暨南大學
JINAN UNIVERSITY

第7章 嵌入式软件系统基础

杨光华

物联网与物流工程研究院 / 电气信息学院
办公室：行政楼 631
电邮：ghyang@jnu.edu.cn 电话：8505687

主要内容

- 嵌入式软件系统概述
- 嵌入式操作系统
- 嵌入式软件开发工具

第一节

嵌入式软件系统概述

- 软件系统
- 嵌入式软件系统的分类
- 嵌入式软件系统的体系结构
- 嵌入式软件运行流程

软件系统

- 软件(software)是计算机系统中与硬件(hardware)相互依存的另一部分，它包括程序(program)、相关数据(data)及其说明文档(document)
  - **程序**：按事先设计的功能和性能要求执行的指令序列
  - **数据**：程序能正常操纵信息的数据结构
  - **文档**：是与程序开发维护和使用有关的各种图文资料

嵌入式软件系统的分类

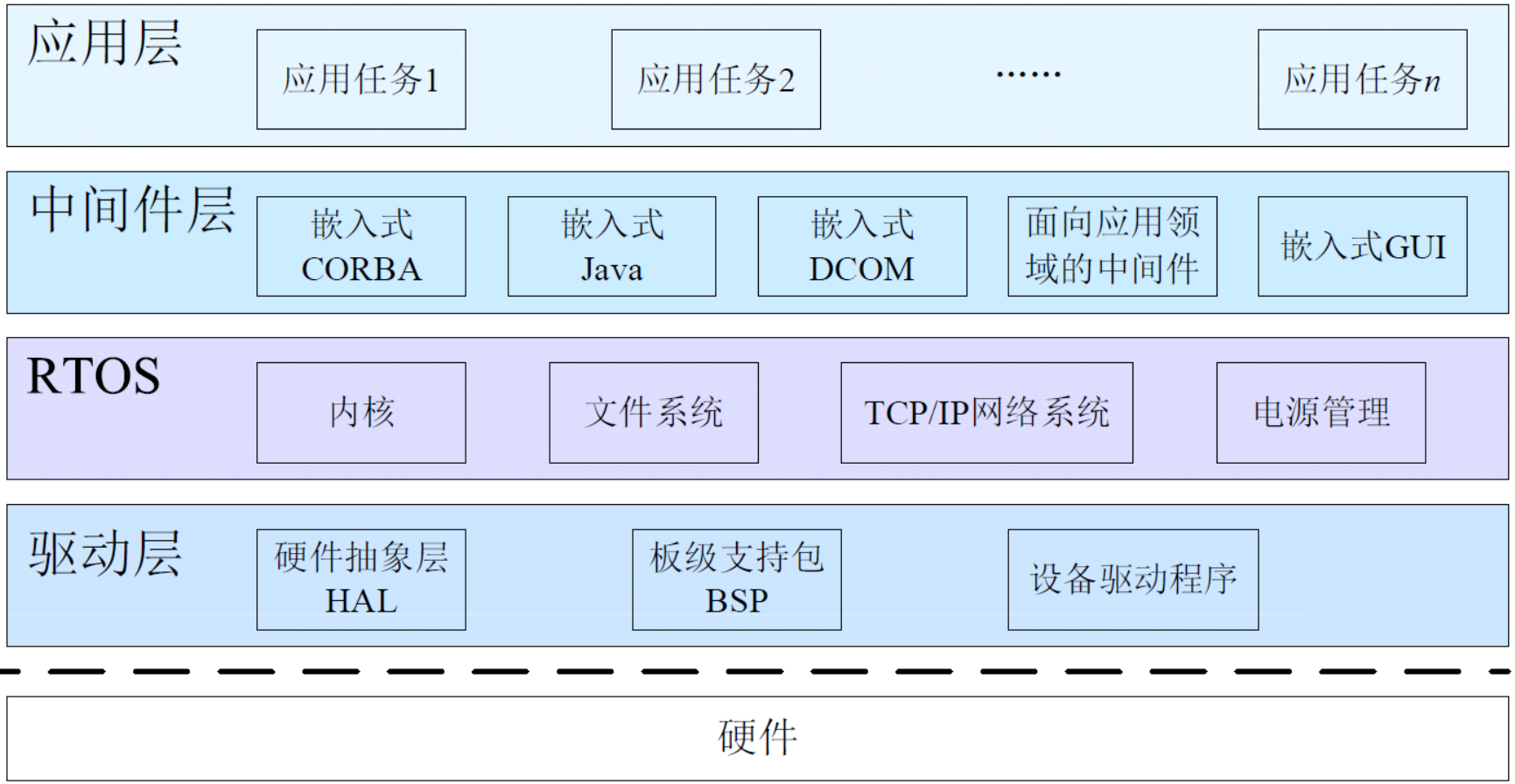
嵌入式软件

- 系统软件：控制、管理计算机系统的资源
  - 嵌入式操作系统
  - 嵌入式中间件（CORBA、Java）
  - 等等
- 支撑软件：辅助软件开发的工具
  - 系统分析设计工具
  - 仿真开发工具
  - 交叉开发工具
  - 测试工具
  - 配置管理工具
  - 维护工具等
- 应用软件：面向应用领域
  - 手机软件
  - 路由器软件
  - 交换机软件
  - 飞控软件等

嵌入式软件系统的分类

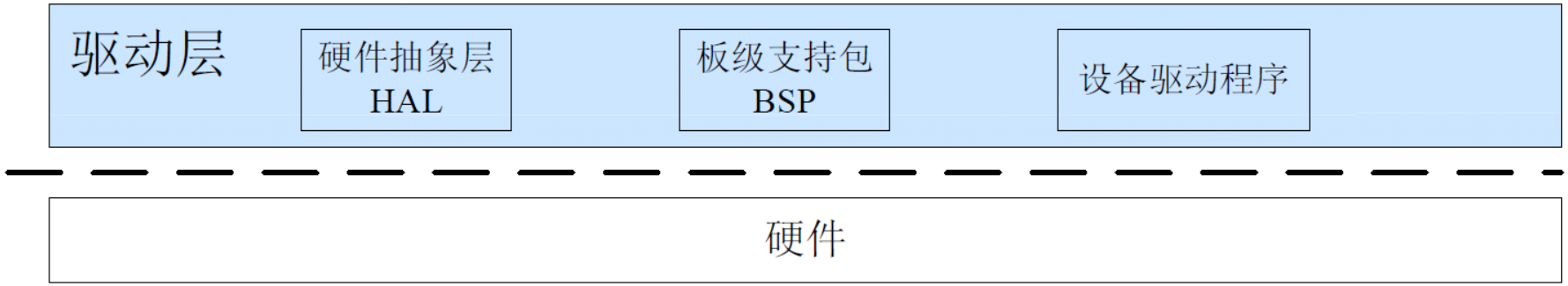
- 从运行平台来分，嵌入式软件可以分为
  - **运行在开发平台上的软件**：设计、开发、测试工具等
  - **运行在嵌入式系统上的软件**：嵌入式操作系统、应用程序、驱动程序及部分开发工具

