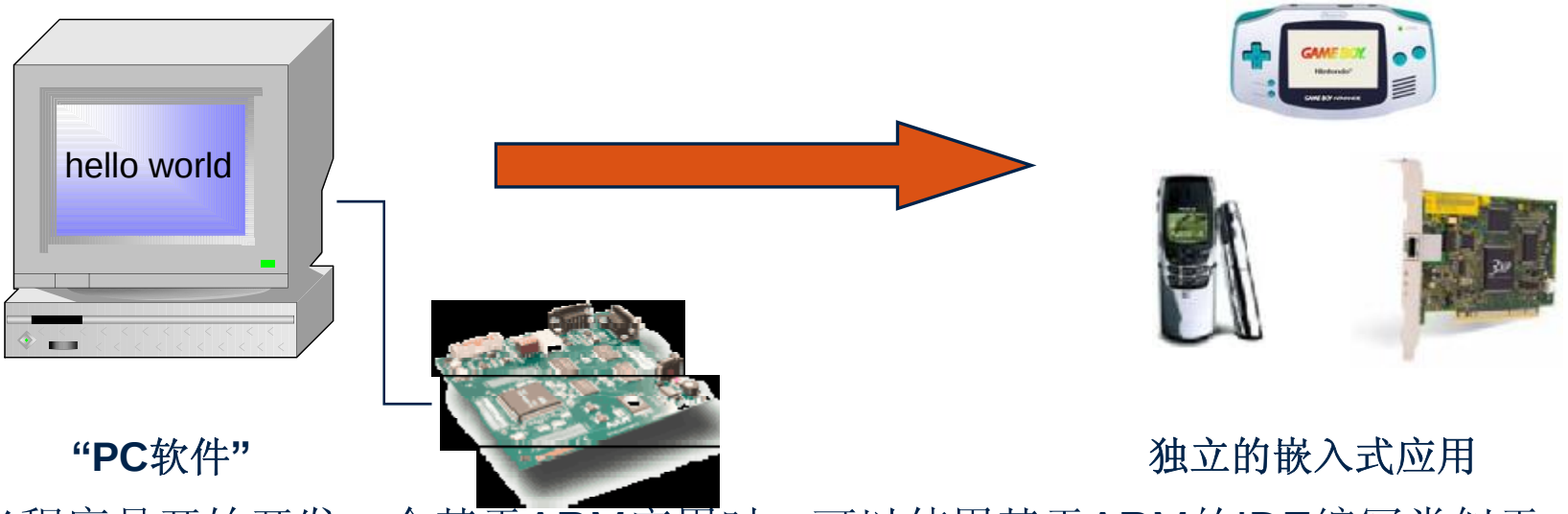
嵌入式ARM软件开发

启动代码设计

嵌入式开发过程



“PC软件”

独立的嵌入式应用

当程序员开始开发一个基于ARM应用时，可以使用基于ARM的IDE编写类似于“HELLO WORLD”的程序，使用Simulator或下载到评估板上调试，基于独立的嵌入式应用设备的程序开发，下面这些问题就成为我们首要考虑的：

- 硬件系统的初始化-启动代码
- 实际系统的软硬件协同开发
  - 集成开发环境
  - 高级C函数的使用
  - 目标板存储器资源
- 应用程序的初始化

讨论内容

- **嵌入式ARM体系结构概述**

  常见ARM处理器启动过程

  ARM处理器复位和初始化

  嵌入式ARM软件程序设计

  典型ARM启动代码分析

  嵌入式软件的编译和调试

ARM Architecture

4

Halfword and signed halfword / byte support

System mode

4T

Thumb instruction set

5TE

Improved ARM/Thumb Interworking

CLZ

Saturated arithmetic

DSP multiply-accumulate instructions

5TEJ

Jazelle

Java bytecode execution

6

SIMD Instructions

Multi-processing

V6 Memory architecture (VMSA)

Unaligned data support

6T2

Thumb-2 instruction set

6Z

TrustZone

AOS extensions(v6k)

7A

Applications profile

7R

Real-Time profile

7M

Microcontroller profile

数据和指令类型

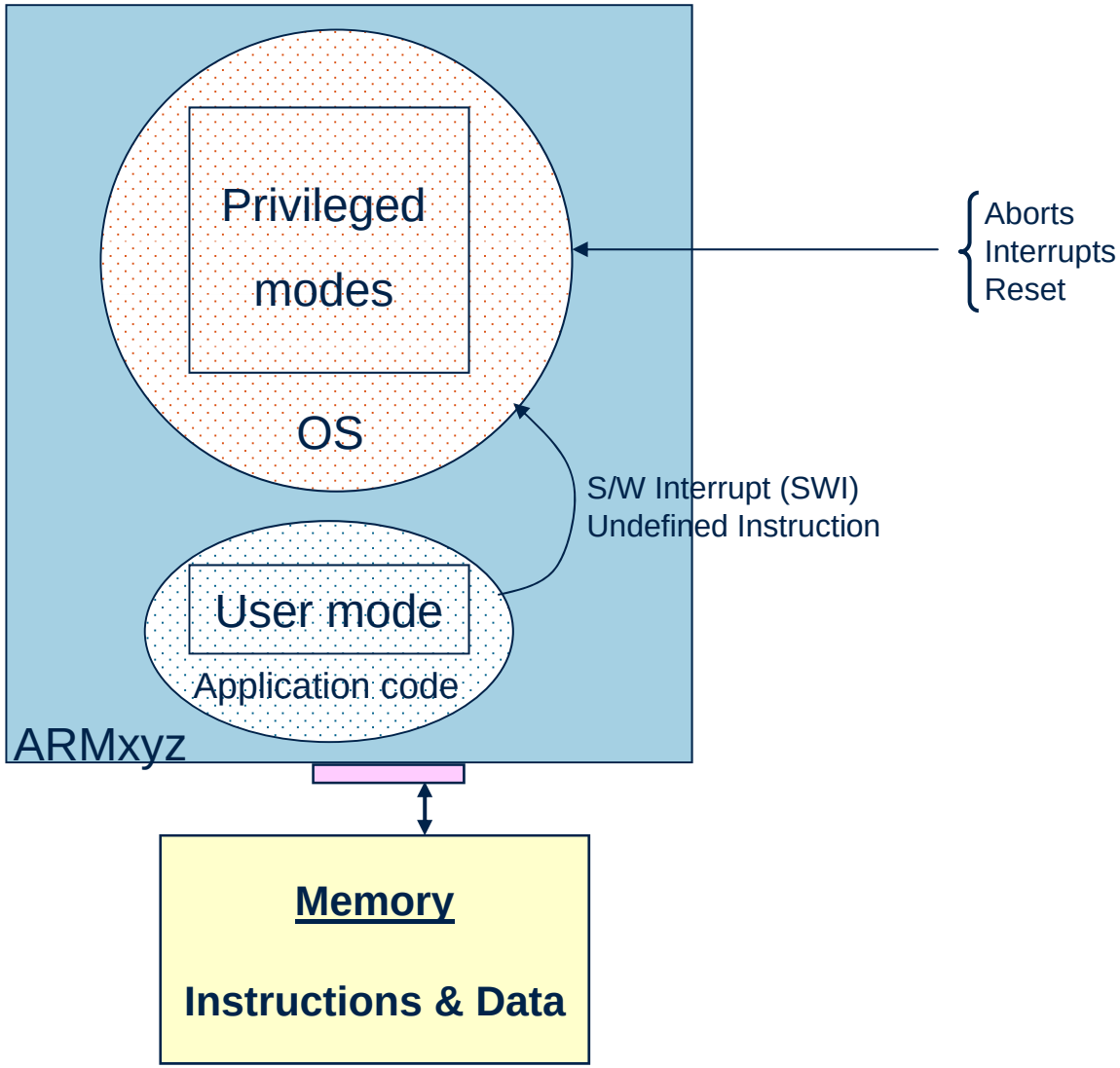
- **ARM 采用的是32位架构.**

- **ARM 约定:**
  - **Byte** 8 bits
  - **Halfword** 16 bits (2 bytes)
  - **Word** 32 bits (4 bytes)

- 大部分**ARM core 提供:**
  - ARM 指令集（32-bit）
  - Thumb 指令集（16-bit ）

- **Jazelle cores 支持 Java byte代码**

处理器工作模式


Privileged
modes
OS
Aborts
Interrupts
Reset
S/W Interrupt (SWI)
Undefined Instruction
User mode
Application code
ARMxyz
Memory
Instructions & Data

处理器工作模式

- **ARM 有7个基本工作模式:**
  - **User** : 非特权模式，大部分任务执行在这种模式 **Un-privileged**
  - **FIQ** : 当一个高优先级（fast) 中断产生时将会进入这种模式 **Privileged**
  - **IRQ** : 当一个低优先级（normal) 中断产生时将会进入这种模式
  - **Supervisor** : 当复位或软中断指令执行时将会进入这种模式
  - **Abort** : 当存取数据异常时将会进入这种模式(memory access violations)
  - **Undef** : 当执行未定义指令时会进入这种模式
  - **System** : 使用和User模式相同寄存器集的特权模式

