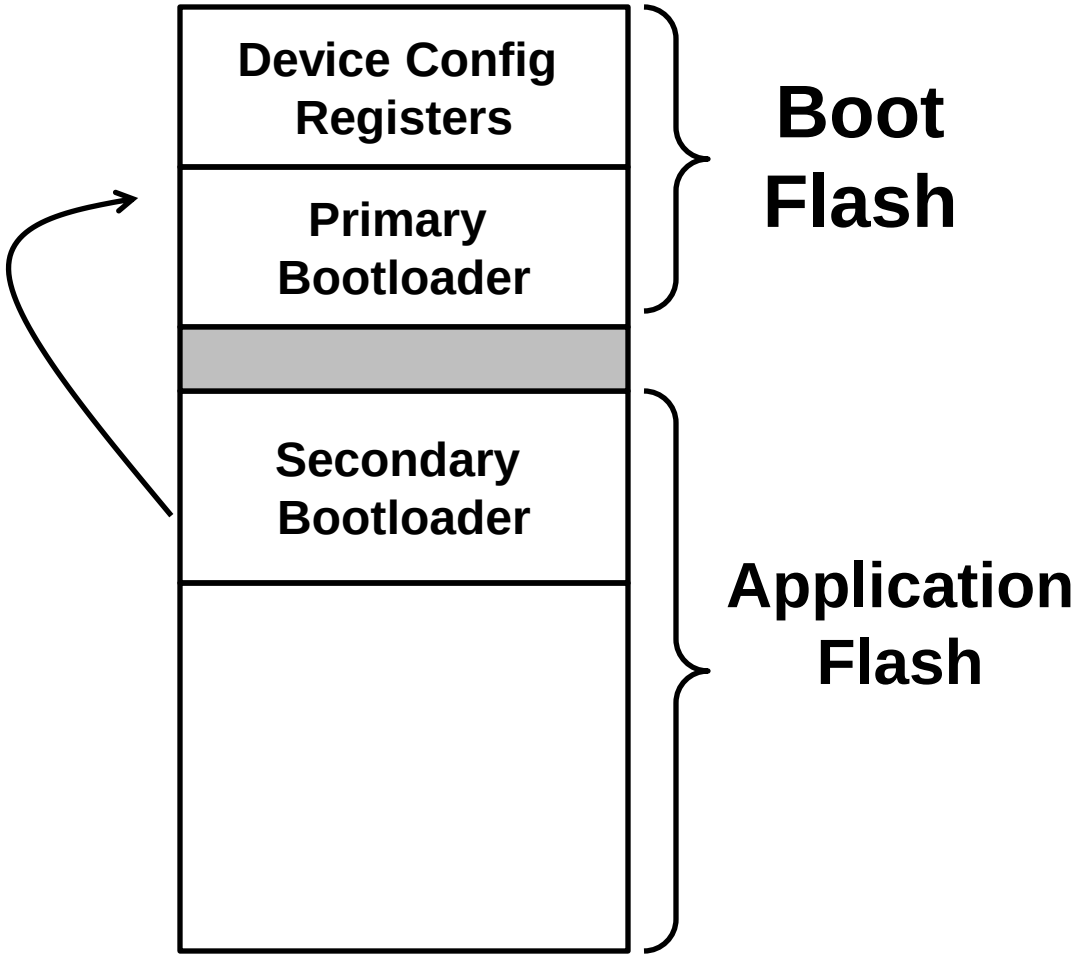
Design hint

- Implement a check in the bootloader to avoid overwriting or erasing configuration registers
- Application re-uses the configuration settings from the Bootloader

How do you upgrade device configuration registers in an emergency?

In an Emergency

- Primary Bootloader downloads secondary bootloader
- Secondary bootloader upgrades Device Configuration Registers + Primary Bootloader
- Take care; device bricks in case of power loss



Flash Controller Registers

Flash Controller - SFRs

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
NVMCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	WR	WREN	WRERR	LVDERR	LVDSTAT	—	—	—
	7:0	—	—	—	—	NVMOP<3:0>			

bit 15: **WR:** **Write Control bit**

bit 14: **WREN:** **Write Enable bit**

bit 13: **WRERR:** **Write Error bit (Read Only)**

bit 12: **LVDERR:** **Low-Voltage Detect Error bit (Read Only)**

bit 11: **LVDSTAT:** **Low-Voltage Detect Status bit (Read Only)**

bit 3-0: **NVMOP<3:0>:** **0101 – Erase Program Flash,**

0011 – Row Program,

0100 – Erase Page,

0001- Word Program

Flash Controller - SFRs

NVMADDR	31:24	NVMADDR<31:24>
	23:16	NVMADDR<23:16>
	15:8	NVMADDR<15:8>
	7:0	NVMADDR<7:0>

bit 31-0: NVMADDR<31:0>: **Flash Address bits**

Page Erase: **Address identifies the page to erase**

Row Program: **Address identifies the row to program**

Word Program: **Address identifies the word to program**

Bulk/PFM Erase: **Address is ignored**

Flash Controller - SFRs

NVMDATA	31:24	NVMDATA<31:24>
	23:16	NVMDATA<23:16>
	15:8	NVMDATA<15:8>
	7:0	NVMDATA<7:0>

bit 31-0 NVMDATA<31:0>:

Flash Programming Data bits. Used only for word programming

Flash Controller - SFRs

NVMKEY	31:24	NVMKEY<31:24>
	23:16	NVMKEY<23:16>
	15:8	NVMKEY<15:8>
	7:0	NVMKEY<7:0>

bit 31-0 NVMKEY<31:0>:

Sequence to unlock any NV Operation.

1) Write 0xAA996655 to NVMKEY

2) Write 0x556699AA to NVMKEY

Flash Controller - SFRs

NVMSRCADDR	31:24	NVMSRCADDR<31:24>
	23:16	NVMSRCADDR<23:16>
	15:8	NVMSRCADDR<15:8>
	7:0	NVMSRCADDR<7:0>

NVMSRCADDR<31:0>:

Source Data Address bits

Physical address of the data to be programmed into the Flash when the NVMOP<3:0> bits are set to perform row programming

Flash Controller - SFRs

Page Erase Steps

- NVMADDR = Address of Flash Page to Erase
- NVMCON ->WREN = 1 (Enable Write)
- NVMCON->NVMOP = 4 (Set Erase Operation)
- NVMKEY = 0xAA996655 (Unlock Sequence)
- NVMKEY = 0x556699AA
- NVMCON->WR = 1 (Start Erase Operation)
- Wait for WR bit to clear
- Check NVMCON->WRERR/LVDERR bits

Flash Controller - SFRs

Word Programming Steps

- NVMADDR = Word Address of Flash
- NVMDATA = Word to write
- NVMCON ->WREN = 1 (Enable Write)
- NVMCON->NVMOP = 1 (Word Programming)
- NVMKEY = 0xAA996655 (Unlock Sequence)
- NVMKEY = 0x556699AA
- NVMCON->WR = 1 (Start Word Programming)
- Wait for WR bit to clear
- Check NVMCON->WRERR/LVDERR bits

Flash Controller - SFRs

Row Programming Steps

- NVMADDR = Row Address
- NVMSRCADDR = Source Buffer Address
- NVMCON ->WREN = 1 (Enable Write)
- NVMCON->NVMOP = 3 (Set Row Programming)
- NVMKEY = 0xAA996655 (Unlock Sequence)
- NVMKEY = 0x556699AA
- NVMCON->WR = 1 (Start Row Programming)
- Wait for WR bit to clear
- Check NVMCON->WRERR/LVDERR bits

Flash Controller - SFRs

Bulk Erase Program Memory

- NVMCON ->WREN = 1 (Enable Write)
- NVMCON->NVMOP = 5 (Bulk Erase Operation)
- NVMKEY = 0xAA996655 (Unlock Sequence)
- NVMKEY = 0x556699AA
- NVMCON->WR = 1 (Start Bulk Erase)
- Wait for WR bit to clear
- Check NVMCON->WRERR/LVDERR bits

Designing PIC32 Bootloader

So far we have learned…

- PIC32 Memory Architecture
- Ways to Partition the Flash
- Handling Interrupts
- Handling Device Configuration Registers
- Flash Controller