嵌入式软件系统的体系结构

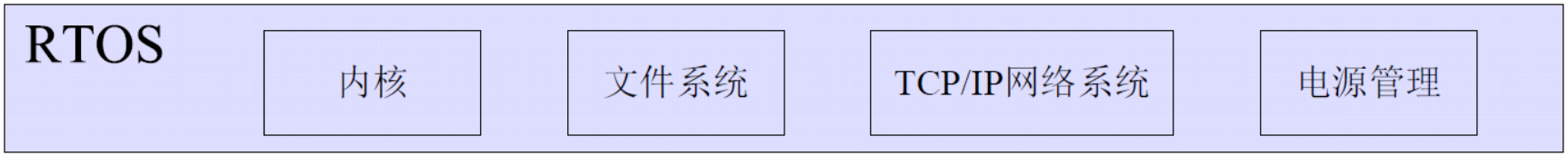


嵌入式软件系统的体系结构 - 驱动层

- 是嵌入式系统中必不可少的重要部分
- 提供所有的外围设备的支持和给上层软件提供设备的操作接口

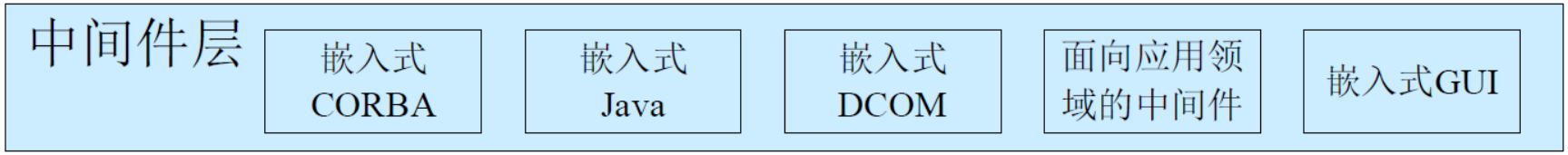


嵌入式软件系统的体系结构- RTOS



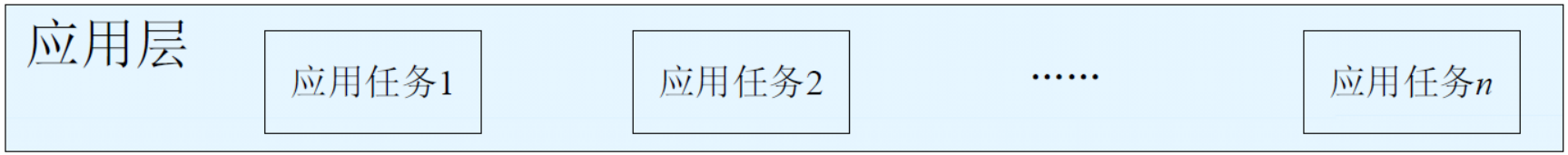
- 主要完成设备和资源的管理
- 内核是必需的，完成基本的功能，包括任务调度和管理、内存管理等功能
- 其他部分可选，包括文件系统、驱动程序、网络通信，可以根据系统功能、成本进行裁减

嵌入式软件系统的体系结构 - 中间件层



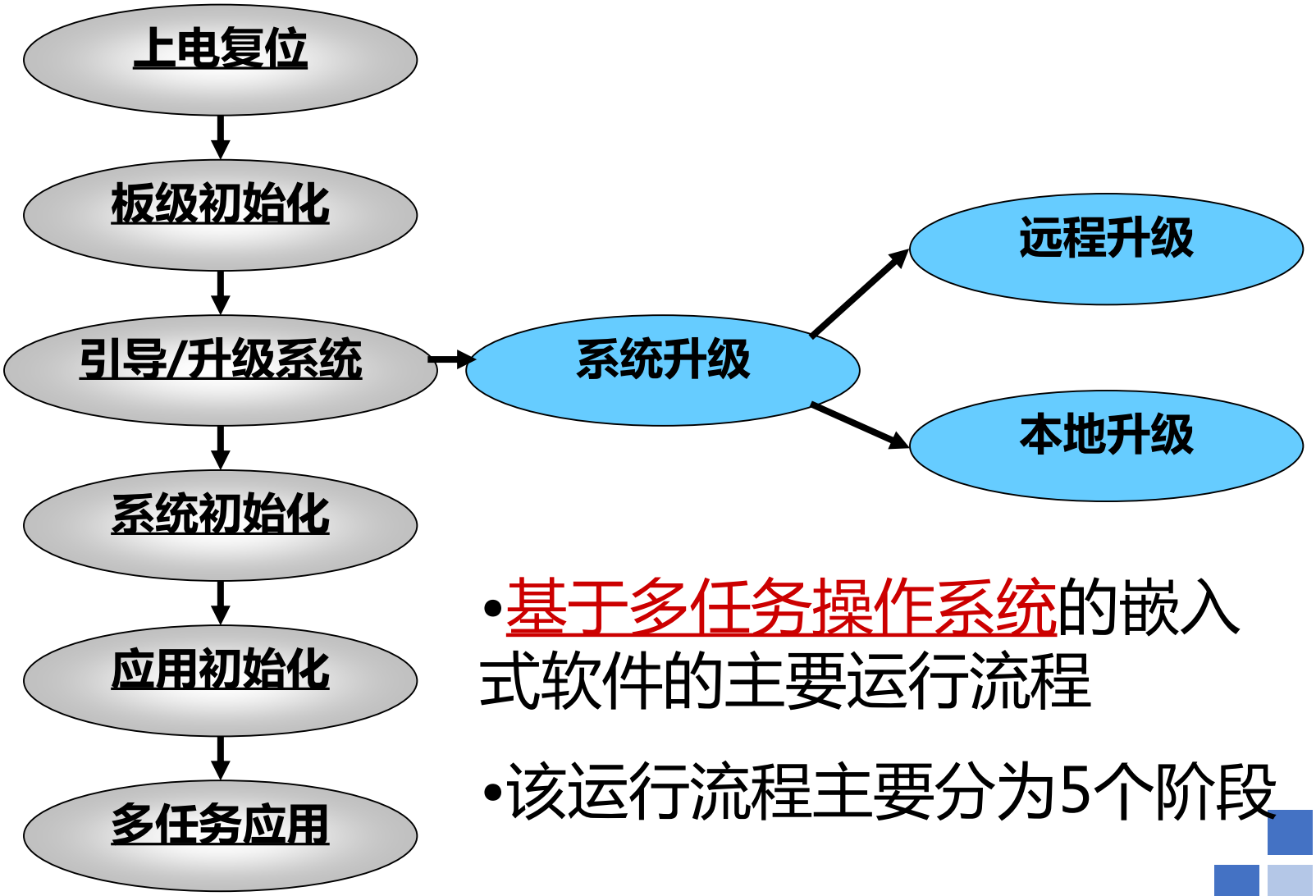
- 中间件是一类连接软件组件和应用的计算机软件，它包括一组服务，以便运行在一台或多台机器上的多个软件通过网络进行交互
- 目前在一些复杂的嵌入式系统中采用中间件技术，主要包括
  - 嵌入式CORBA（跨语言、跨OS互操作）
  - 嵌入式Java
  - 嵌入式DCOM
  - 嵌入式GUI
  - 面向专门应用领域的中间件

嵌入式软件系统的体系结构 - 应用层



- 指操作系统之上运行的用户程序
- 主要由多个相对独立的应用任务组成
- 每个任务完成特定的工作，如
  - I/O任务、计算的任务、通信的任务
- 由操作系统调度各个任务的运行

嵌入式软件运行流程



- 基于多任务操作系统的嵌入式软件的主要运行流程
- 该运行流程主要分为5个阶段

嵌入式软件运行流程

- 上电复位、板级初始化阶段
  - 嵌入式系统上电复位后完成板级初始化工作
  - 板级初始化程序具有硬件特性，一般采用汇编语言实现。不同的嵌入式系统，板级初始化要完成的工作有一定的特殊性，但以下工作一般是必须完成的：
    - CPU芯片级初始化：时钟、中断、DMA、内存等的初始化
    - CPU中堆栈指针寄存器的初始化
    - BSS段（Block Storage Space表示未被初始化的数据）的初始化

嵌入式软件运行流程

- 系统引导/升级阶段
  - 根据需要分别进入系统软件引导阶段或系统升级阶段
  - 软件可通过测试通信端口数据或判断特定开关的方式分别进入不同阶段

嵌入式软件运行流程

- 系统引导阶段

  系统引导的几种情况

  - 将系统软件从NOR Flash中读取出来加载到RAM中运行：这种方式可以解决成本及Flash速度比RAM慢的问题。软件可压缩存储在Flash中
  - 不需将软件引导到RAM中而是让其直接在NOR Flash上运行，进入系统初始化阶段
  - 将软件从外存（如Nand Flash、CF卡、MMC等）中读取出来加载到RAM中运行：这种方式的成本更低

嵌入式软件运行流程

- 系统升级阶段
  - 进入系统升级阶段后系统可通过网络进行远程升级或通过串口进行本地升级
  - 远程升级一般支持TFTP、FTP、HTTP等方式
  - 本地升级可通过Console口使用超级终端或特定的升级软件进行