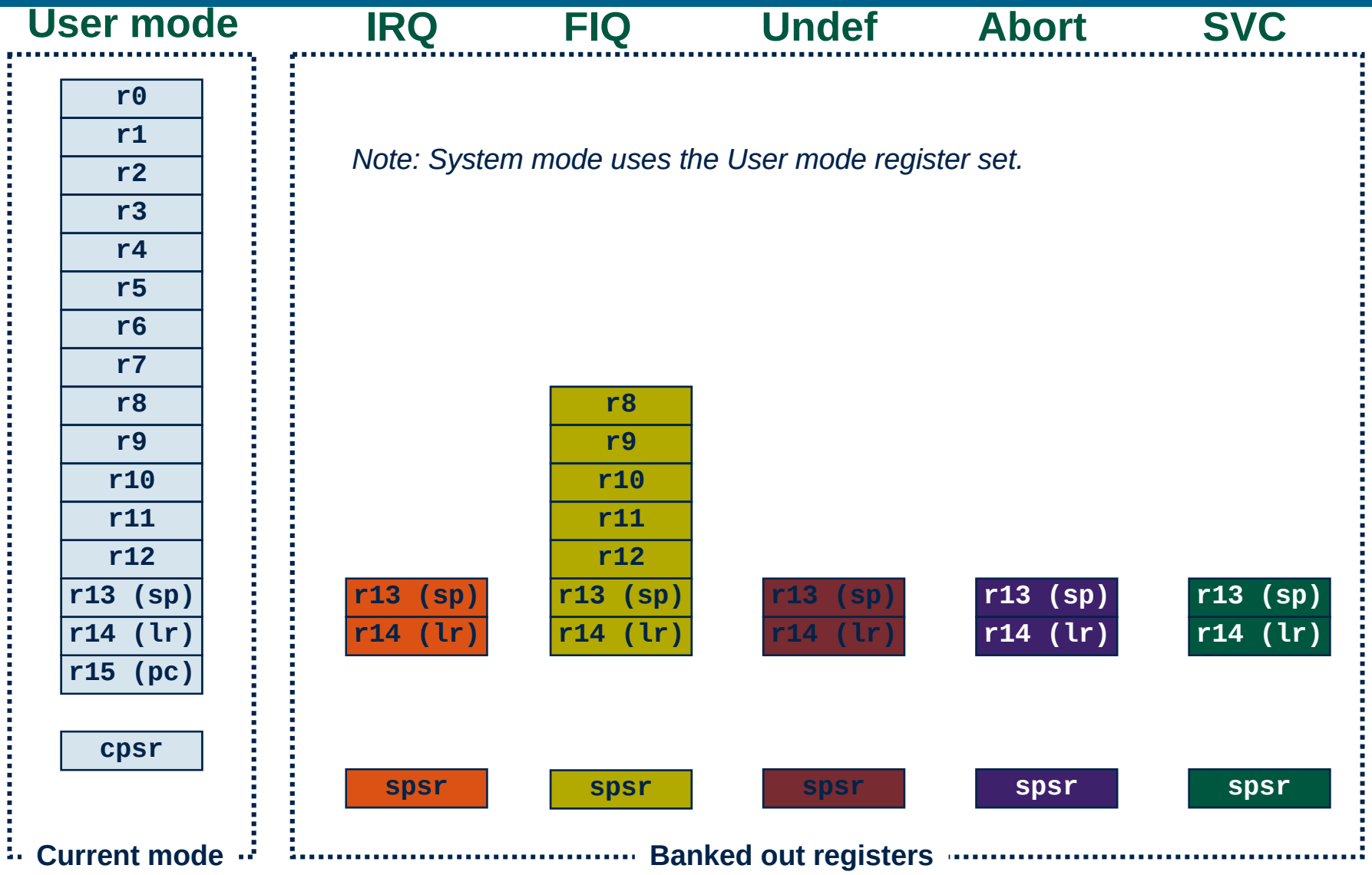
ARM 寄存器

- **ARM 有37个32-Bits长的寄存器.**
  - 1 个用作PC (program counter)
  - 1个用作CPSR (current program status register)
  - 5个用作SPSR (saved program status registers)
  - 30 个通用寄存器

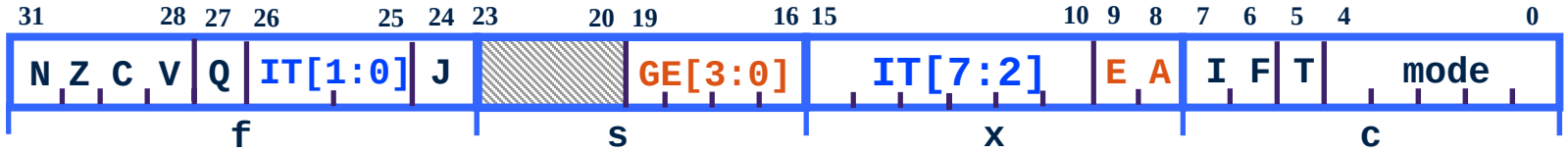
- 当前处理器的模式决定着哪组寄存器可操作. 任何模式都可以存取：
  - 相应的r0-r12子集
  - 相应的 r13 (the stack pointer, sp) and r14 (the link register, lr)
  - 相应的 r15 (the program counter, pc)
  - 相应的CPSR (current program status register, cpsr)

- 特权模式 **(除system模式)** 还可以存取；
  - 相应的 SPSR (saved program status register)

ARM寄存器分组


User mode
IRQ
FIQ
Undef
Abort
SVC
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
Note: System mode uses the User mode register set.
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
r13 (sp)
r14 (lr)
spsr
spsr
spsr
spsr
spsr
Current mode
Banked out registers

程序状态寄存器



- 条件位：
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Q 位：
  - 仅ARM 5TE/J架构支持
  - 指示饱和状态
- J 位
  - 仅ARM 5TE/J架构支持
  - J = 1: 处理器处于Jazelle状态
- 中断禁止位：
  - I = 1: 禁止 IRQ.
  - F = 1: 禁止 FIQ.
- T Bit
  - 仅ARM xT架构支持
  - T = 0: 处理器处于 ARM 状态
  - T = 1: 处理器处于 Thumb 状态
- Mode位：
  - 处理器模式位

---

v6以上新定义：

- GE[3:0:]: SIMD指令状态
- E: 大小endian设定； A: 禁止不正确的Abort
- IF[7:0]: Thumb-2指令IT状态位

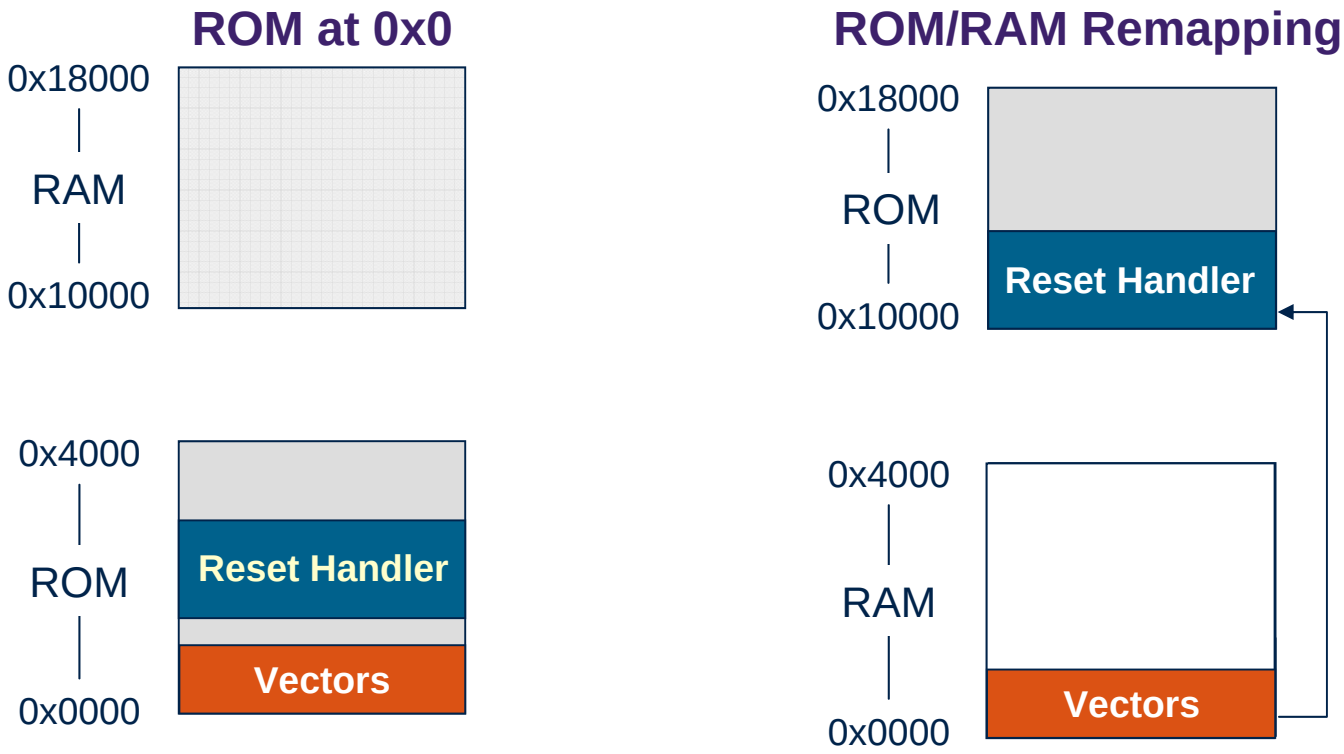
v6以前是undefined.