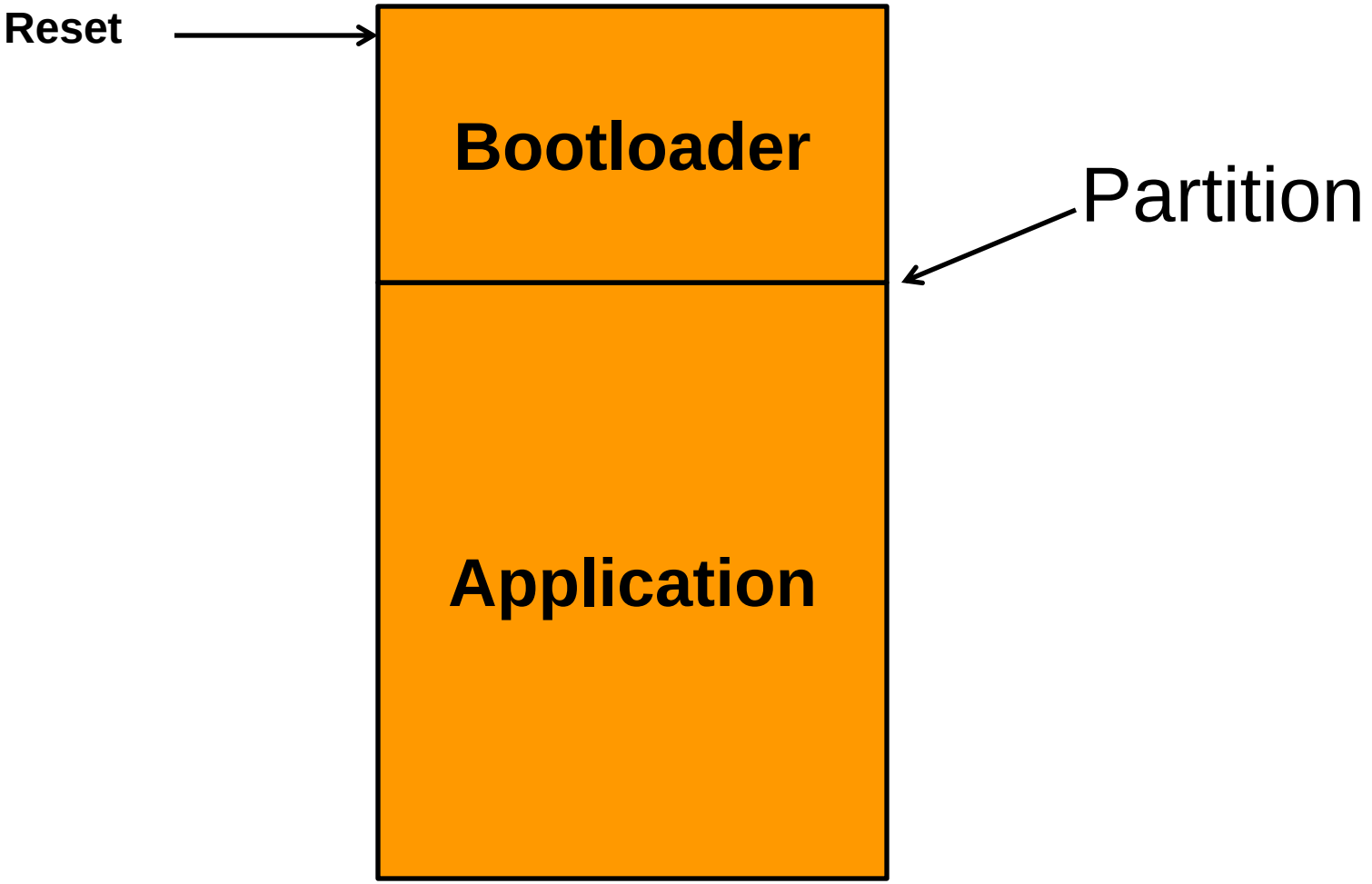
Mapping Application/Bootloader

Mapping Application/ Bootloader

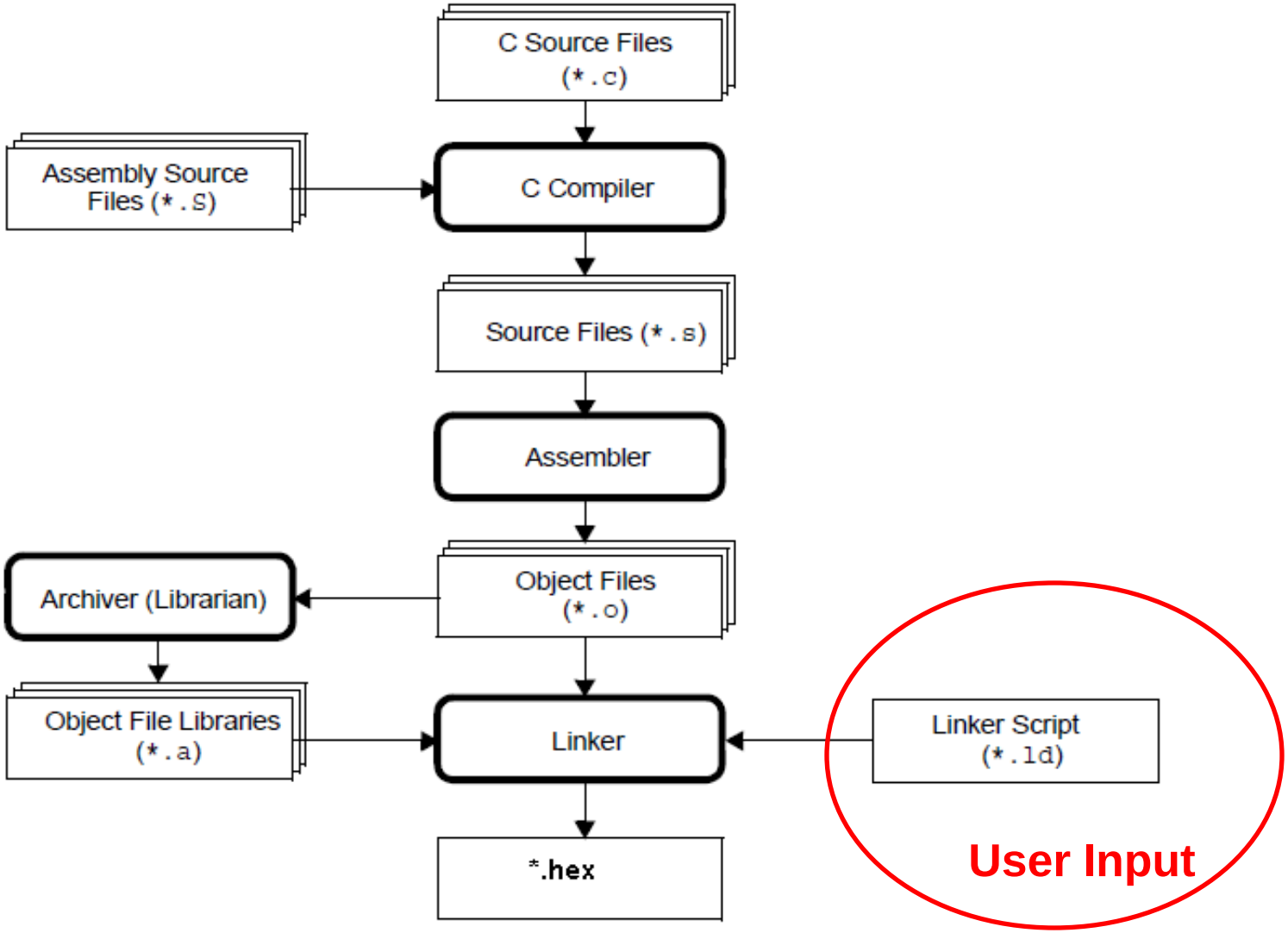
Agenda

- Linker Script Basics
- Contents of Linker Script Files
- Lab 1
- Application/Bootloader Mapping Steps
- Lab 2
- Merging the Hex Files

Why talk about the Linker?