嵌入式软件运行流程

- 系统初始化阶段
  - 在该阶段进行操作系统等系统软件各功能部分必需的初始化工作，如根据系统配置初始化数据空间、初始化系统所需的接口和外设等
  - 系统初始化阶段需要按特定顺序进行，如首先完成内核的初始化，然后完成网络、文件系统等的初始化，最后完成中间件等的初始化工作

嵌入式软件运行流程

- 应用初始化阶段

在该阶段进行应用任务的创建，信号量、消息队列的创建和与应用相关的其它初始化工作

- 多任务应用运行阶段

各种初始化工作完成后，系统进入多任务状态，操作系统按照已确定的算法进行任务的调度，各应用任务分别完成特定的功能

第二节

嵌入式操作系统

概述
嵌入式操作系统的演变
嵌入式操作系统分类
嵌入式操作系统体系结构
嵌入式操作系统的组成
嵌入式实时操作系统μC/OS-II简介

概述

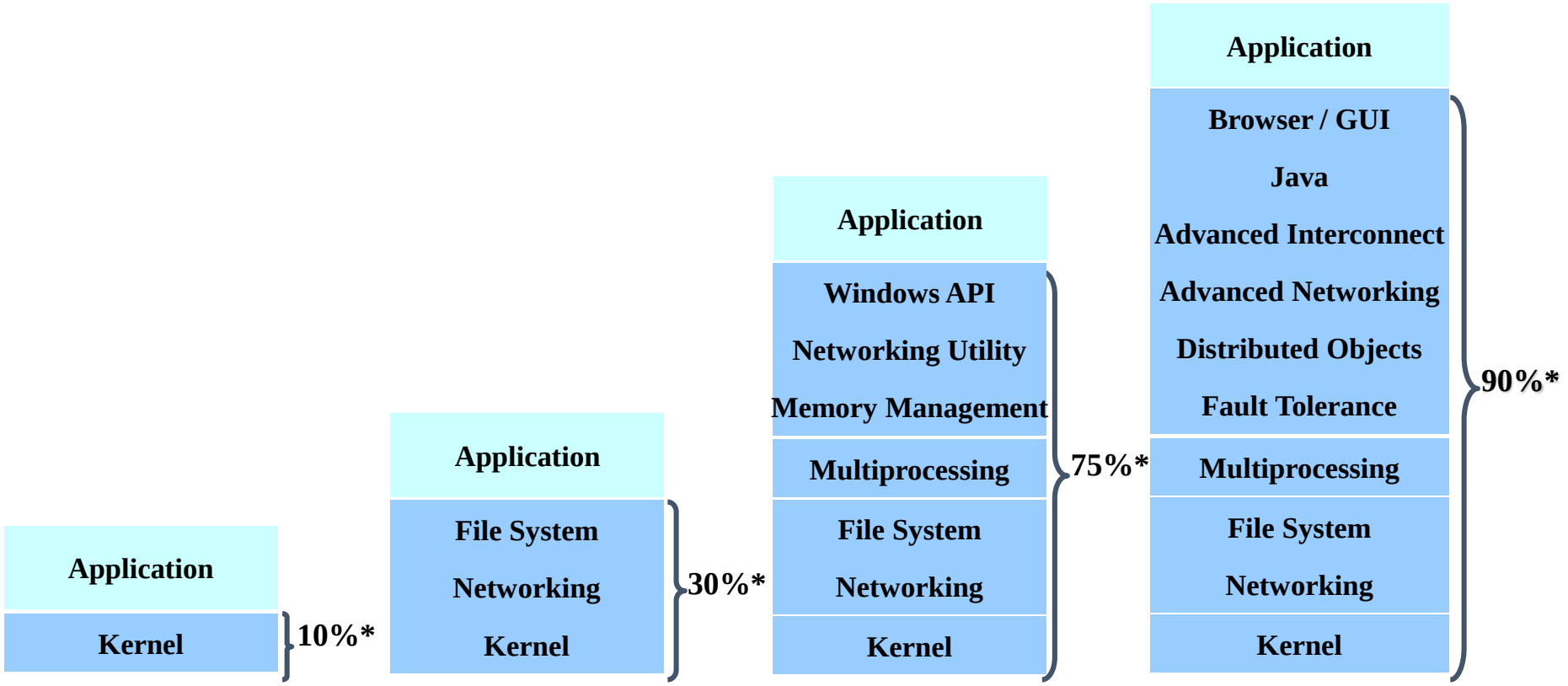
嵌入式操作系统可以统称为应用在嵌入式系统的操作系统，它具有一般操作系统的功能，同时具有嵌入式软件的特点，主要有：

- 可固化
- 可配置、可剪裁
- 独立的板级支持包，可修改
- 不同的CPU有不同的版本
- 应用的开发需要有集成的交叉开发工具

概述

- 近十年来，嵌入式操作系统得到飞速的发展
  - 从支持8位微处理器到16位、32位甚至64位微处理器
  - 从支持单一品种的微处理器芯片到支持多品种微处理器芯片
  - 从只有内核到除了内核外还提供其他功能模块，如文件系统，TCP/IP网络系统，UI系统等
- 随着嵌入式系统应用领域的扩展，目前嵌入式操作系统的市场在不断细分，出现了针对不同领域的产品，这些产品按领域的要求和标准提供特定的功能