讨论内容

- 嵌入式ARM体系结构概述
- **常见ARM处理器启动过程**
- ARM处理器复位和初始化
- 嵌入式ARM软件程序设计
- 典型ARM启动代码分析
- 嵌入式软件的编译和调试

ROM or RAM at 0x0?

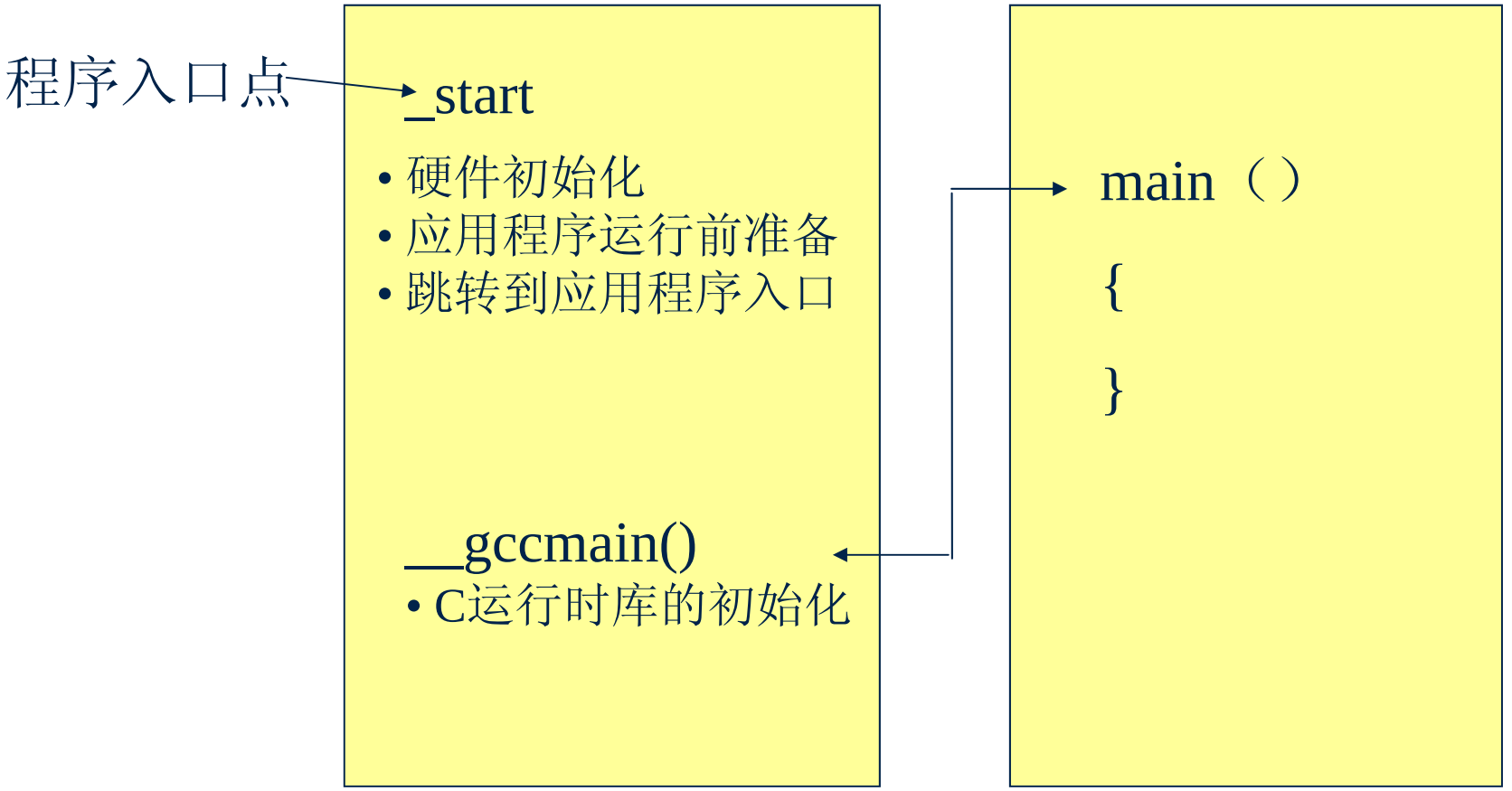
- 需要一个有效的地址在 **0x0**



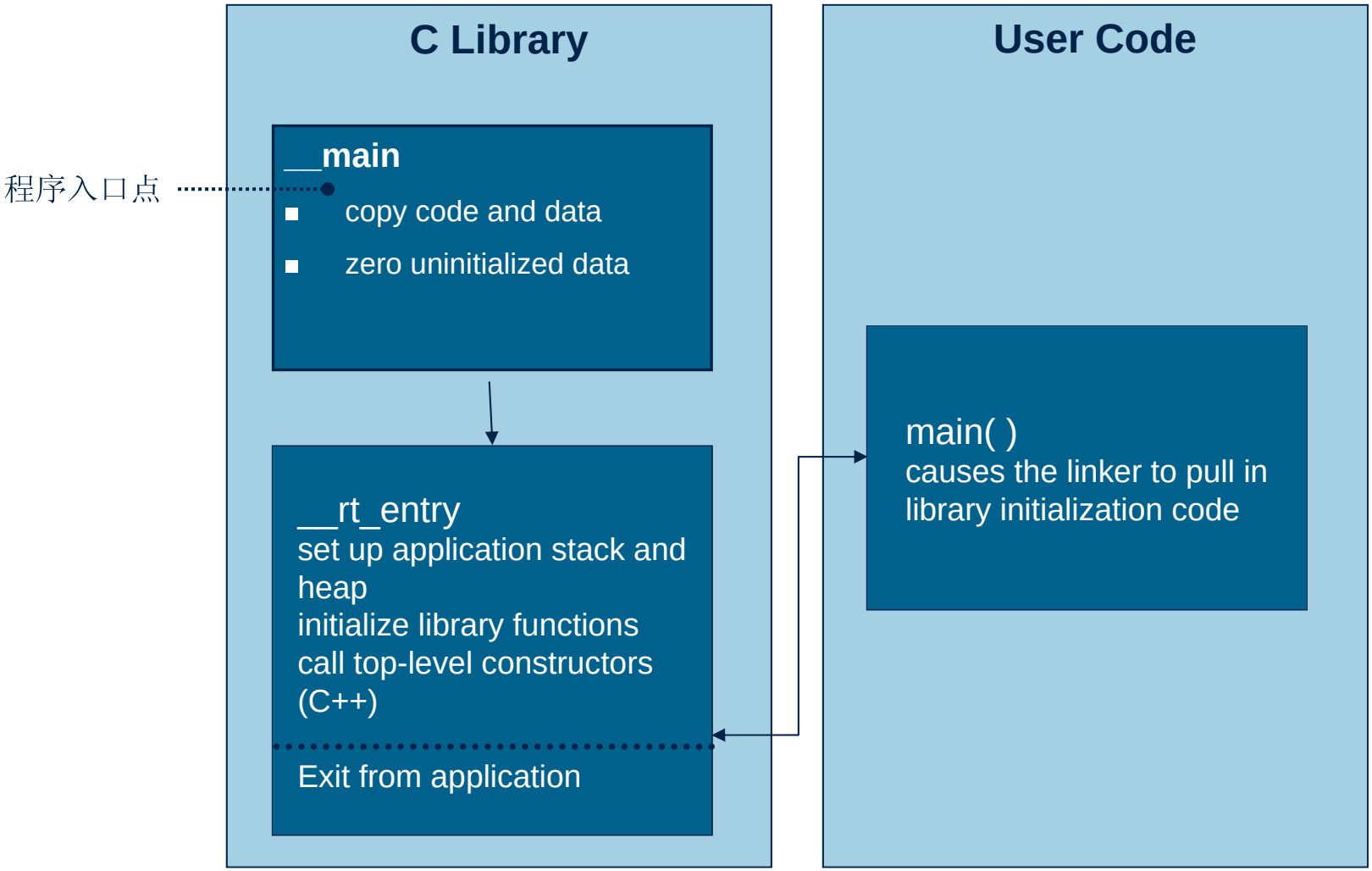
这项功能可被编码在像RESET HANDLER 一样的模块中

后面存储映射内容还会提到……

程序启动-GCC


程序入口点
_start
• 硬件初始化
• 应用程序运行前准备
• 跳转到应用程序入口
__gccmain()
• C运行时库的初始化
main（）
{
}

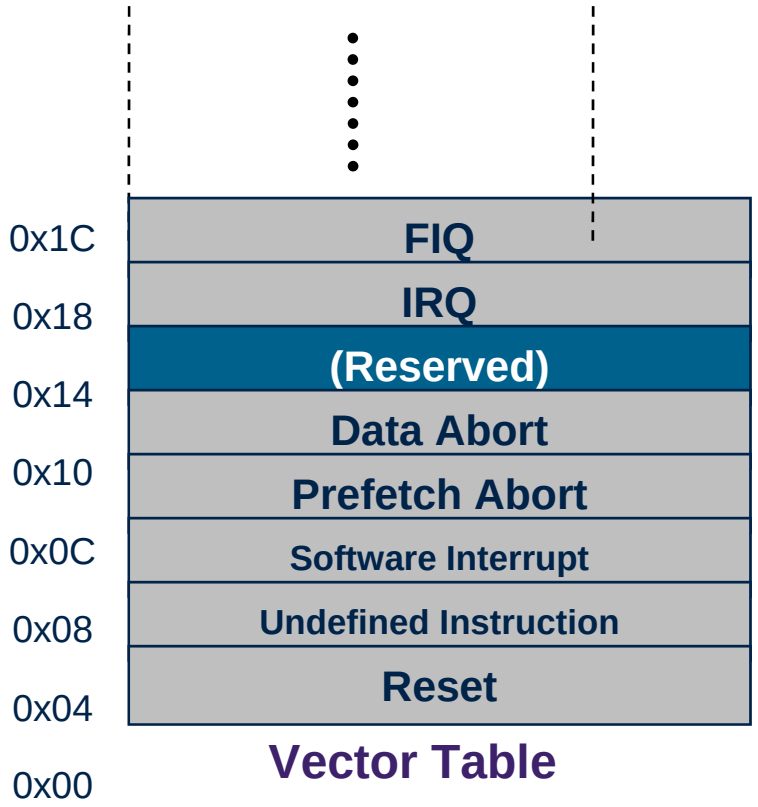
程序启动-ARM


C Library
User Code
程序入口点
__main
copy code and data
zero uninitialized data
__rt_entry
set up application stack and heap
initialize library functions
call top-level constructors (C++)
Exit from application
main( )
causes the linker to pull in library initialization code

讨论内容

- 嵌入式ARM体系结构概述
- 常见ARM处理器启动过程
- **ARM处理器复位和初始化**
- 嵌入式ARM软件程序设计
- 典型ARM启动代码分析
- 嵌入式软件的编译和调试

ARM处理器异常



Vector Table

Vector table can be at
0xFFFF0000 on ARM720T
and on ARM9/10 family devices

异常向量表

```
        AREA Vectors, CODE, READONLY