Tool Flow



Linker Script

Tells the linker where to place the following:

- **Reset Address of an image**
- **C-Startup code**
- **Interrupt Vector Table (IVT)**
- **Text and Data sections**

Linker Script

Additionally informs the location and size of:

- **RAM**
- **SFRs**
- **Device Configuration Registers**

PIC32 Linker Script Arrangement

PIC32 Linker Script Files

```
/* Default linker script, for normal executables */
OUTPUT_FORMAT("elf32-tradlittlemips")
OUTPUT_ARCH(pic32mx)
ENTRY(_reset)
/*
 * Provide for a minimum stack and heap size
 * - _min_stack_size - represents the minimum space that must be made
 *                     available for the stack.  Can be overridden from
 *                     the command line using the linker's --defsym option.
 * - _min_heap_size  - represents the minimum space that must be made
 *                     available fo
 *                     the command
 */
EXTERN (_min_stack_size _min_heap_s
PROVIDE(_min_stack_size = 0x400) ;
PROVIDE(_min_heap_size = 0) ;
INCLUDE procdefs.ld
```

elf32pic32mx.x (Common to all part numbers)

```
/*****************************************************************************
 * Processor-specific object file.  Contains SFR definitions.
 *****************************************************************************/
INPUT("processor.o")

/*****************************************************************************
 * For interrupt vector handling
 *****************************************************************************/
PROVIDE(_vector_spacing = 0x00000001);
_ebase_address  = 0x9FC01000;

/*****************************************************************************
 * Memory Address Equates
 *****************************************************************************/
_RESET_ADDR                 = 0xBFC00000;
_BEV_EXCPT_ADDR             = 0xBFC00380;
_DBG_EXCPT_ADDR             = 0xBFC00480;
_DBG_CODE_ADDR              = 0xBFC02000;
_GEN_EXCPT_ADDR             = _ebase_address + 0x180;

/*****************************************************************************
 * Memory Regions
 *
 * Memory regions without attributes cannot be used for orphaned sections
```

procdefs.ld (Part number specific)

PIC32 Linker Script Files

procdefs.ld

- Device specific header file
- Defines Memory Regions

elf32pic32mx.x

- Default linker file common to all devices
- Maps code sections to memory regions

Focus on Procdefs.ld

- procdefs.ld has to be modified for image mapping
- There is nothing to modify in elf32pic32mx.x

Contents of Procdefs.ld

Vector Spacing and Exception Base Address

```
/*************************************
For interrupt vector handling
*************************************/
_vector_spacing= 0x00000001;
_ebase_address = 0x9FC01000;
```

Memory Address Equates

```
/*************************************
Memory Address Equates
*************************************/
_RESET_ADDR= 0xBFC00000;
_BEV_EXCPT_ADDR= 0xBFC00380;
_DBG_EXCPT_ADDR= 0xBFC00480;
_DBG_CODE_ADDR= 0xBFC02000;
_GEN_EXCPT_ADDR=_ebase_address+0x180;
```

Memory Regions

```
MEMORY
{
    kseg0_program_mem (rx) :ORIGIN = 0x9D000000, LENGTH = 0x8000
    kseg0_boot_mem         : ORIGIN = 0x9FC00490, LENGTH = 0x970
    exception_mem          : ORIGIN = 0x9FC01000, LENGTH = 0x1000
    kseg1_boot_mem         : ORIGIN = 0xBFC00000, LENGTH = 0x490
    debug_exec_mem         : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
    config3                : ORIGIN = 0xBFC02FF0, LENGTH = 0x4
    config2                : ORIGIN = 0xBFC02FF4, LENGTH = 0x4
    config1                : ORIGIN = 0xBFC02FF8, LENGTH = 0x4
    config0                : ORIGIN = 0xBFC02FFC, LENGTH = 0x4
    kseg1_data_mem (w!x)   : ORIGIN = 0xA0000000, LENGTH = 0x2000
    sfrs                   : ORIGIN = 0xBF800000, LENGTH = 0x10000
}
```

Memory Region - Alignment

		Kseg1 address	Kseg0 address	Physical address
Boot Flash	Config 0, Config 1, Config2, Config3 (Device Config Reg)	0xBFC02FFF 0xBFC02FF0		0x1FC02FFF 0x1FC02FF0
	debug_exec_mem (Debugger Code)	0xBFC02FEF 0xBFC02000		0x1FC02FEF 0x1FC02000
	exception_mem (IVT)		0x9FC01FFF 0x9FC01000	0x1FC01FFF 0x1FC01000
	kseg0_boot_mem (Not Used)		0x9FC00E00 0x9FC00490	0x1FC00E00 0x1FC00490
Reset → Vector	kseg1_boot_mem (C Startup Code)	0xBFC0048F 0xBFC00000		0x1FC0048F 0x1FC00000
	Reserved			
	Sfrs (Peripheral Registers)	0xBF8FFFFF 0xBF800000		0x1F8FFFFF 0x1F800000
	Reserved			
Program Flash	kseg0_program_mem (All C Files)		0x9D07FFFF 0x9D000000	0x1D07FFFF 0x1D000000
	Reserved			
	kseg1_data_mem (RAM)	0xA001FFFF 0xA0000000		0x0001FFFF 0x00000000

Contents of elf32pic32mx.x

elf32pic32mx.x in Brief

- Maps standard code sections to memory regions defined in Procdefs.ld
- For example:
  - *.startup* -> kseg1_boot_mem
  - *.bss* -> kseg1_data_mem
  - *.text* -> kseg1_program_mem
  - *.heap* -> kseg1_data_mem
  - *.stack* -> kseg1_data_mem

Lab 1

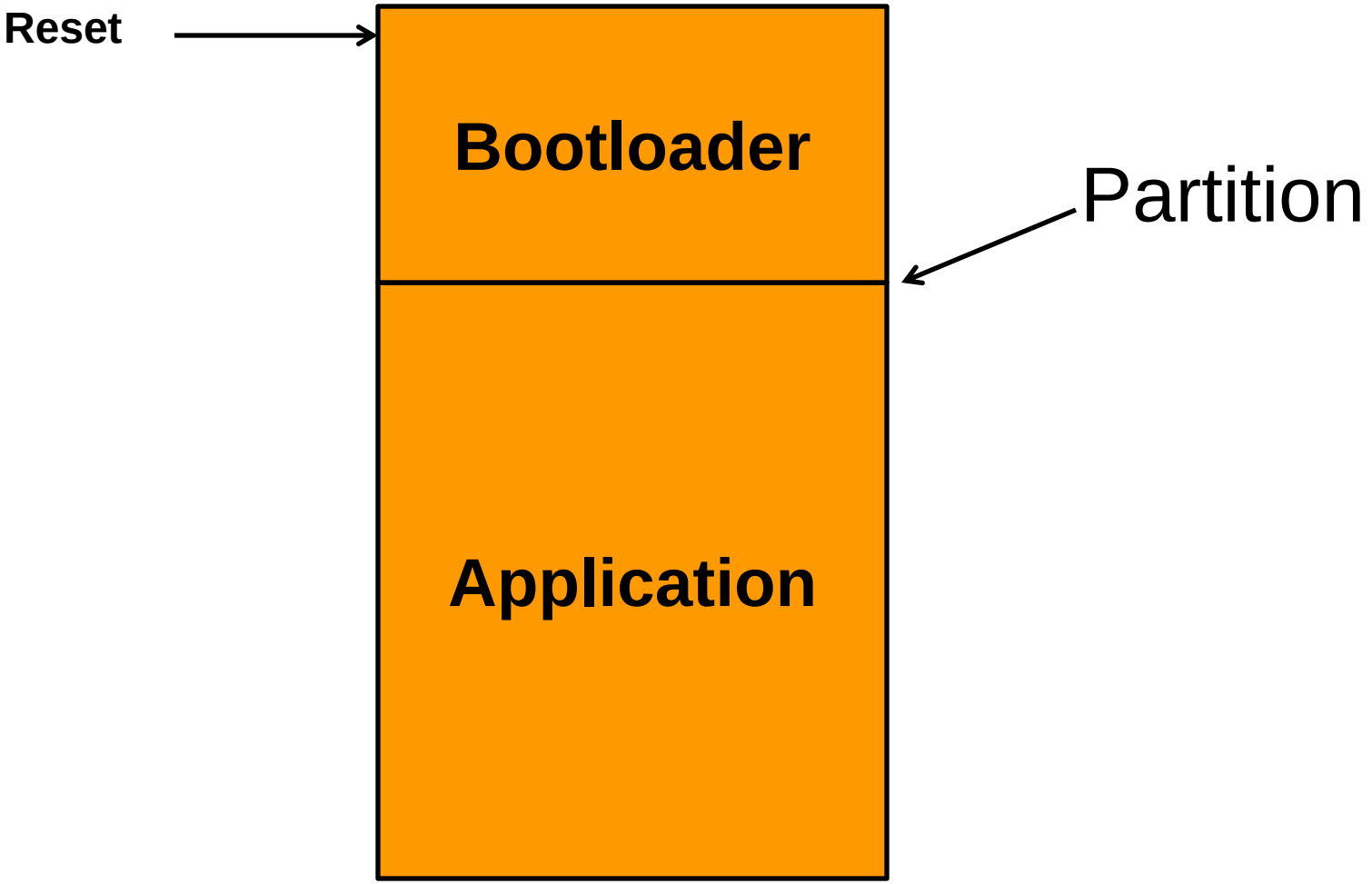
Lab 1

Objective

- Locate procdefs.ld file
- Understand the contents of procdefs.ld file
- Locate the template of main linker script file
- Take a brief look into the main linker script file

Application/Bootloader Mapping

Our Objective



How to achieve?