嵌入式操作系统的演变



*Percent of total software supplied by RTOS vendor in a typical embedded device

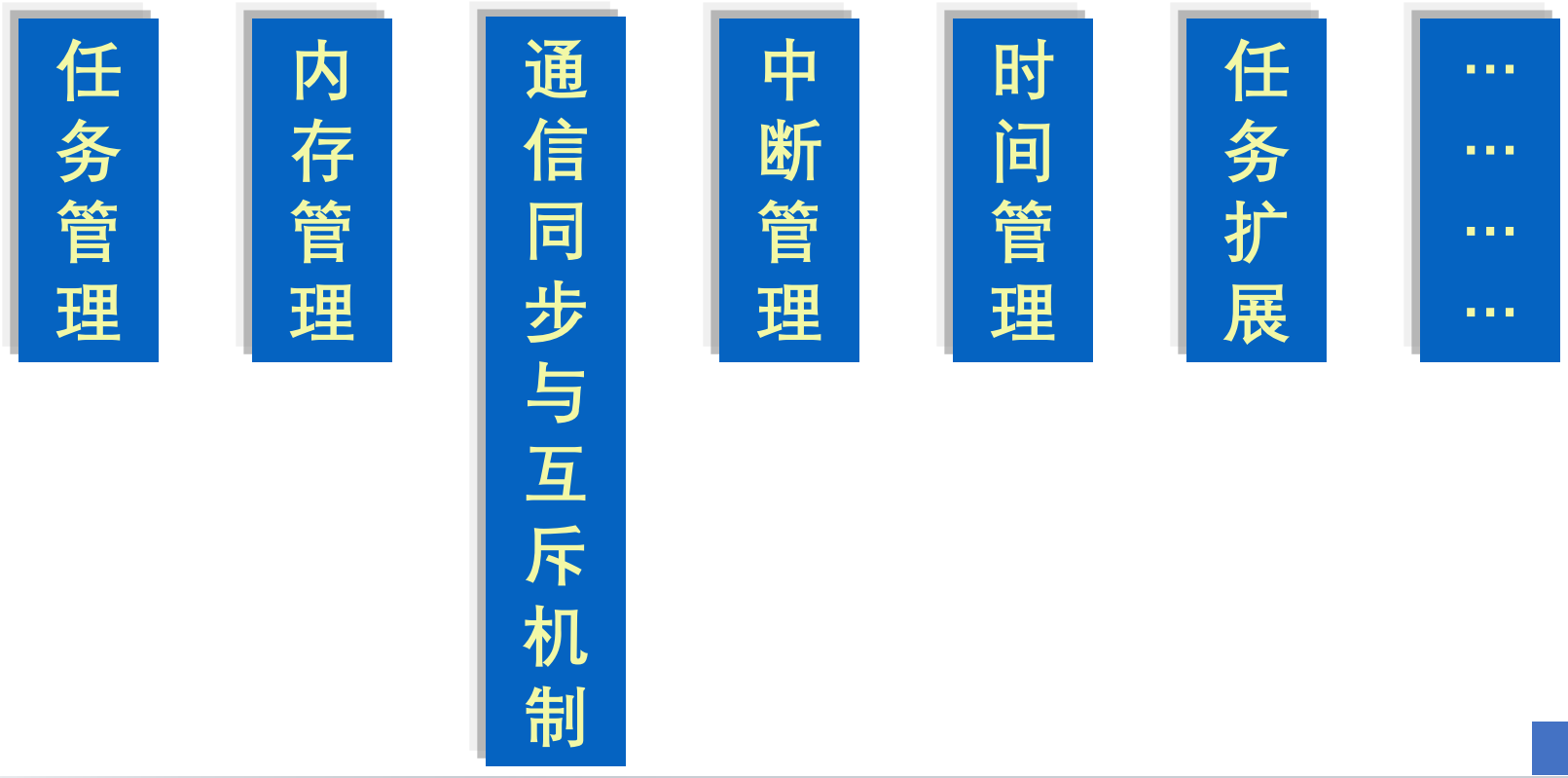
嵌入式操作系统分类

- **从应用领域来分类**
  - 面向信息家电的嵌入式操作系统
- **从实时性的角度来分类**
  - 嵌入式实时操作系统，具有强实时特点　如
- **从嵌入式系统的商业模式来分类**
  - 商用型：功能稳定、可靠，有完善的技术支持和售后服务，开发费用+版税
  - 开源型：开放源码，只收服务费，没有版税。如Embedded linux，RTEMS，eCOS

嵌入式操作系统的组成

嵌入式内核

- 内核是嵌入式操作系统的基础，也是必备的部分
- 内核还提供特定的应用编程接口，但目前没有统一的标准



嵌入式内核

- 任务管理
  - 内核的核心部分，具有任务调度、创建任务、删除任务、挂起任务、解挂任务、设置任务优先级等功能
  - 通用计算机的操作系统追求的是最大的吞吐率，为了达到最佳整体性能，其调度原则是公平，采用Round-Robin或可变优先级调度算法，调度时机主要以时间片为主驱动
  - 而嵌入式操作系统多采用基于静态优先级的可抢占的调度，任务优先级是在运行前通过某种策略静态分配好的，一旦有优先级更高的任务就绪就马上进行调度

嵌入式内核

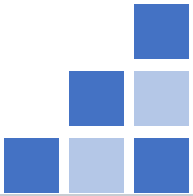
- 内存管理
  - 嵌入式操作系统的内存管理比较简单
  - 通常不采用虚拟存储管理，而采用静态内存分配和动态内存分配（固定大小内存分配和可变大小内存分配）相结合的管理方式。
  - 有些内核利用MMU机制提供内存保护功能
  - 通用操作系统广泛使用了虚拟内存的技术，为用户提供一个功能强大的虚存管理机制

嵌入式内核

- 通信、同步和互斥机制
  - 这些机制提供任务间、任务与中断处理程序间的通信、同步和互斥功能
  - 一般包括信号量、消息、事件、管道、异步信号和共享内存等功能
  - 与通用操作系统不同的是，嵌入式操作系统需要解决在这些机制的使用中出现的优先级反转问题

嵌入式内核

- 中断管理，一般具有以下功能：
  - 安装中断服务程序
  - 中断发生时，对中断现场进行保存，并且转到相应的服务程序上执行
  - 中断退出前，对中断现场进行恢复
  - 中断栈切换
  - 中断退出时的任务调度

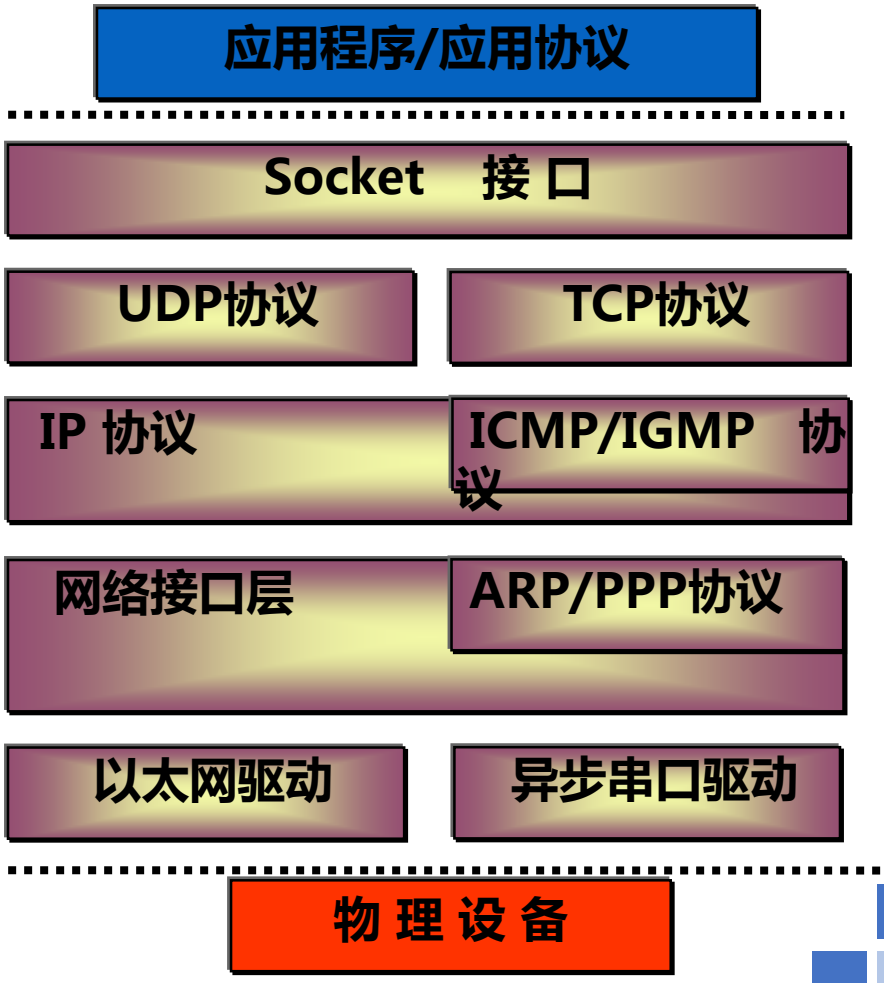


嵌入式内核

- 时间管理
  - 提供高精度、应用可设置的系统时钟，该时钟是嵌入式系统的时基，可设置为十毫秒以下
  - 提供日历时间，负责与时间相关的任务管理工作如任务对资源有限等待的计时、时间片轮转调度等，提供软定时器的管理功能等
  - 通用操作系统的系统时钟的精度由操作系统确定，应用不可调，且一般是几十个毫秒

嵌入式TCP/IP

- TCP/IP协议已经广泛地应用于嵌入式系统中
- 嵌入式TCP/IP网络系统提供符合TCP/IP协议标准的协议栈，提供Socket编程接口



嵌入式TCP/IP

- 嵌入式TCP/IP网络系统具有以下的特点：
  - 可剪裁

    能根据嵌入式系统的功能的要求选择所需的协议，对完整的TCP/IP协议簇进行剪裁，以满足用户的需要。
  - 采用“零拷贝”（Zero Copy）技术，提高实时性

    TCP/IP协议栈没有用于各层间数据传递的缓冲区，协议栈各层间传递的都是数据指针，只有当数据最终要被驱动程序发送出去或是被应用程序取走时，才进行真正的数据搬移

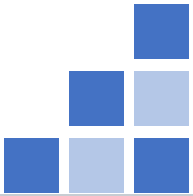
嵌入式TCP/IP

- 采用静态分配技术

网络初始化时就静态分配通信缓冲区，设置了专门的发送和接收缓冲（其大小一般小于或等于物理网络上的MTU值），从而确保了每次发送或接收时处理的数据不会超过MTU值，也就避免了数据处理任务的阻塞等待

嵌入式文件系统

- 通用操作系统的文件系统通常具有以下功能：
  - 提供用户对文件操作的命令
  - 提供用户共享文件的机制
  - 管理文件的存储介质
  - 提供文件存取控制机制，保障文件及文件系统的安全性
  - 提供文件及文件系统的备份和恢复功能
  - 提供对文件的加密和解密功能