        IMPORT Reset_Handler
; import other exception handlers
        ; …
   ENTRY
        B       Reset_Handler
        B       Undefined_Handler
        B       SWI_Handler
        B       Prefetch_Handler
        B       Data_Handler
        NOP     ; Reserved vector
        B       IRQ_Handler
                ; FIQ_Handler will follow directly
                END
```

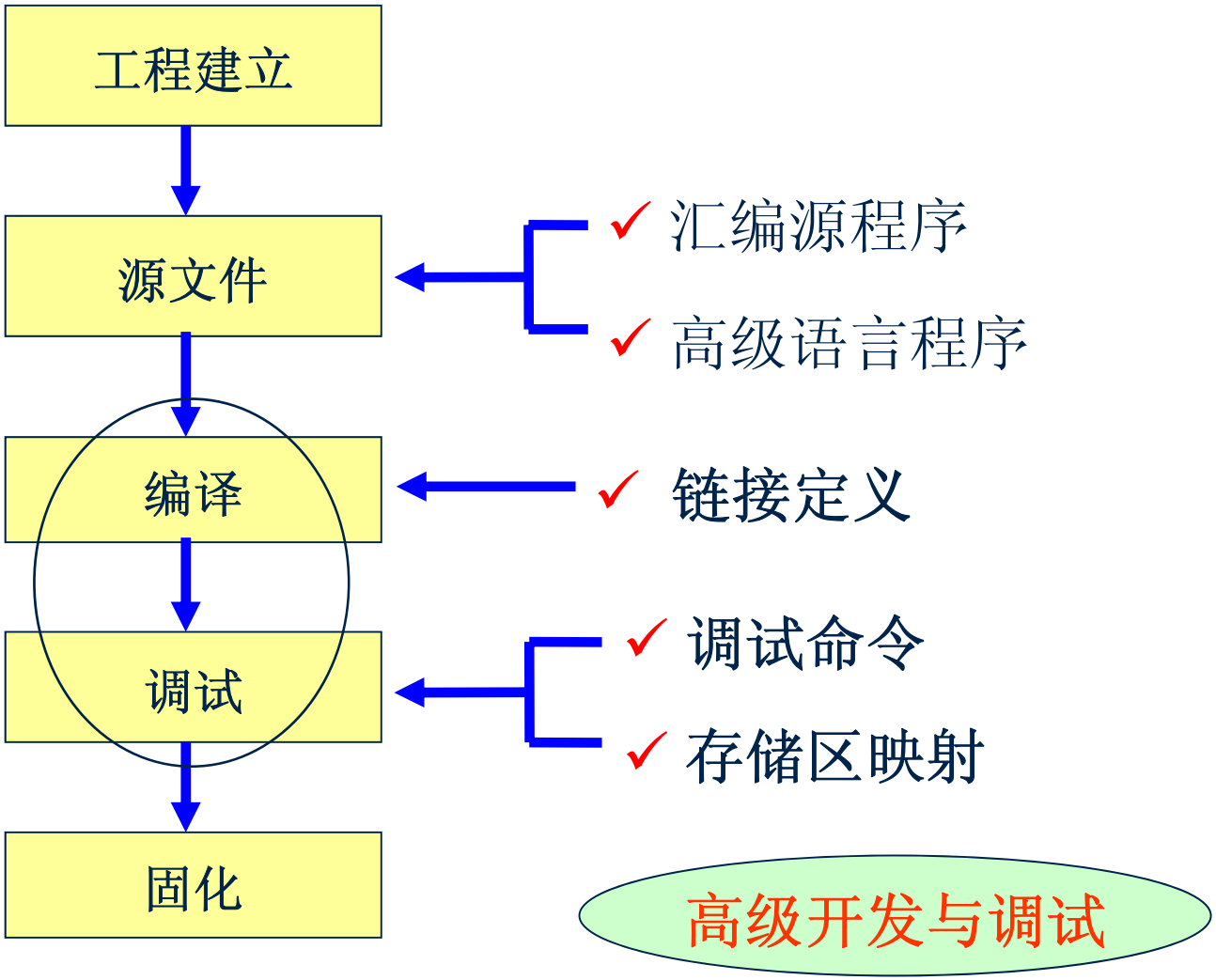
在使用scatterloading+FIRST时直接定位在0X0（或 0xFFFF0000）

ENTRY 直接告诉链接器这是一个入口点，防止某些段被删除

讨论内容

- 嵌入式ARM体系结构概述
- 常见ARM处理器启动过程
- ARM处理器复位和初始化
- **嵌入式ARM软件程序设计**
- 典型ARM启动代码分析
- 嵌入式软件的编译和调试

嵌入式软件开发流程



高级语言中的数据类型

- **char** **8 bit byte**
- **short** **16 bit half-word**
- **int** **32 bit word**
- **long** **32 bit integer**
- **float** **32 bit IEEE 单精度**
- **double** **64 bit IEEE 双精度**
- **pointer** **32 bits**
- **long long** **64 bit integer**
- **bool** **8 bit byte**
- **wchar_t** **32 bit word**