Remap the following in procdefs.ld

kseg0_program_mem	Text and Data Sections
kseg1_boot_mem	C- Startup Code
exception_mem	Interrupt Vector Table
debug_exec_mem	Debugger Executive Code
config0, config1, config2 and config3	Device Configuration Registers
kseg1_data_mem	RAM Location
sfrs	Special Function Registers

Step by Step Approach

Step 1- Select procdefs.ld

- Choose correct procdefs.ld for the selected part number from

  “C:\Program Files\Microchip\MPLAB C32\pic32mx\lib\proc”

- Copy procdefs.ld into the project folder of bootloader and application project

Step 2 - Copy main linker script file

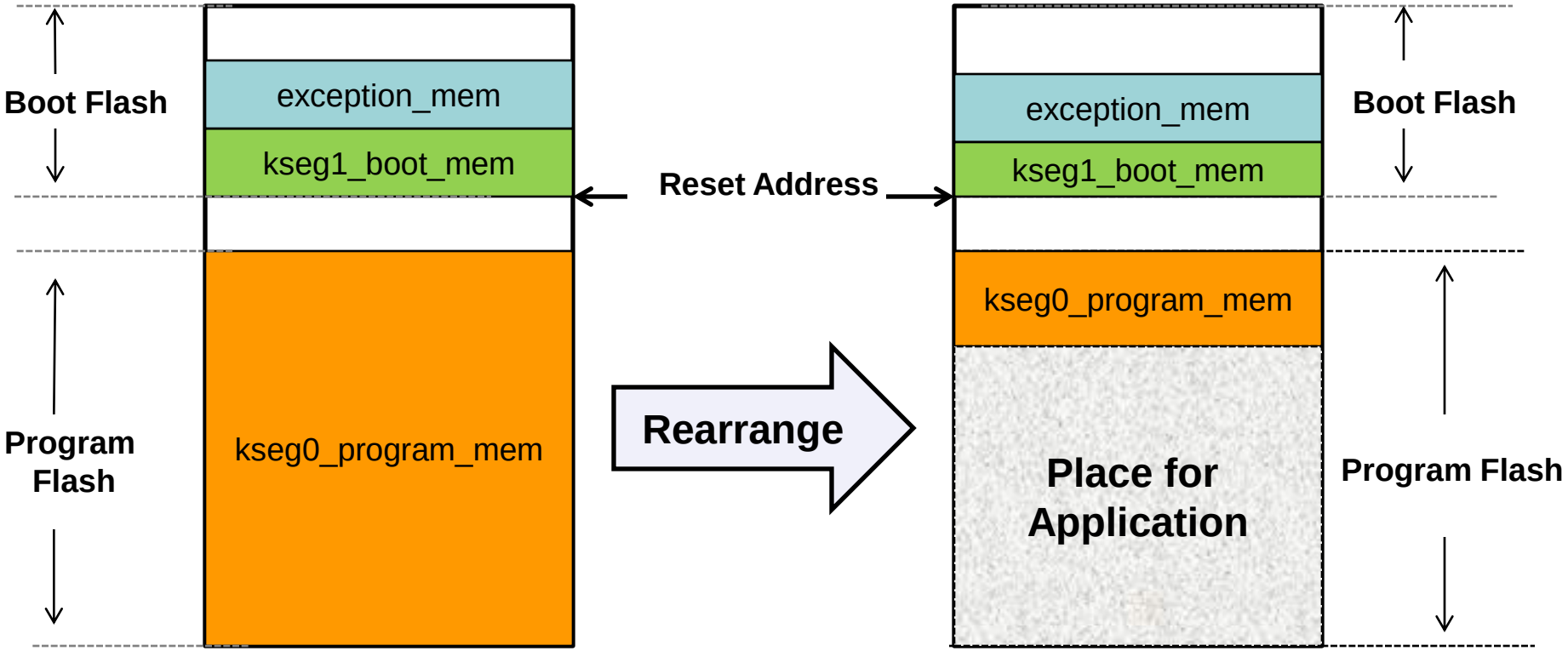
- Copy “elf32pic32mx.x” into the project folder of bootloader and application projects from

  “C:\Program Files\Microchip\MPLAB C32\pic32mx\lib\ldscripts”

- Rename the file to **“elf32pic32mx.ld”**

Step 3 - Bootloader Mapping

Edit Bootloader's *procdefs.ld* to rearrange the Memory Regions

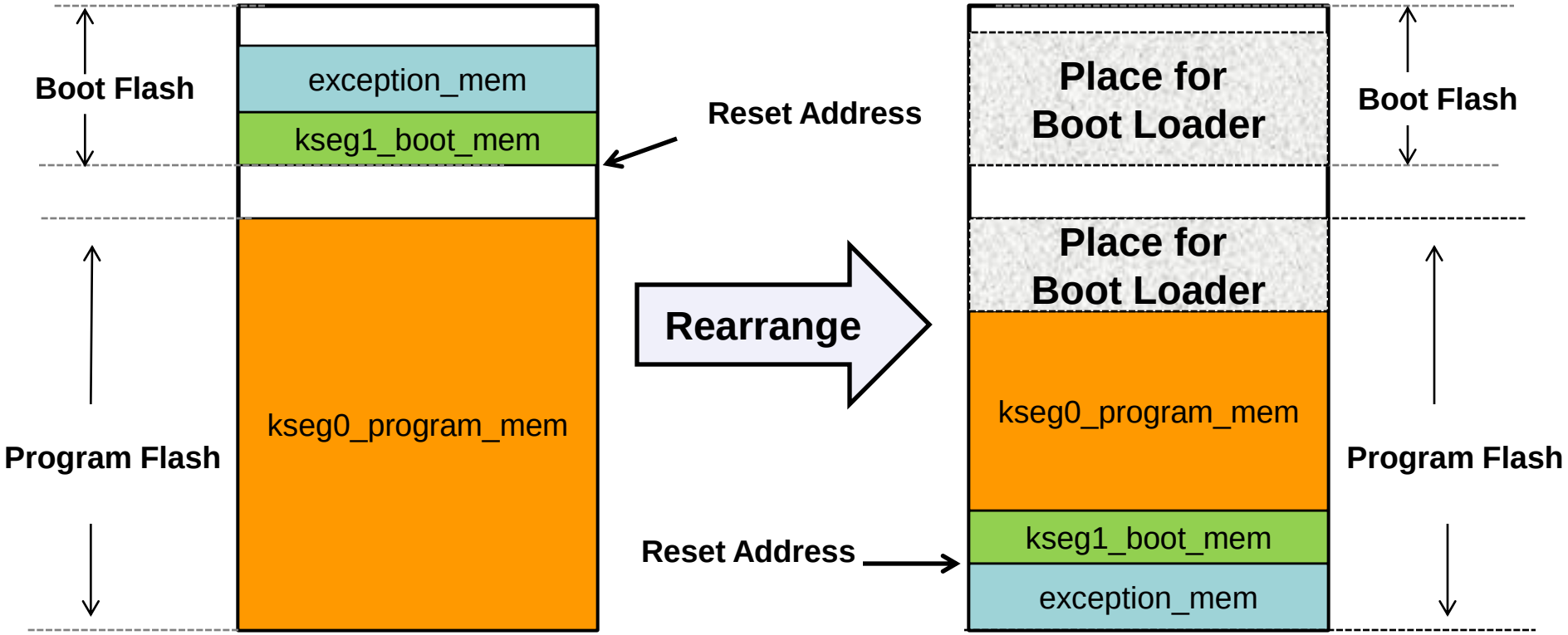


Mapping as per Default Linker Script

Mapping changed to fit bootloader in Boot Flash

Step 4 - Application Mapping

Edit Application's *procdefs.ld* to rearrange the Memory Regions



Mapping as per Default Linker Script

Mapping changed to fit application in Program Flash

Important!