嵌入式文件系统

- 嵌入式文件系统相比之下较为简单，主要具有文件的存储、检索、更新等功能，一般不提供保护和加密等安全机制
- 它以系统调用和命令方式提供对文件的各种操作
  - 设置和修改对文件和目录的存取权限
  - 提供建立、修改、改变、删除目录等服务
  - 提供创建、打开、读、写、关闭、撤消文件等服务

嵌入式实时操作系统μC/OS-II简介

- μC/OS-II是一个抢占式实时多任务内核。它是用ANSI的C语言编写的，包含一小部分汇编语言代码，使之可以提供给不同架构的微处理器使用
- 至今，从8位到64位，μC/OS-II已经在40多种不同架构的微处理器上使用
- 使用μC/OS的领域包括：照相机行业、航空业、医疗器械、网络设备、自动提款机以及工业机器人等

嵌入式实时操作系统μC/OS-II简介

- μC/OS-II全部以源代码的方式提供，大约有5500行
- CPU相关的部分使用的是针对Intel80x86微处理器的代码
- μC/OS-II可以很容易地移植到不同架构的嵌入式微处理器上

嵌入式实时操作系统μC/OS-II简介

- μC/OS-II的特点：
  - 源代码
  - 可移植
  - 可固化
  - 可裁减
  - 可抢占性
  - 支持多任务
    - 可确定性
    - 任务栈
    - 系统服务
    - 中断管理
    - 稳定性和可靠性

嵌入式实时操作系统μC/OS-II简介

- 源代码文件介绍
  - 对函数和环境的定义： **PC.C**
  - 与处理器类型无关部分：

    OS_CORE.C　OS_FLAG.C　OS_MBOX.C　OS_MEM.C
    OS_MUTEX.C　OS_Q.C　OS_SEM.C　OS_TASK.C
    OS_TIME.C　μCOS-II.C　μCOS-II.H
  - 与处理器类型相关部分：

    OS_CPU_A.S　OS_CPU_C.C　OS_CPU.H
  - 给整个内核库提供总体的include文件：**INCLUDES.H**
  - 配置文件，定义使用μC/OS-II中的哪些功能：**OS_CFG.H**

嵌入式实时操作系统μC/OS-II简介

μC/OS-II不是自由软件，
用于商业目的时须取得许可证

第三节

嵌入式软件开发工具

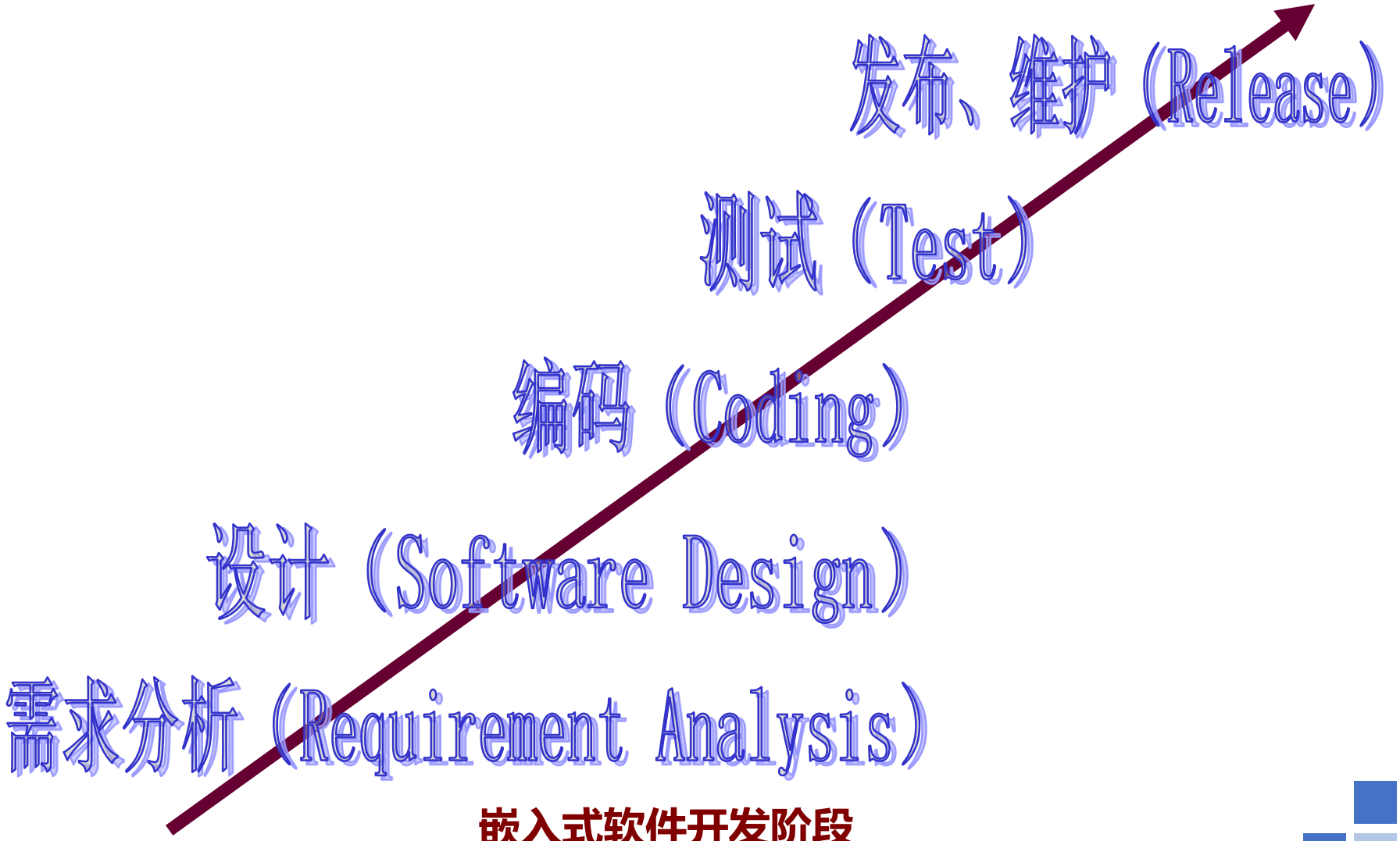
嵌入式软件开发工具的分类
嵌入式软件的交叉开发环境
嵌入式软件实现阶段的开发过程
嵌入式软件开发工具的发展趋势

嵌入式软件开发工具

“工欲善其事，必先利其器”