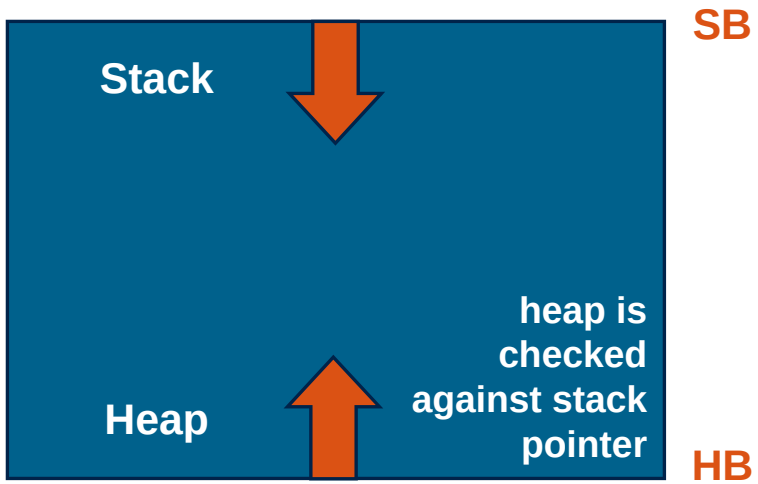
- **除char和pointer型数据外，缺省都是有符号数**

标准函数库

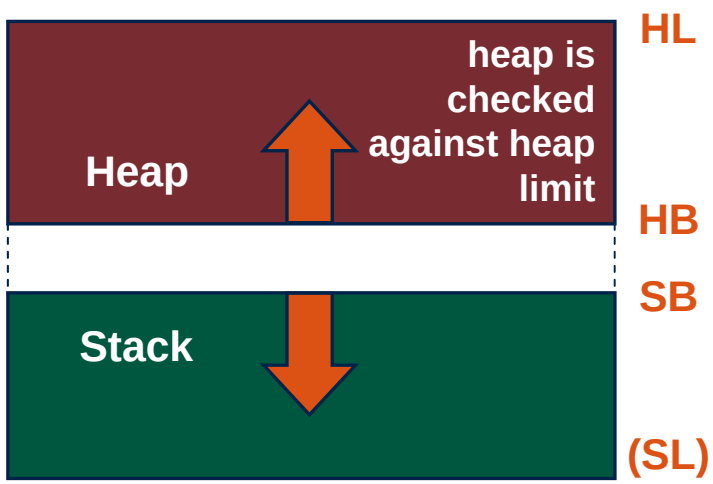
- 输入输出
  - 标准C库函数功能是支持PC软件的
  - 目标板上的可执行软件则依赖相关的硬件资源
  - 通常需要重新定向输入输出

堆和栈

- 你必须决定在放置**stack**和**heap**时所使用的区域是单一的区**(one-region model)**或是不同的两个区**(two-region model)**



One region model

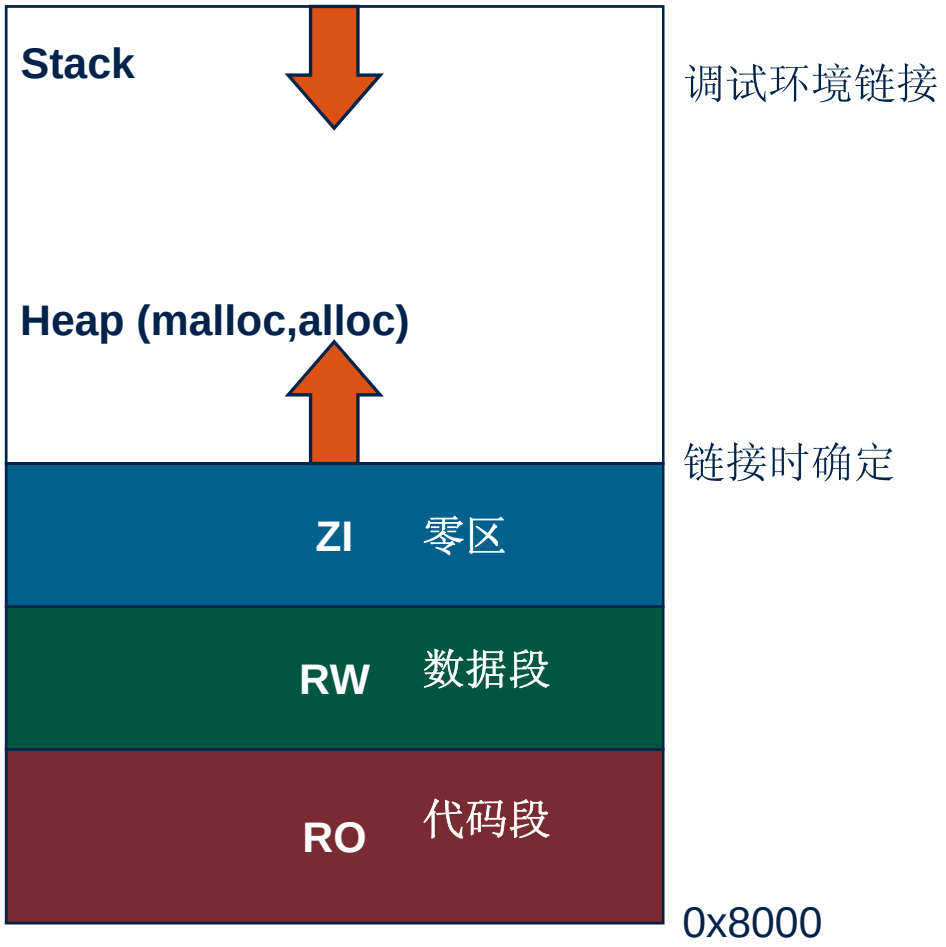


Two region model

单一存储器模式是默认方式
为了实现多区域模式，你可以使用 use_two_region_memory
在所有的模式下，软件堆栈检查要许可。
编译开关是： -apcs /swst
指定堆栈限制 (为 two-region 模式)

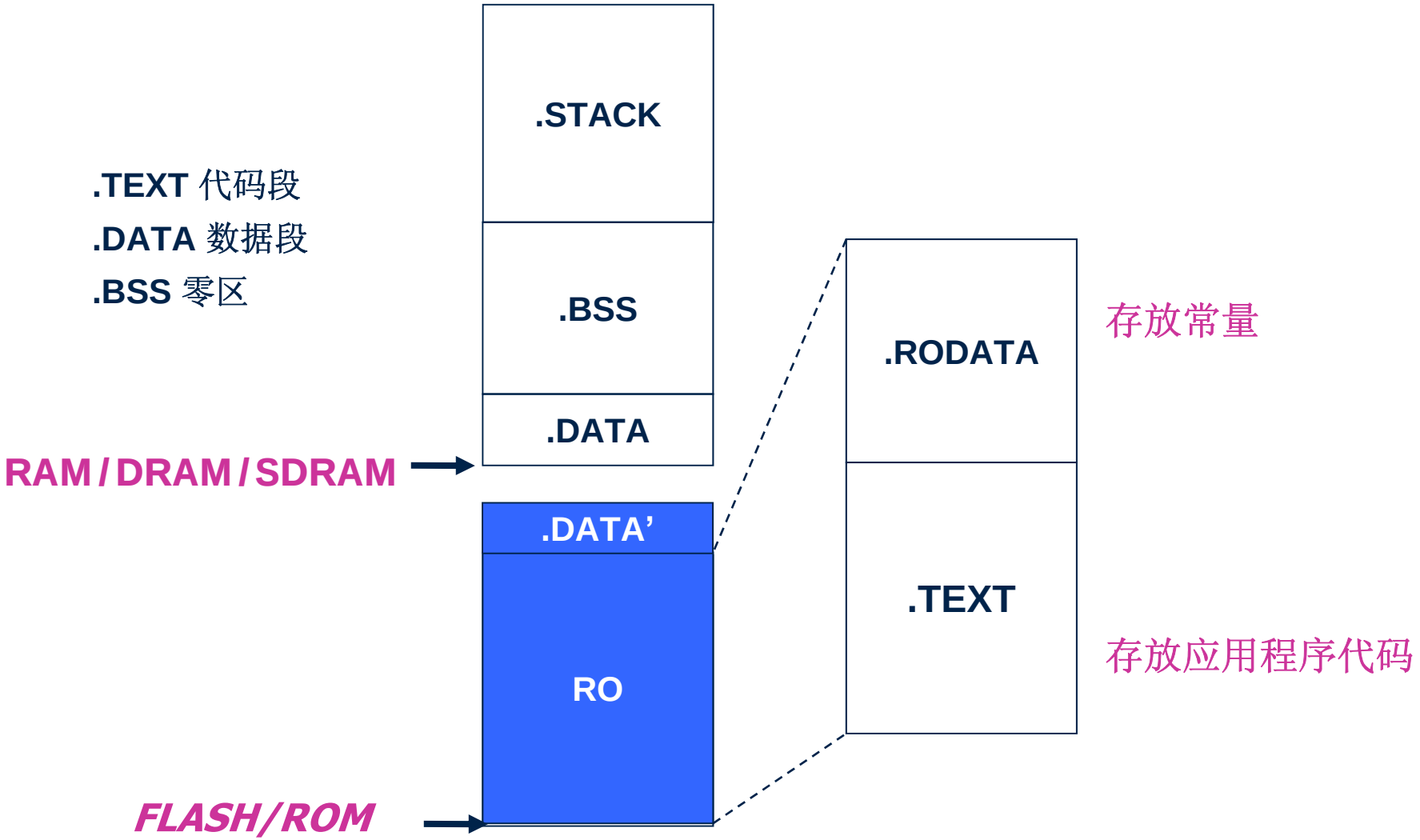
链接映象段--ARM

- 默认情况下，链接、定位、运行于 **0x8000**
- **heap** 被直接放置在数据区的上面
- 堆栈的基地址是通过调试环境从**C**库函数的**Startup Code** 里读取出来的。
- ARMulator **=> from configuration file (peripherals.ami)**
  - default = 0x08000000
  - **Multi-ICE** => from debugger internal variable $top_of_memory
    - default = 0x80000