- Align *exception_mem* on 4KB Flash Page
- Recommend keeping default length for:
  - exception_mem (IVT)
  - kseg1_boot_mem

Step 5 - Change Memory Address Equates

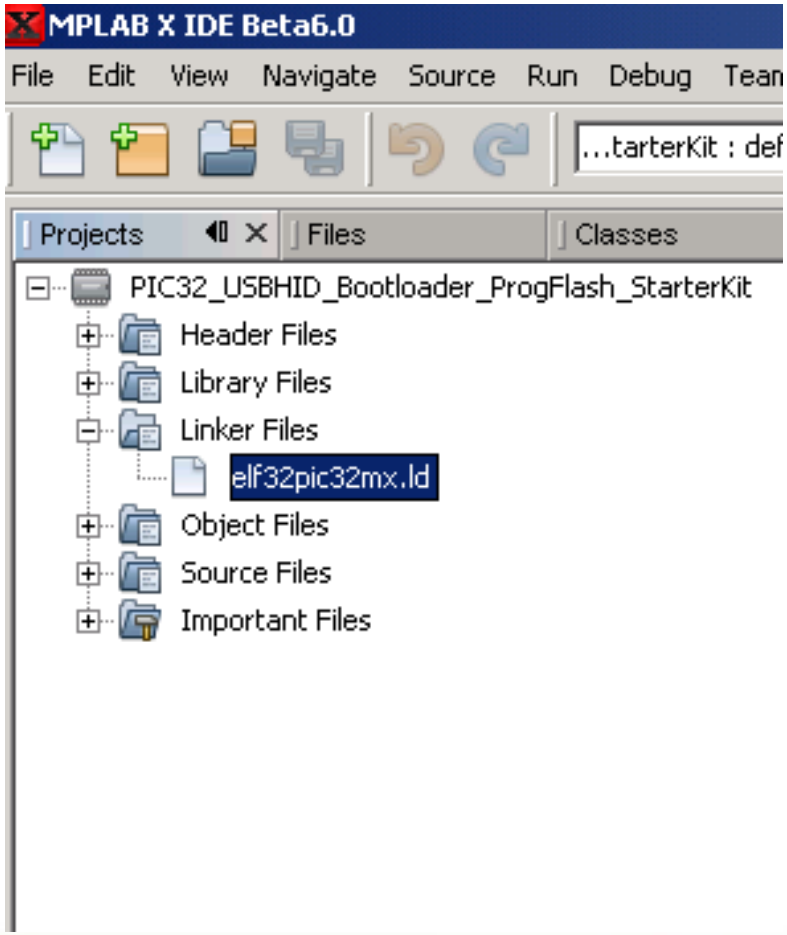
Set Address

- _ebase_address = exception_mem origin
- _RESET_ADDR = kseg1_boot_mem origin
- _BEV_EXCPT_ADDR

  = kseg1_boot_mem origin + 0x380

Step 6 - Linking

- Add “elf32pic32mx.ld” to the project
- Rebuild the project



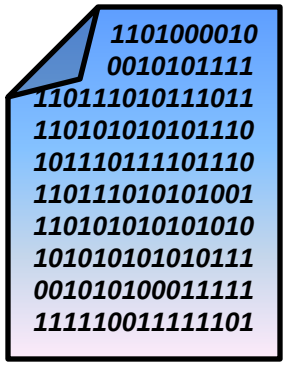
Lab 2
Application and Bootloader Mapping

Lab 2

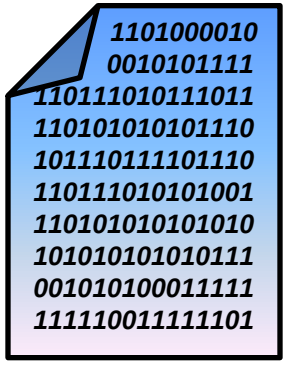
Objective

- Edit procdefs.ld of the bootloader project to remap the bootloader
- Compile and download the bootloader into the target device using the programmer
- Edit procdefs.ld of the application project to remap the application
- Compile the application and download the application into the target device using the bootloader

Merging the Hex Files



Bootloader.hex



Application.hex

Merging the Hex Files

- Merge Application image and Bootloader image for production programming
- Remove the overlapped sections

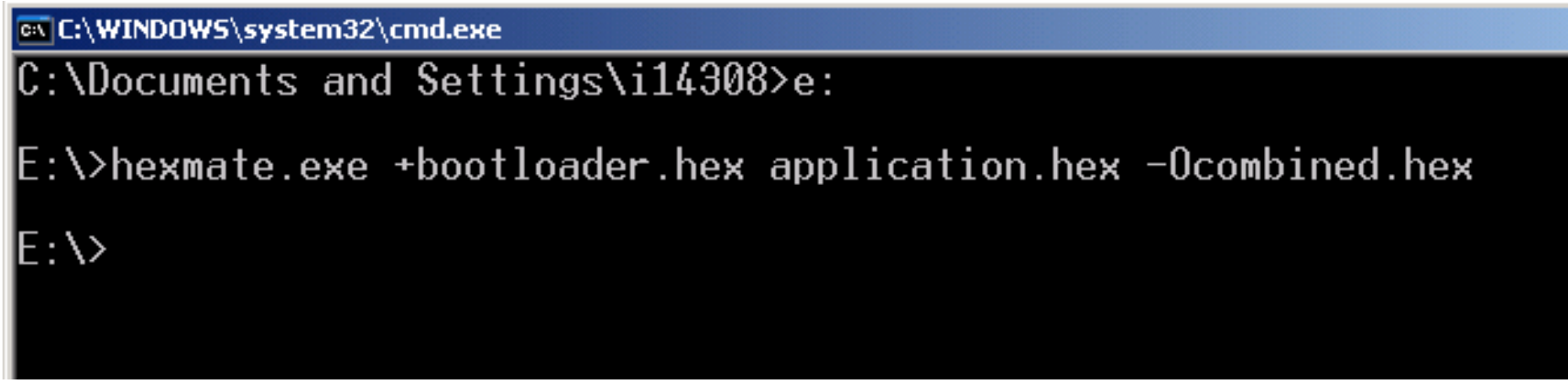
HexMate

Features

- Stand alone command line tool from HI-TECH C® Compiler
- Merges 2 hex files to produce a single hex file
- Removes all overlapping sections automatically

HexMate

- **Usage**



  - The “+” option retains overlapping sections from bootloader

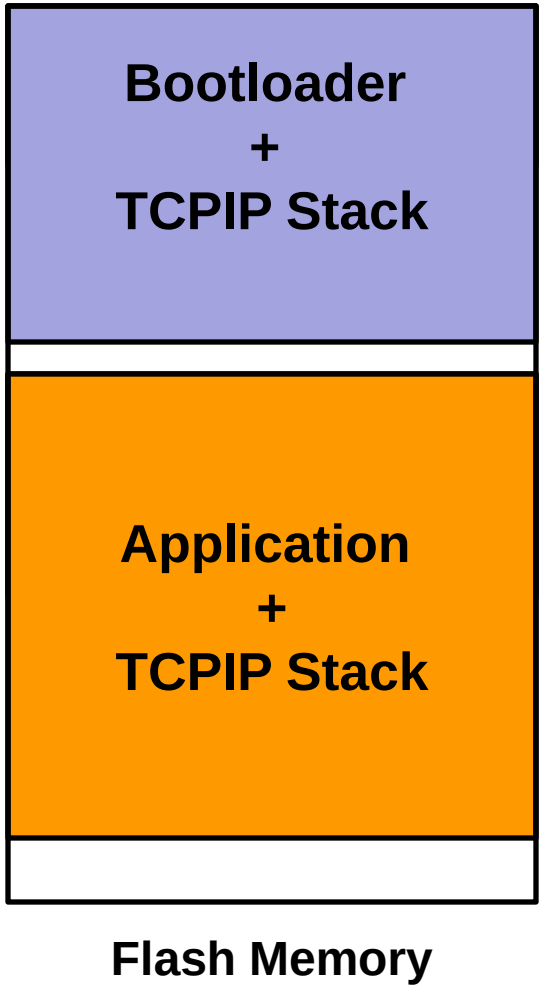
Run-Time Library Loading (RTLL) Technique