嵌入式软件开发工具的集成度和可用性将直接关系到嵌入式系统的开发效率

嵌入式软件开发工具的分类

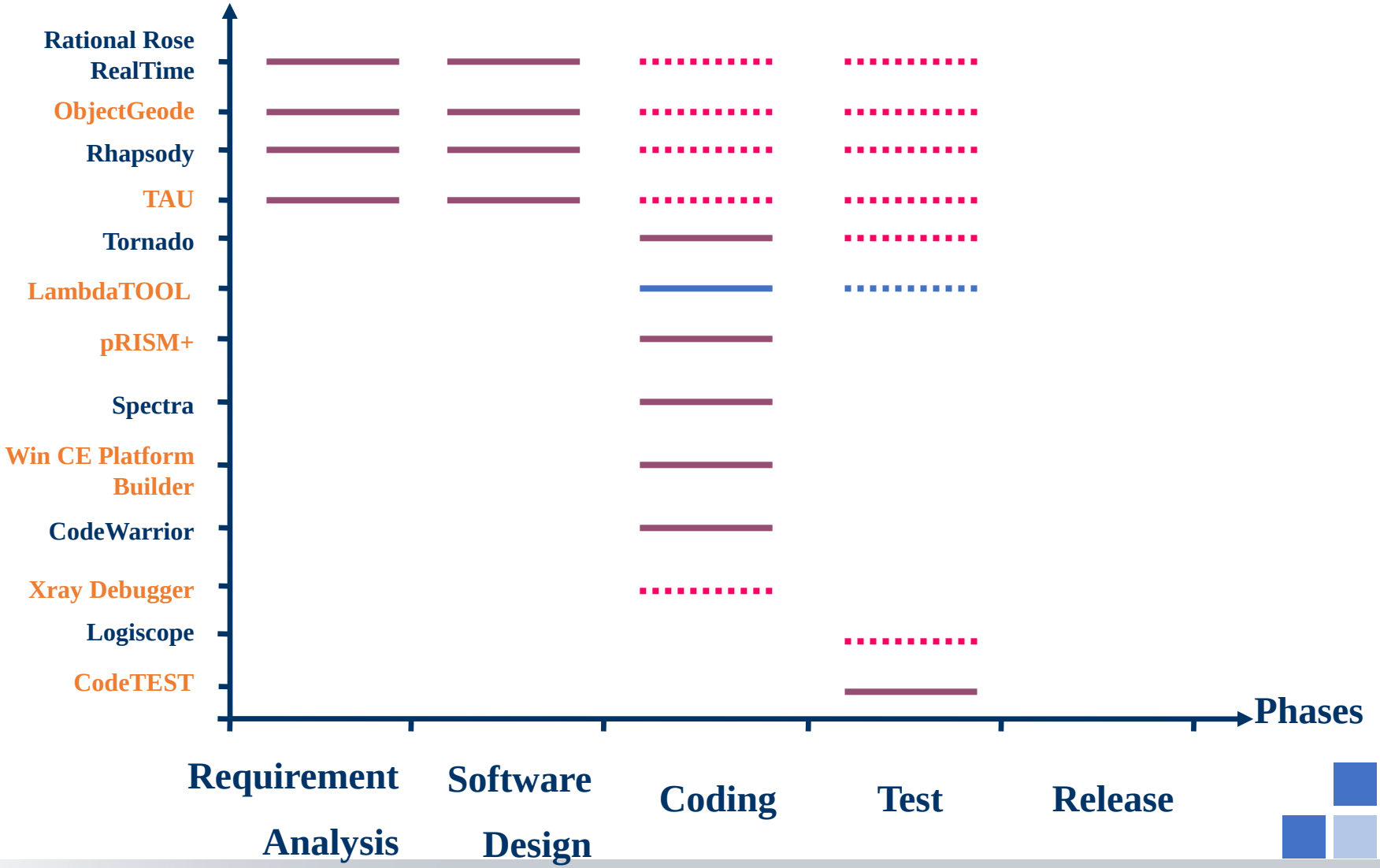


嵌入式软件开发阶段

嵌入式软件开发工具的分类

- 根据不同的阶段，嵌入式软件开发工具可以分为：
  - 需求分析工具（Requirement Analysis Tools）
  - 软件设计工具（ Software Design Tools ）
  - 编码、调试工具（ Coding Tools ）
  - 测试工具（ Testing Tools ）
  - 配置管理工具、维护工具等

主要嵌入式软件开发工具产品



嵌入式软件开发工具的分类

- 嵌入式软件的开发可以分为以下几种
  - 编写简单的板级测试软件，主要是辅助硬件调试
  - 开发基本的驱动程序
  - 开发特定嵌入式操作系统的驱动程序（板级支持包）
  - 开发嵌入式系统软件，如：嵌入式操作系统等
  - 开发应用软件

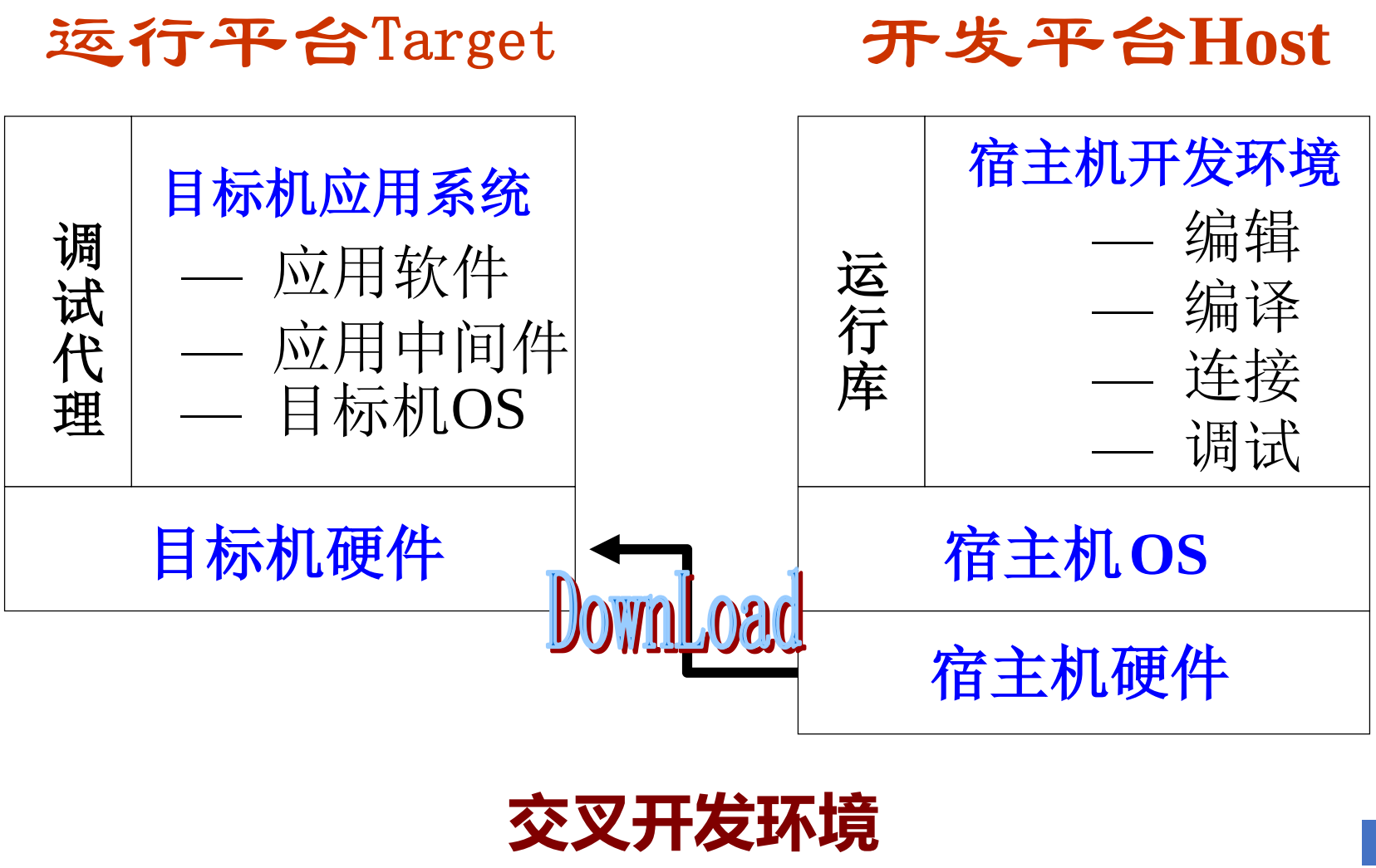
嵌入式软件开发工具的分类

- 从以上嵌入式软件开发分类来看，嵌入式软件开发工具可以分为
  - 与嵌入式OS相关的开发工具，用于开发
    - 基于嵌入式OS的应用
    - 部分驱动程序等
  - 与嵌入式OS无关的开发工具，用于开发
    - 基本的驱动程序
    - 辅助硬件调试程序
    - 系统软件等

嵌入式软件的交叉开发环境

- 交叉开发环境是指用于嵌入式软件开发的所有工具软件的集合，一般包括
  - 文本编辑器
  - 交叉编译器
  - 交叉调试器
  - 仿真器
  - 下载器等
- 交叉开发环境由宿主机和目标机组成，宿主机与目标机之间在物理连接的基础上建立起逻辑连接

嵌入式软件的交叉开发环境



交叉开发环境

嵌入式软件的交叉开发环境

- 宿主机（Host）：是用于开发嵌入式系统的计算机。一般为PC机（或者工作站），具备丰富的软硬件资源，为嵌入式软件的开发提供全过程支持
- 目标机（Target）：即所开发的嵌入式系统，是嵌入式软件的运行环境，其硬件软件是为特定应用定制的
- 在开发过程中，目标机端需接收和执行宿主机发出的各种命令如设置断点、读内存、写内存等，将结果返回给宿主机，配合宿主机各方面的工作