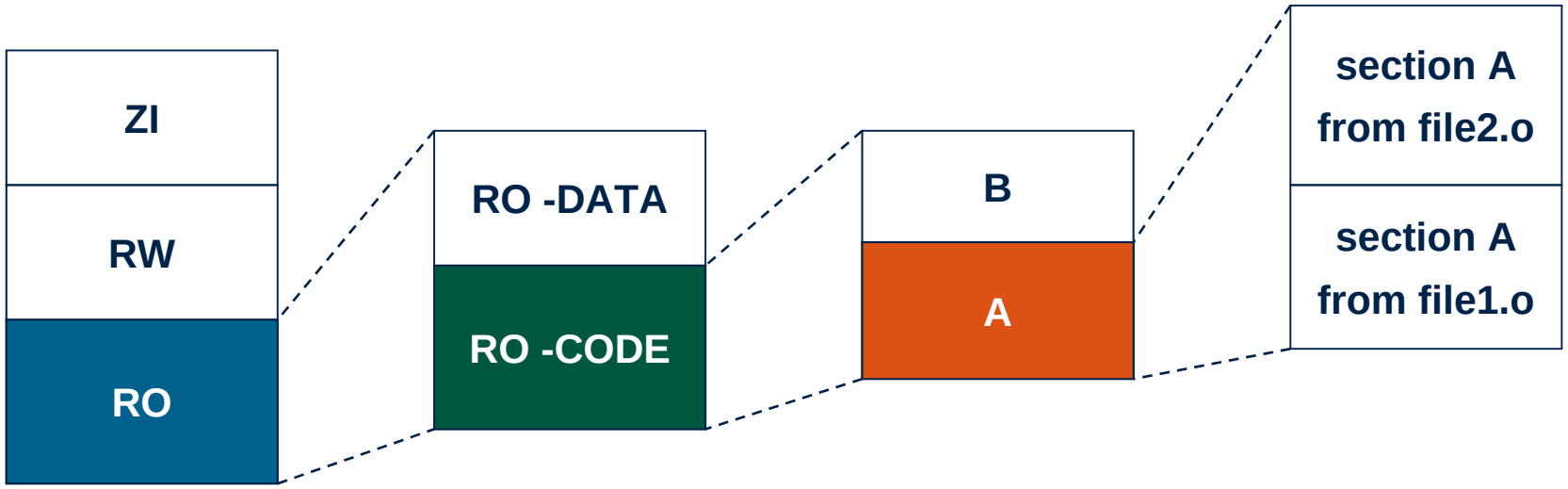
链接映象段--GCC

.TEXT 代码段
.DATA 数据段
.BSS 零区



链接器放置规则

- 在每个可执行区，链接器通过一些基本的规则来放置**CODE** 和**DATA**
- 基本的排序方法是通过属性来安排的:
  - RO 领先于RW ， RW 领先于ZI
  - 有相同的属性时，CODE 在DATA之前放置。
- 更多的排序方法决定于:
  - 输入的组名按字母排序,
  - 在ARMLINK命令行中指定的顺序。
    - eg: **`armlink file1.o file2.o`** …

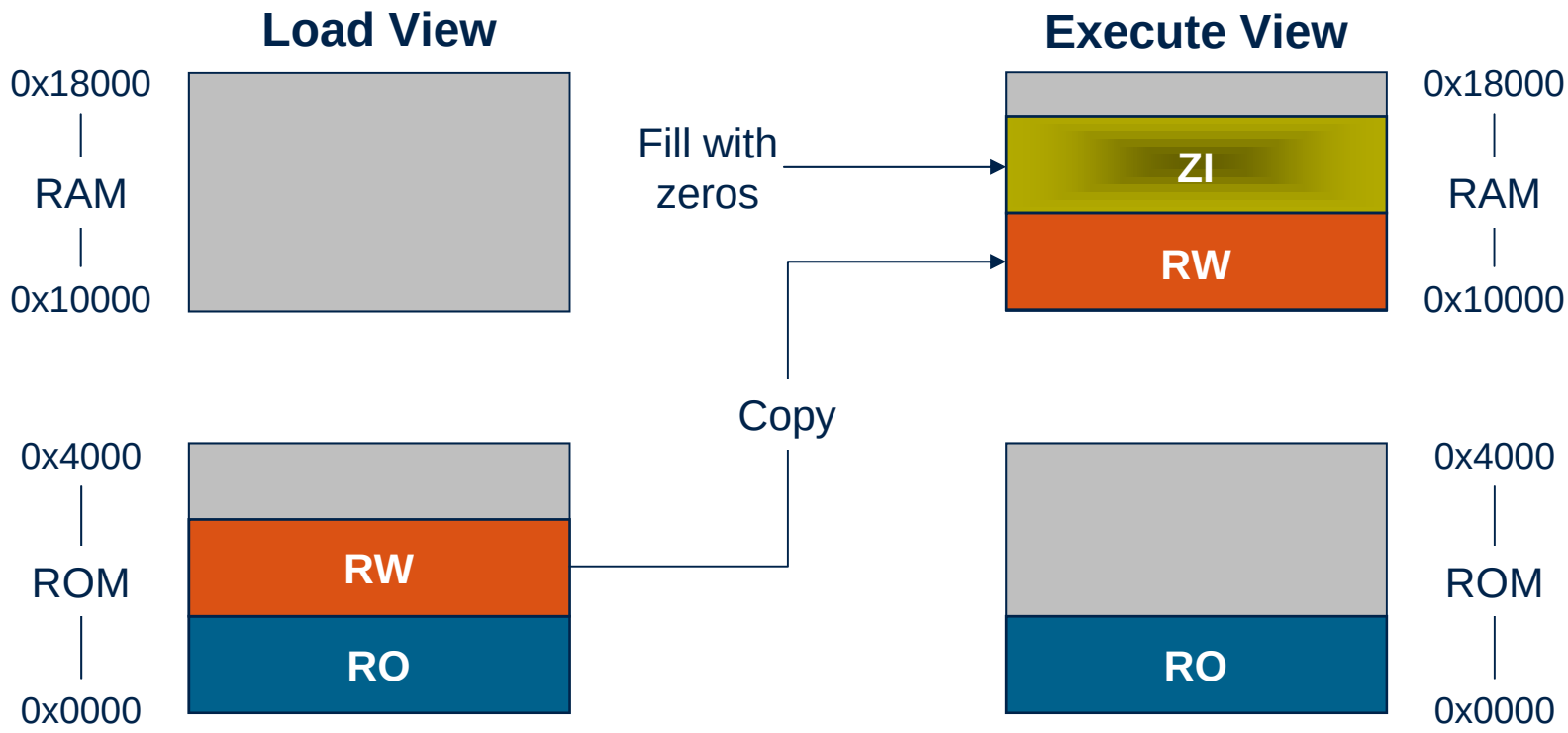


程序链接定位

- 程序链接定位，也叫**Scatter Loading**
  - 在系统级别的嵌入式开发中需要对映象链接定位，即重定位映象段的存放，包括代码段，数据段，零区等，链接器必须使用按约定或指定规则，对整个系统的代码做正确的定位，这些规则通常写成链接脚本
- 在一个实际应用当中，可能并不想在**0x8000**处开始运行
  - 嵌入式系统各种存储器设备的地址空间在整个存储器映射中经常交叉出现
  - 链接脚本提供一种把代码和数据放在不同存储器定位
- 链接脚本描述了存储器映射定义，保存为文件可作为参数给链接器使用