Run-Time Library Loading

Agenda

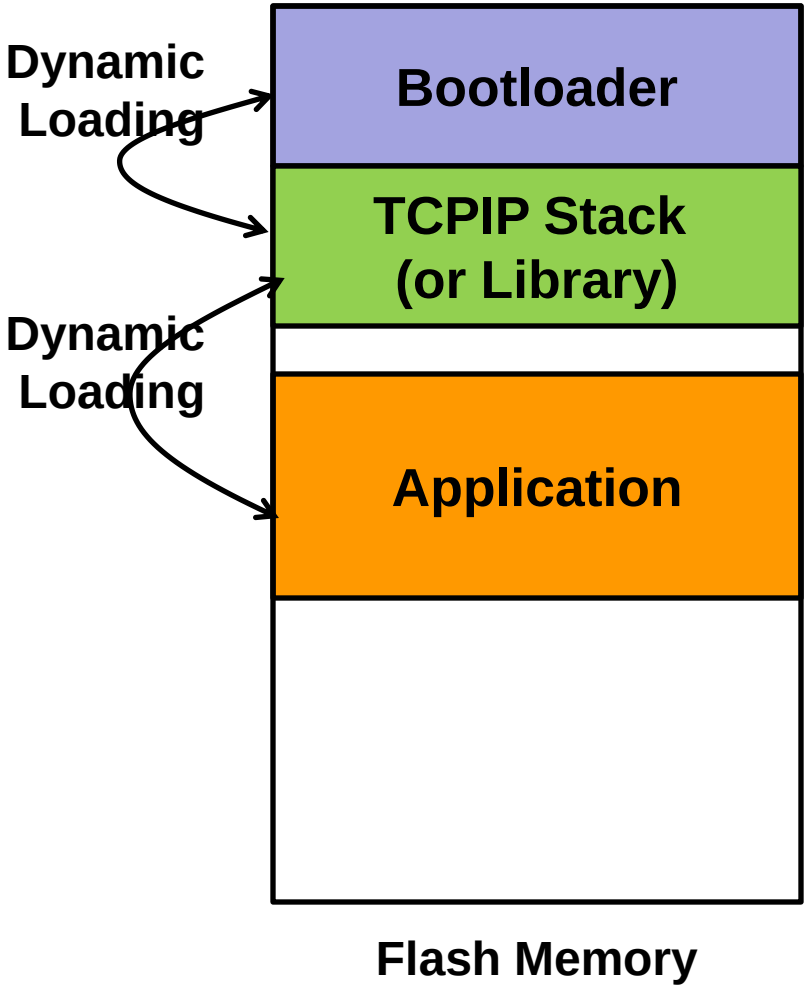
- Usage
- Concept
- Lab-3

Static Linking



- Two copies of the Stack
- Unnecessary waste of memory

RTLL - Dynamic Loading



- Single copy of the Stack
- Effective use of Flash memory

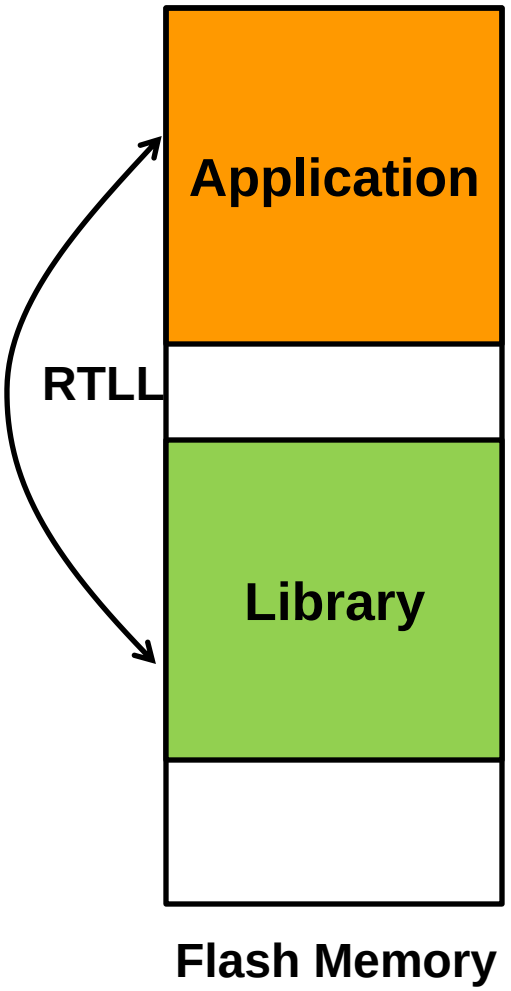
Other Usage

- Dynamically linking a Library subjected to the Open Source End User License Agreement

Concept

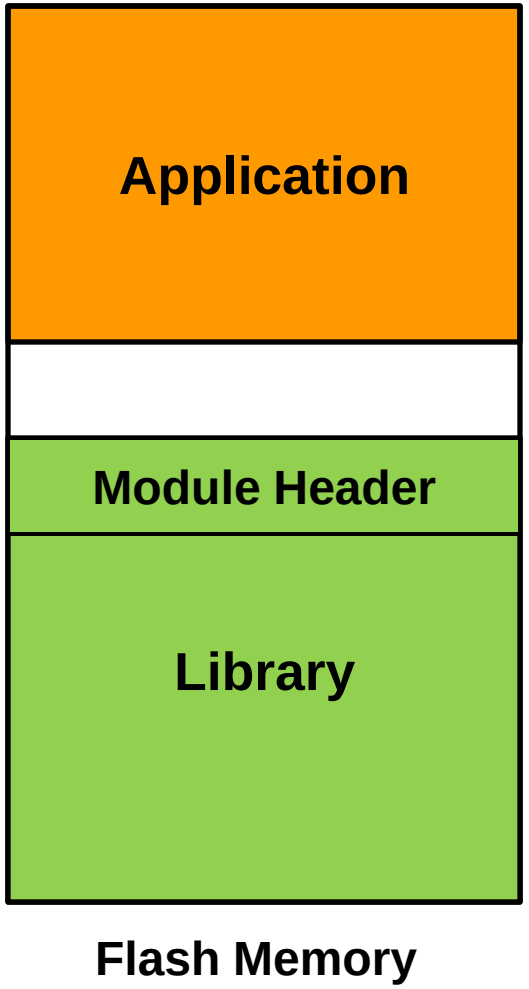
RTLL Objective

- Run-Time Resolution of Library API addresses

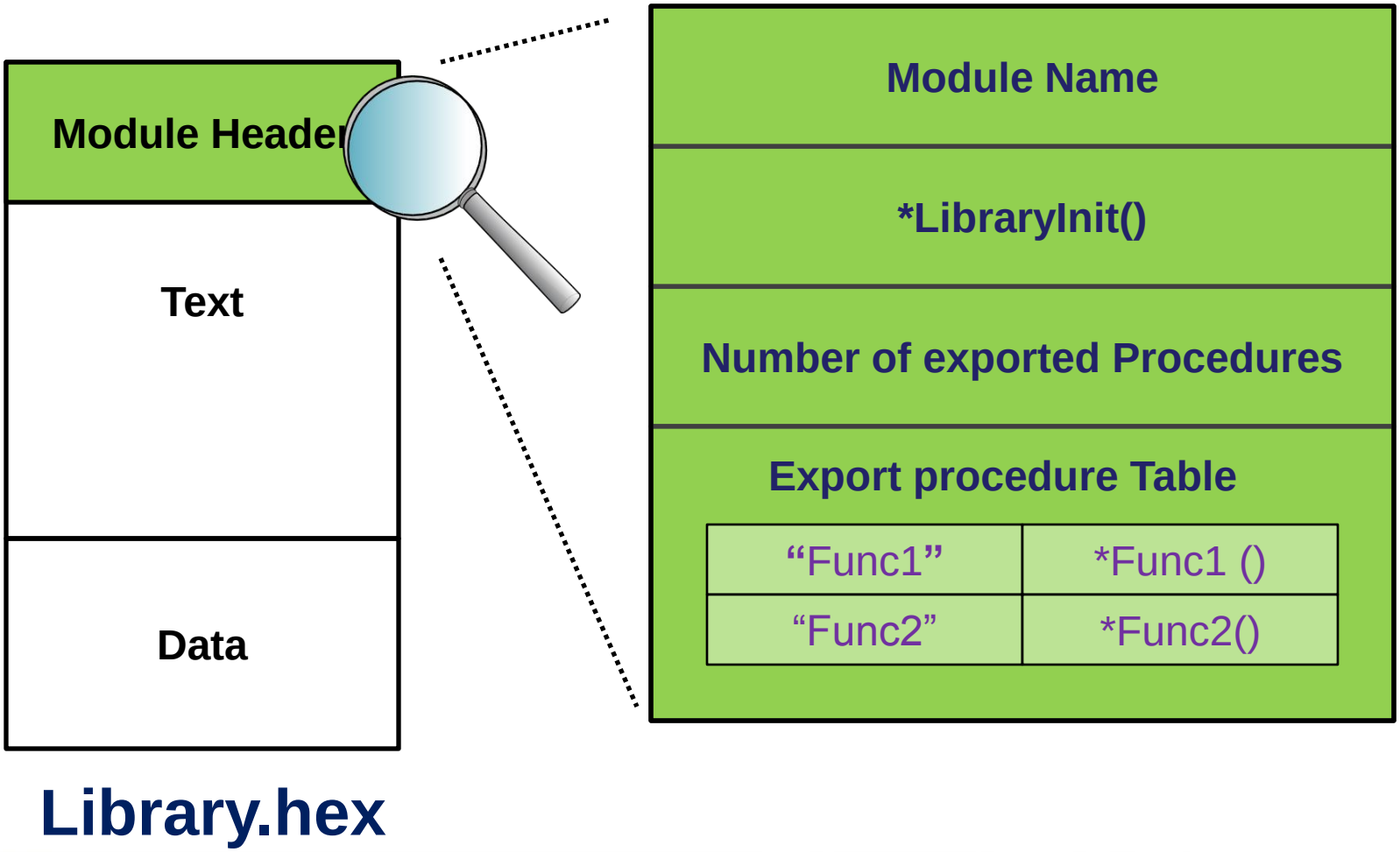


How is it Achieved?