嵌入式软件的交叉开发环境

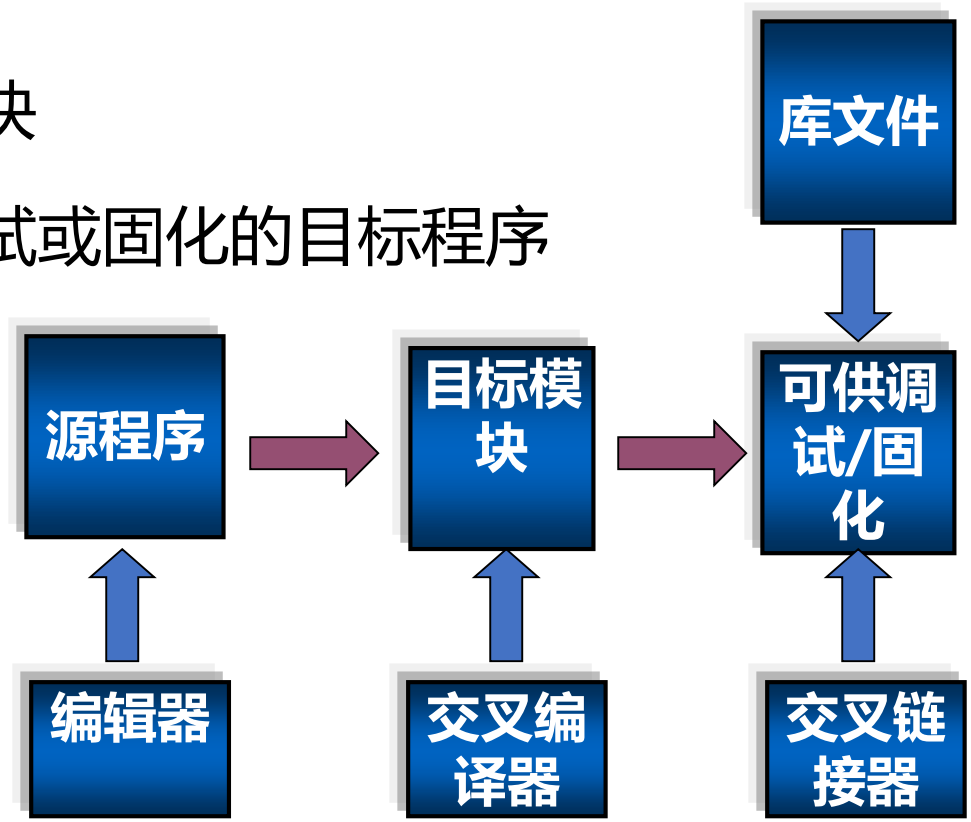
- 物理连接和逻辑连接
  - **物理连接**是指宿主机与目标机通过物理线路连接在一起，连接方式主要有三种：
    - 串口
    - 以太口
    - OCD（On Chip Debug）方式，如JTAG、BDM等
  - 物理连接是逻辑连接的基础
  - **逻辑连接**指宿主机与目标机间按某种通信协议建立起来的通信连接，已逐步形成一些通信协议标准

嵌入式软件实现阶段的开发过程

- 设计完成后，嵌入式软件的开发进入实现阶段，可分为三个步骤：生成、调试和固化运行
  - **软件的生成：**在宿主机上进行，利用各种工具完成应用程序的编辑、交叉编译和链接工作，生成可供调试或固化的目标程序
  - **调试：**通过交叉调试器完成软件的调试工作。调试完成后还需进行必要的测试工作
  - **固化运行：**先用一定的工具将应用程序固化到目标机上，然后启动目标机，在没有任何工具干预的情况下应用程序能自动地启动运行

嵌入式软件生成阶段

- 三个过程
  - 源代码程序的编写
  - 编译成各个目标模块
  - 链接成可供下载调试或固化的目标程序



交叉编译

- 把在宿主机上编写的高级语言程序编译成可以运行在目标机上的代码，即在宿主机上能够编译生成另一种CPU（嵌入式微处理器）上的二进制程序

嵌入式软件的调试

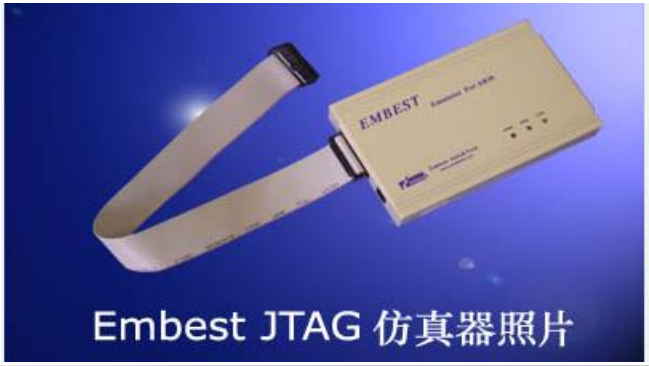
- 交叉调试器
  - 是指调试程序和被调试程序运行在不同机器上的调试器
  - 调试器通过某种方式能控制目标机上被调试程序的运行方式
  - 通过调试器能查看和修改目标机上的内存、寄存器以及被调试程序中的变量等

几种常用的调试方法

- 指令集模拟器：一种利用PC机端的仿真开发软件模拟调试的方法
- 驻留监控软件：驻留监控程序运行在目标板上，PC机端调试软件可通过并口、串口、网口与之交互，以完成程序执行、存储器及寄存器读写、断点设置等任务
- JTAG仿真器：通过ARM芯片的JTAG边界扫描口与ARM核进行通信，不占用目标板的资源，是目前使用最广泛的调试手段
- 在线仿真器(In-Circuit Emulator，ICE)：使用仿真头代替目标板上的CPU，可以完全仿真ARM芯片的行为。但结构较复杂，价格昂贵，通常用于ARM硬件开发中

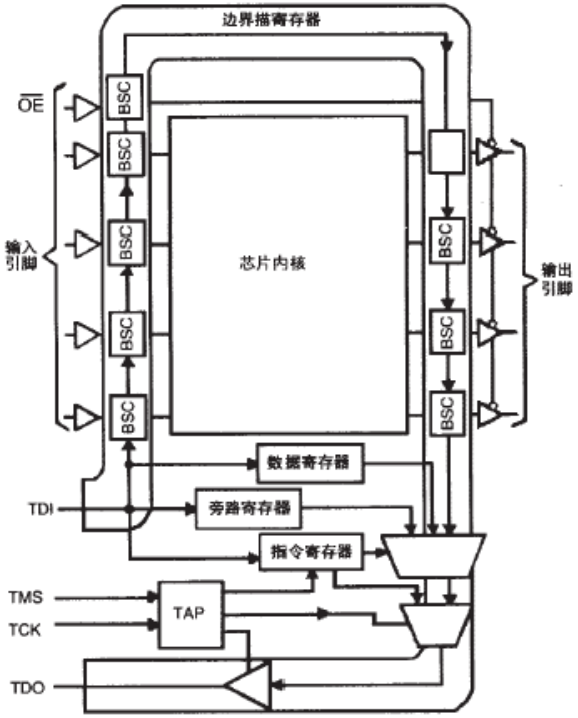
各种JTAG仿真器实例

VITRA shown here with both debug and trace connections to ARM™ Integrator/CM-966E-S development board.