eg: `armlink program.o -scatter scatter.scf -o program.axf`

程序链接定位实例



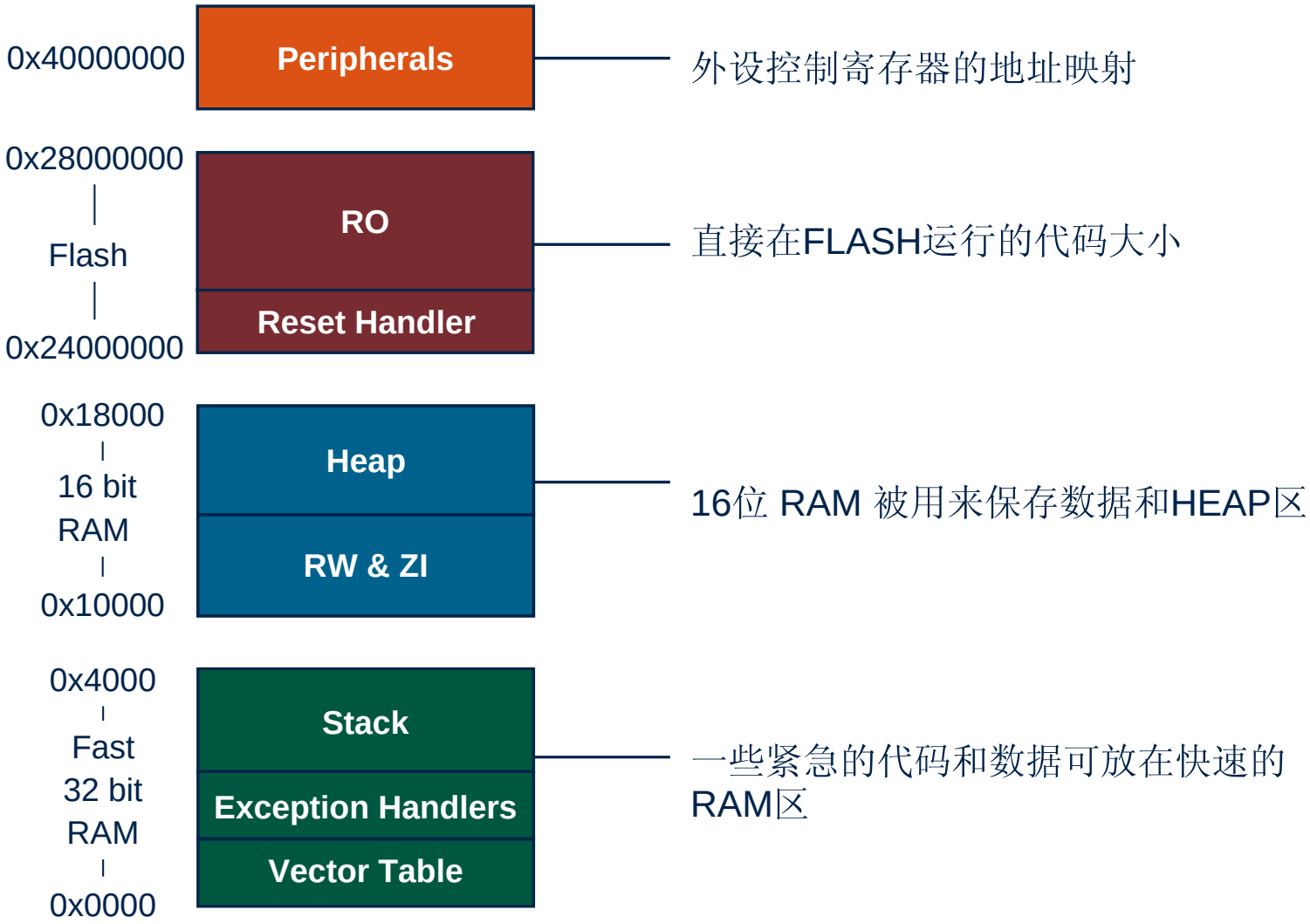
只读代码和数据保存在ROM中

C库函数初始化代码 (在 __main) 将 :

从ROM拷贝RW数据到RAM

在RAM中的ZI 数据初始化

存储器分配实例


0x40000000
Peripherals
外设控制寄存器的地址映射
0x28000000
Flash
0x24000000
RO
Reset Handler
直接在FLASH运行的代码大小
0x18000
16 bit
RAM
0x10000
Heap
RW & ZI
16位 RAM 被用来保存数据和HEAP区
0x4000
Fast
32 bit
RAM
0x0000
Stack
Exception Handlers
Vector Table
一些紧急的代码和数据可放在快速的RAM区

段的开始及结束

- 链接器产生各个映象段的基地址和限制地址的符号指针
  - 可在任意代码中引入并使用这些符号

```
IMPORT ||Image$$RO$$Base||
IMPORT ||Image$$STACK$$ZI$$Base||
IMPORT ||Image$$RW$$Base||
IMPORT ||Image$$RW$$Limit||

code_base  DCD ||Image$$RO$$Base||
stack_base DCD ||Image$$STACK$$ZI$$Base||
data_base  DCD ||Image$$RW$$Base ||
data_limit DCD ||Image$$RW$$Limit ||
```

```
.extern  Image_RO_Base
.extern  Image_RO_Limit
.extern  Image_RW_Base
.extern  Image_ZI_Base
.extern  Image_ZI_Limit

.equ code_base,Image_RO_Base
.equ stack_base, Image_ZI_Limit
.equ data_base, Image_RW_Base
.equ data_limit, Image_ZI_Limit
```

- 重命名这些段，需要时直接使用：

```
ldr   r1,=code_base
ldr   r3,=data_limit
```

讨论内容

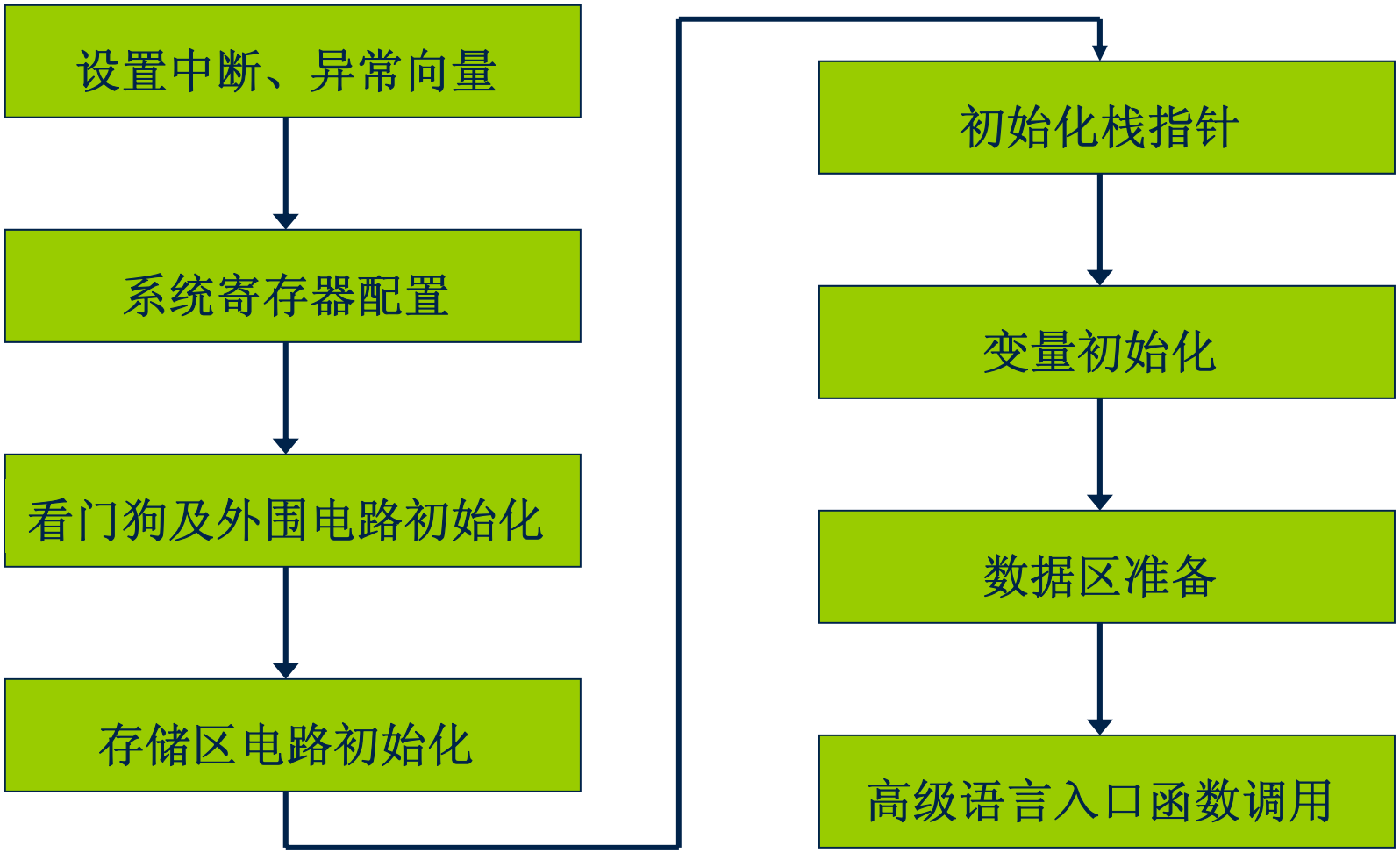
- 嵌入式ARM体系结构概述
- 常见ARM处理器启动过程
- ARM处理器复位和初始化
- 嵌入式ARM软件程序设计
- **典型ARM启动代码分析**
- 嵌入式软件的编译和调试