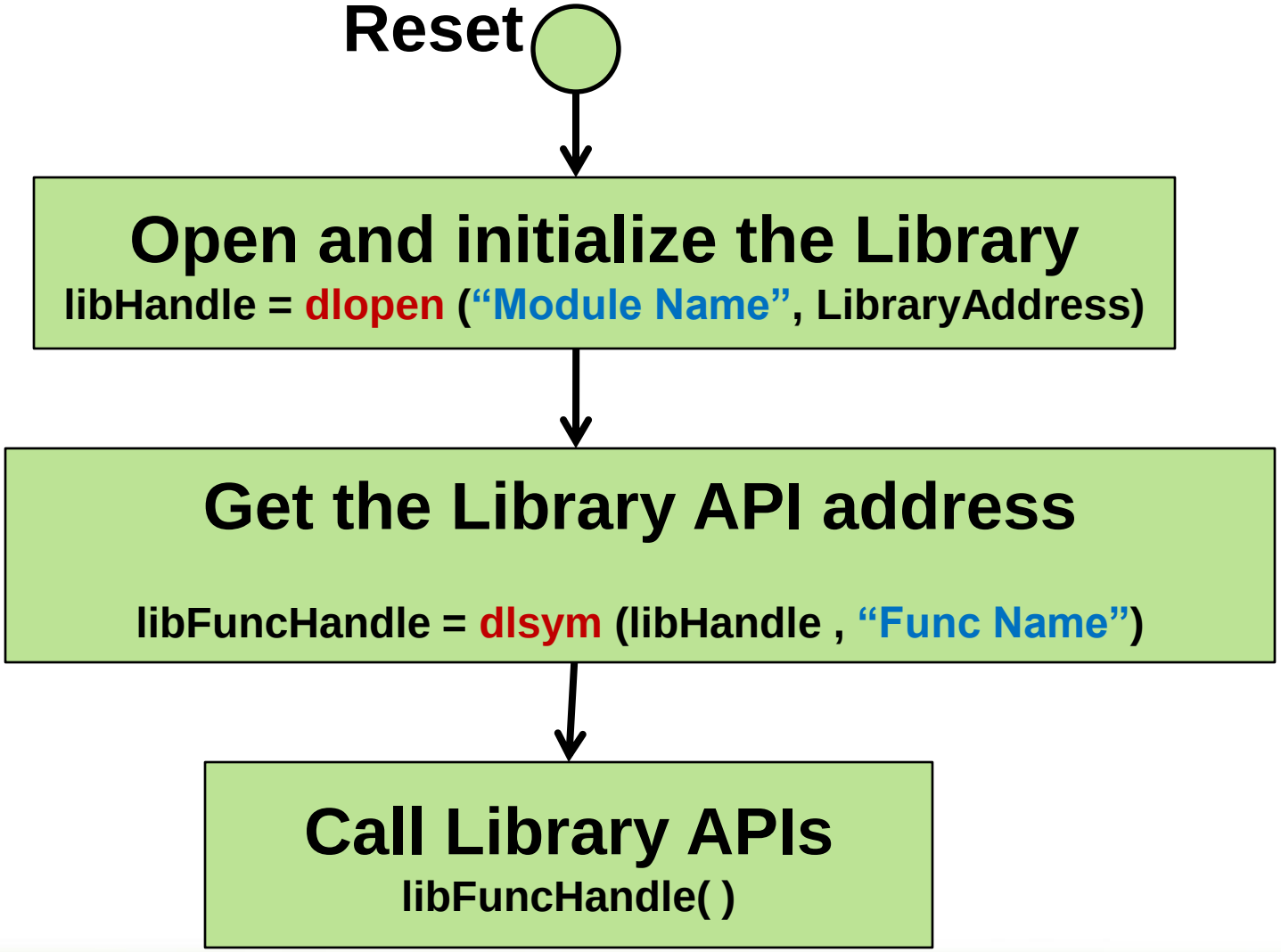
- Library exposes its API addresses using a *standard structure*
- Location of *standard structure* is known to application
- Application *digs out* the library API addresses at run time from *standard structure*



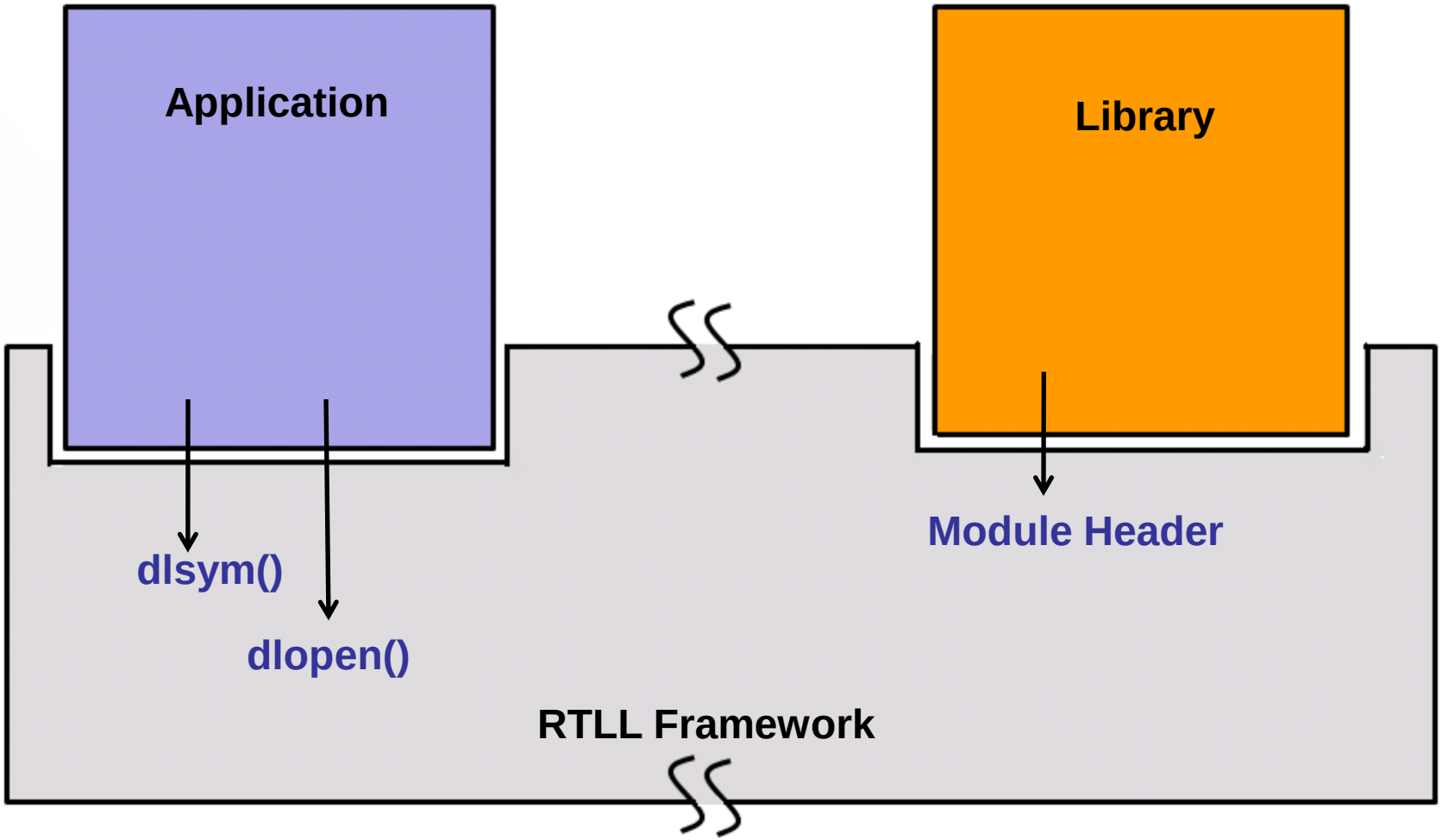
Library Requirement



Application Flow



RTLL - Framework



Application <-> Library Context Saving

No context saving mechanism to keep the design simple

- Library uses application's stack
- Library uses application's heap
- Only application uses GP register. GP relative addressing is disabled in the library (disabled by compiler option "-G0")

Placing Module Header in Absolute Address

//Defining Module Header Structure in the Library

```
// Standard header which must be exposed by the library.
const T_MODULE_DYN_HDR __attribute__((address(0x9D06E000))) _ModuleLoadHdr =
{

    _MODULE_NAME_,          // name of the Module
    _libInit,               // Start up code
    _nProcs,                // Number of procedures
    _exportProcTbl          // Export Procedure Table
};
```

Small Things, but Important

- No ISRs in the library
- No “device configuration register” settings in the library
- No C-Startup code in the library. (No main())
- Use linker option *“--no-gc-sections”* to disable garbage collection

RTLL – Lab 3

RTLL Lab 3

Objective

- Export the APIs from a simple Math Library, compile and build it
- Build an application which calls the Library APIs
- Program the application and library separately into the PIC32 device
- Debug the application to see how the application calls these Library APIs using RTLL technique

PIC32 Bootloader Solutions from Microchip

AN1388 Application Note

Solutions available so far…

- UART Bootloader
- Ethernet Bootloader
- USB HID Bootloader
- USB Mass Storage Bootloader
- SD Card Bootloader

AN1388 - UART Bootloader

- Download image from PC
- Bootloader mapping
  - Fits well inside 12KB Boot Flash
- User application can occupy entire program Flash
- Hardware required
  - Explorer 16 Board + PIC32 PIM
- Default baud rate – 115200 (Changeable by setting DEFAULT_BAUDRATE)

AN1388 - USB HID Bootloader

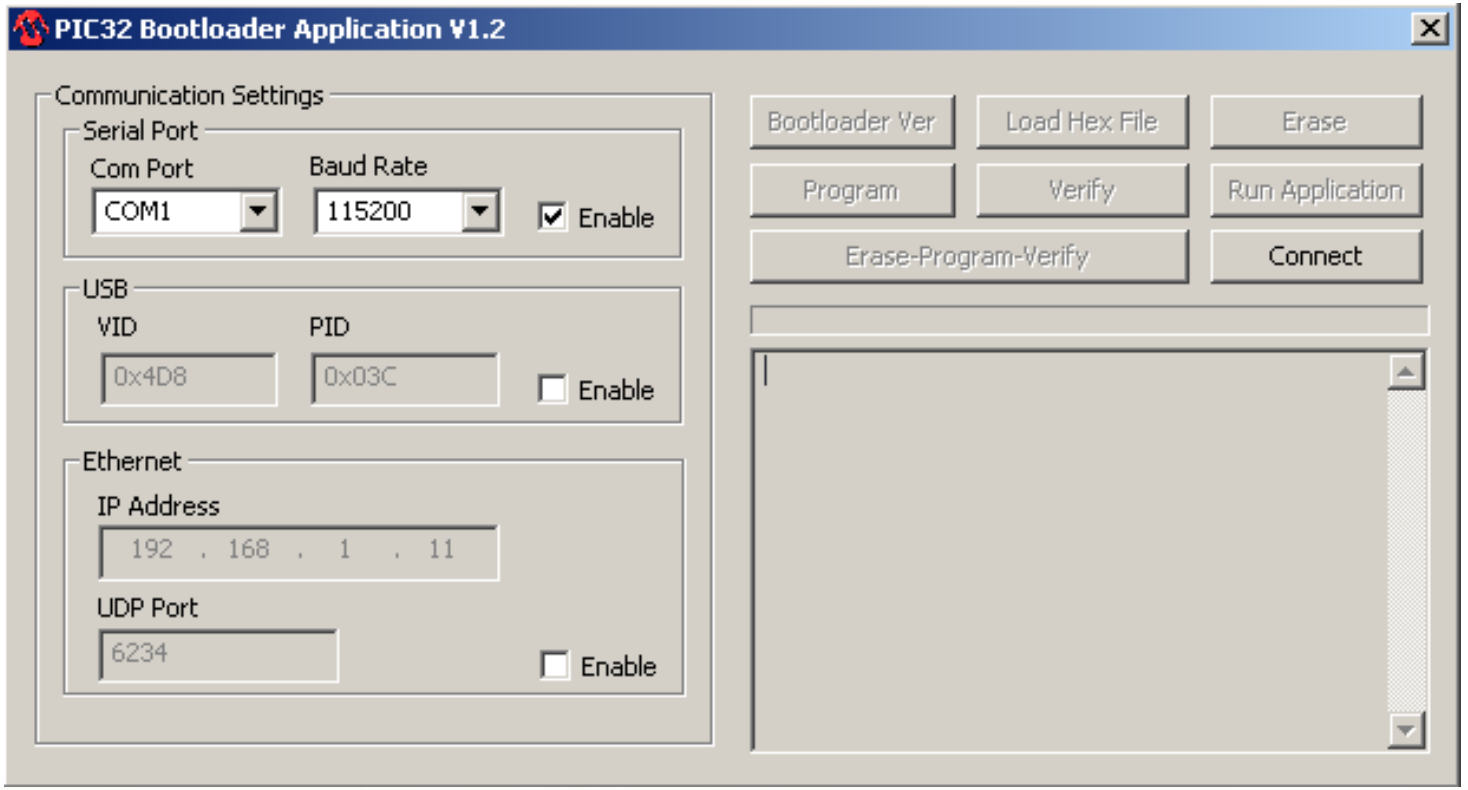
- Download image from PC
- Bootloader mapping
  - C-Startup code and IVT mapped in Boot Flash
  - Rest of the bootloader in program Flash (around 12KB)
- Hardware required
  - PIC32 USB Starter Kit OR PIC32 Ethernet Starter Kit
- Uses Microchip's USB device stack
- VID and PID modifiable using USB_VENDOR_ID and USB_PRODUCT_ID

AN1388 - Ethernet Bootloader

- Download image from PC
- Bootloader mapping
  - C-Startup code and IVT mapped in Boot Flash
  - Rest of the bootloader in program Flash (around 20KB)
- Hardware required
  - PIC32 Ethernet Starter Kit OR
  - Explorer 16 Board + PIM + Ethernet PICtail™ Plus Board
- Uses Microchip's TCP/IP stack
- Device static IP address, UDP port and MAC address are configurable

AN1388 PC Application

- One PC application for UART/USB HID and Ethernet Bootloader



AN1388 - SD Card Bootloader

- Programs the image from SD card
- Bootloader mapping
  - C-Startup code and IVT mapped in Boot Flash
  - Rest of the bootloader in program Flash (around 12KB)
- Hardware required
  - Explorer 16 Board + PIM + PICtail™ Daughter Board for SD card
- Uses Microchip's File system

AN1388 - USB Thumb Drive Bootloader

- Programs the image from USB Thumb Drive
- Bootloader mapping
  - C-Startup code and IVT mapped in Boot Flash
  - Rest of the bootloader in program Flash (around 24KB)
- Hardware required
  - PIC32 USB Starter Kit OR PIC32 Ethernet Starter Kit
- Uses Microchip's USB Host Stack

AN1388 - Bootloader General Settings

- **APP_FLASH_BASE_ADDRESS**
  - Tells the bootloader the start address of the Flash reserved for the application
- **APP_FLASH_END_ADDRESS**
  - Tells the bootloader the end address of the Flash reserved for the application
- **USER_APP_RESET_ADDRESS**
  - Bootloader loads program counter with this address to begin running the application

Summary

- Bootloader Basics
- Designing the Bootloader on the PIC32
- Application Mapping
- Run-Time Library Loading Technique

Additional Resources

- **Application Notes**
  - **AN1388** (PIC32 Bootloader)
  - **AN1367** (Porting the Helix MP3 Decoder onto Microchip's PIC32MX 32-bit MCUs)
- **Data Sheets**
  - **DS61143** (PIC32MX3XX/4XX Data Sheet)
  - **DS61156** (PIC32MX5XX/6XX/7XX Family Data Sheet)
  - **DS61168** (PIC32MX1XX/2XX Family Data Sheet)

Additional Resources

- **Other Documents**
  - **DS51833** (MPLAB® Assembler, Linker and Utilities for PIC32 MCUs User's Guide)
  - **DS51686** (MPLAB XC32 C Compiler User's Guide)

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.