启动代码

- 启动代码是用来初始化电路以及为高级语言写的软件做好运行前准备的一小段汇编语言，是任何处理器上电复位时的程序运行入口点
  - 功能
    - 初始化电路
    - 为高级语言编写的软件运行做准备
  - 特征
    - 汇编语言
    - 处理器上电复位的程序运行入口点
- 系统初始化代码通常在进入主应用之前运行
  - reset handler 不是一个适合使能中断和使能caches地方
  - 在reset handler最后应该放一个C运行库初始化代码

```
IMPORT  __main
B       __main
```

启动代码流程


设置中断、异常向量
系统寄存器配置
看门狗及外围电路初始化
存储区电路初始化
初始化栈指针
变量初始化
数据区准备
高级语言入口函数调用

启动代码最小流程

异常向量表初始化

↓

存储区配置

↓

初始化堆栈

↓

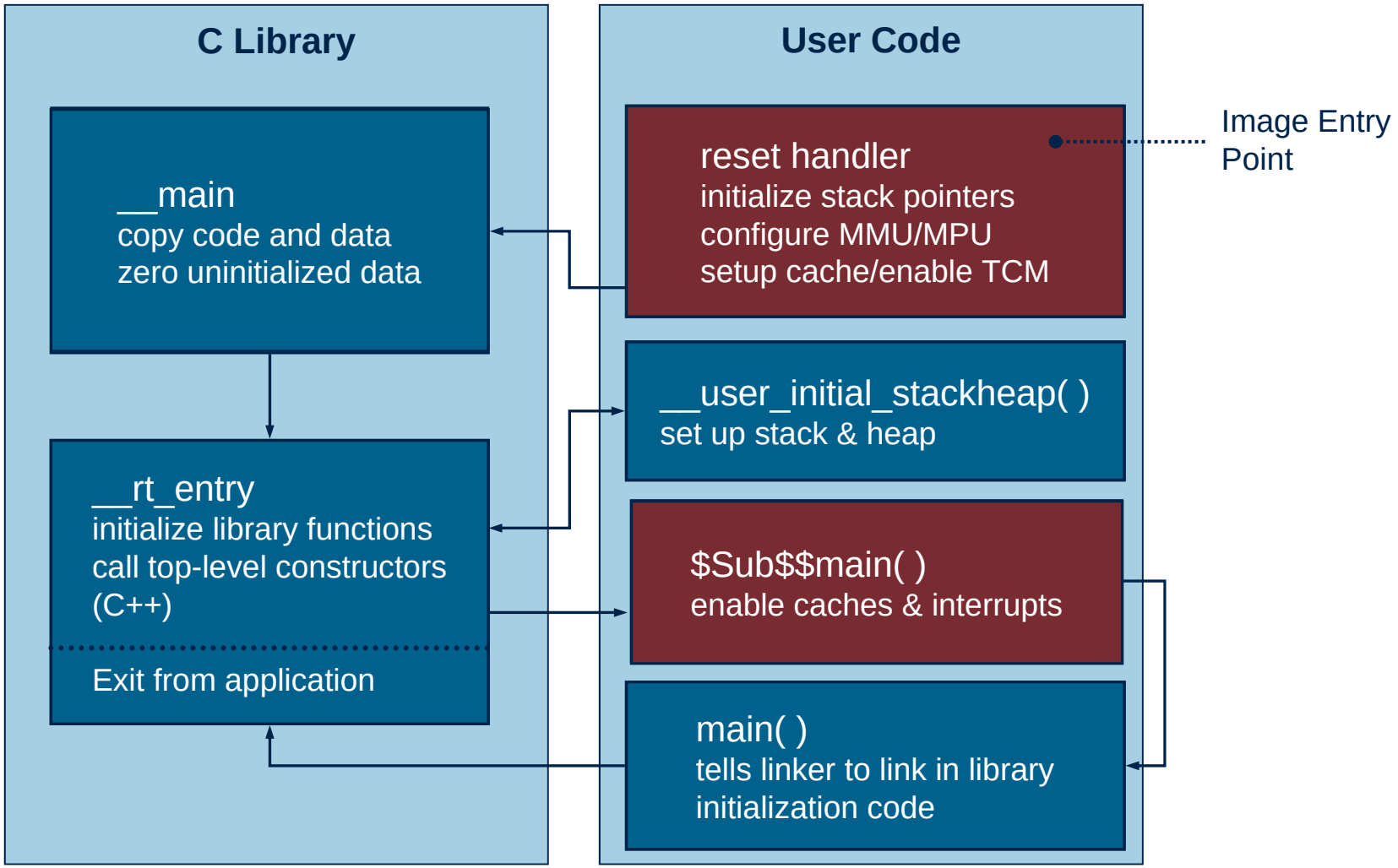
高级语言入口函数调用

Main( )
Rtos_Init( )
MyRoot( )
……

启动代码常见形式

- **Bootloader**
  - 典型提供1-2种通信接口驱动
  - 要求实现某种通信协议
  - 通常可加载运行某特定应用程序或操作系统
- **BSP (Board Support Package)**
  - 丰富的接口驱动及协议支持软件
  - 有特定的设备驱动层接口
- **BIOS**
  - 有指定的输入输出设备
  - 具有设备检测功能
  - 通常有系统自检能力

应用程序运行步骤

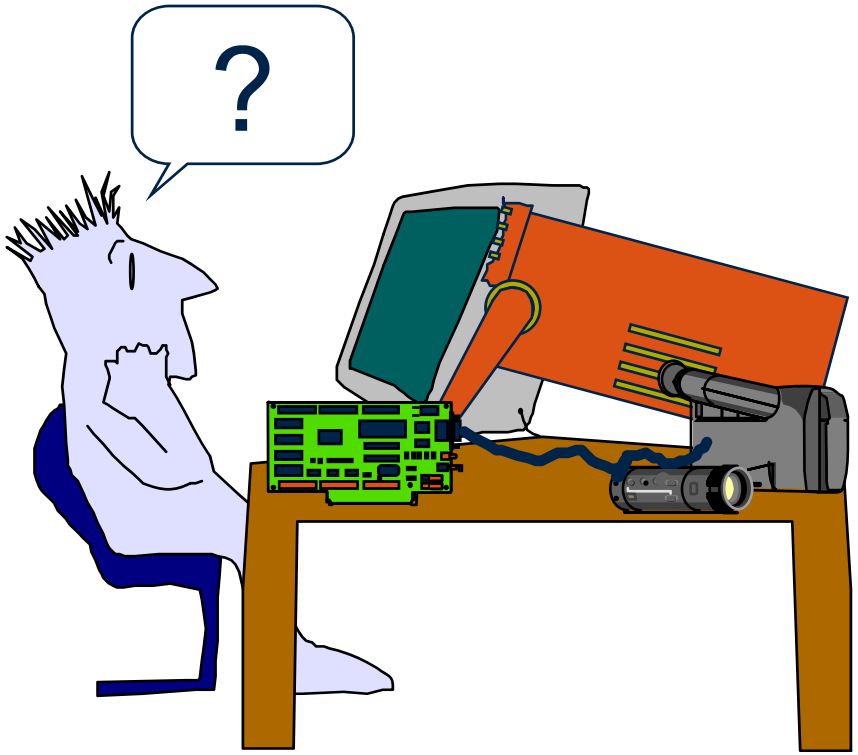

C Library
User Code
__main
copy code and data
zero uninitialized data
__rt_entry
initialize library functions
call top-level constructors
(C++)
Exit from application
reset handler
initialize stack pointers
configure MMU/MPU
setup cache/enable TCM
Image Entry Point
__user_initial_stackheap( )
set up stack & heap
$Sub$$main( )
enable caches & interrupts
main( )
tells linker to link in library
initialization code

讨论内容

- 嵌入式ARM体系结构概述
- 常见ARM处理器启动过程
- ARM处理器复位和初始化
- 嵌入式ARM软件程序设计
- 典型ARM启动代码分析
- **嵌入式软件的编译和调试**

基本的调试需求

- 运行监控
  - 设置断点
  - 代码的单步执行
- 状态查看
  - 处理器状态，读写寄存器值
  - 系统状态，内存访问
- 故障定位
  - 异常捕获
  - 故障跟踪
- 硬件检测
  - 在线仿真调试



- 如果您想要更清楚地答复更多疑问

欢迎参加嵌入式开发技术培训班

ARM开发培训班（ATC授权培训）

- ARM体系结构
- ARM编程、开发流程
- 系统移植的方法
- Bootloader的基本概念