边界扫描技术（JTAG）

- JTAG——标准测试访问接口与边界扫描结构（Standard Test Access Port and Boundary Scan Architecture），已被IEEE1149.1标准所采纳，是面向用户的测试接口
- 该接口一般由4个引脚组成：
  - 测试数据输入（TDI）
  - 测试数据输出（TDO）
  - 测试时钟（TCK）
  - 测试模式选择引脚（TMS）
  - 异步测试复位引脚（TRST，可选）



边界扫描技术（JTAG）

- 优点
  - 可以通过边界扫描操作测试整个板的电气连接，特别为表面贴元件提供方便
  - 各个引脚信号的采样，并可强制引脚输出用以测试外围芯片
  - 可以软件下载、执行、调试和控制，为复杂的实时跟踪调试提供路径
  - 可以进行多内核和多处理器的板级和芯片级的调试，通过串接，为芯片制造商提供芯片生产、测试的途径
  - 不占用系统资源，能够调试没有外部总线的芯片，代价非常小

边界扫描技术（JTAG）

- 缺点
  - 通过串口依次传递数据，速度比较慢
  - 只能进行软件断点级别的调试
  - 不能完成实时跟踪和多种事件触发等复杂调试功能
- 几种增强版本
  - ARM芯片的实时调试方案（E-TRACE）
  - 背景调试模式BDM
  - 片上仿真OnCE

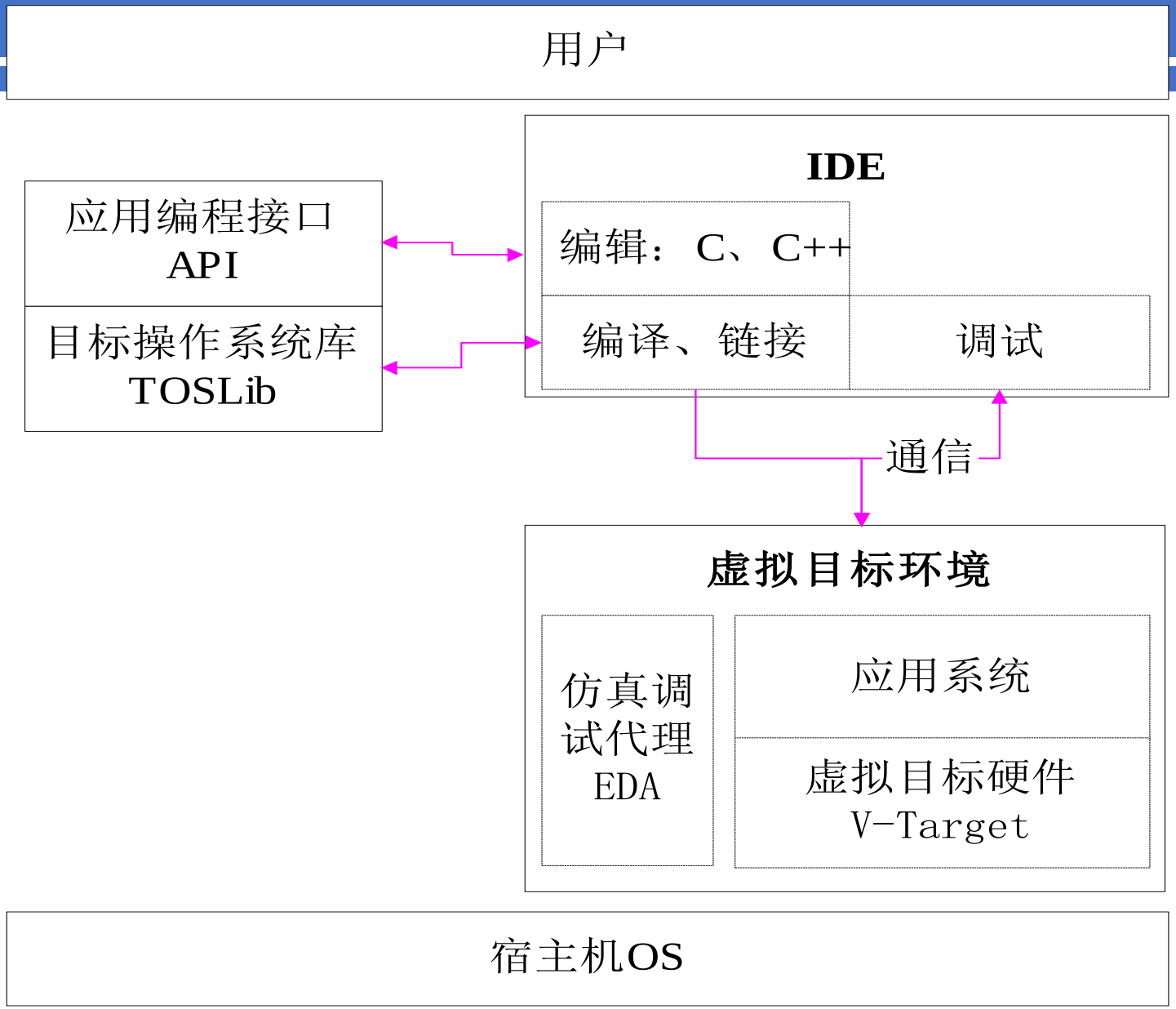
指令集模拟器 Simulator

交叉开发方式存在如下缺点：

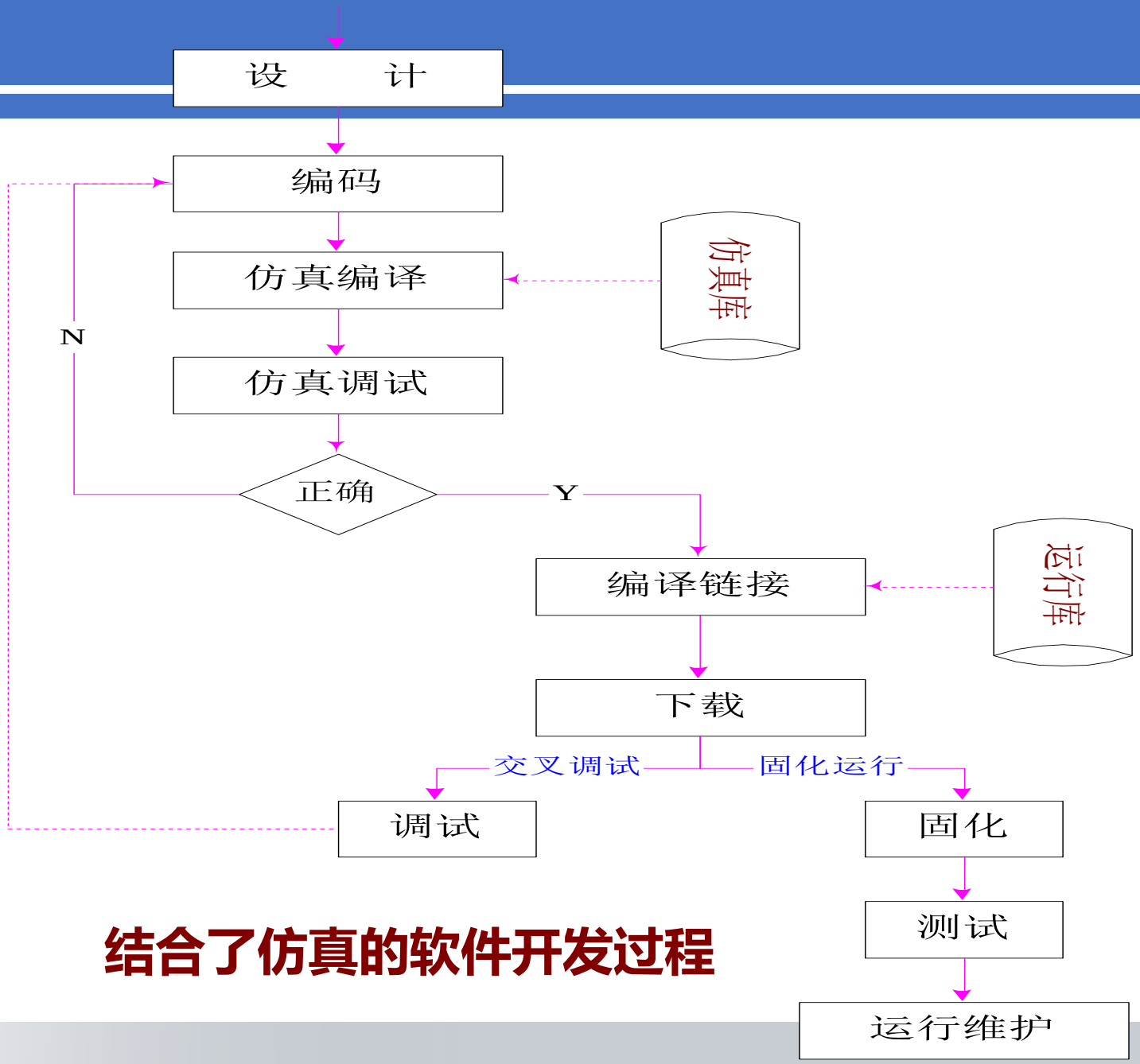
- 硬件支持　必须有目标机或评估板
- 易使用性　普通编程人员不熟悉
- 廉价性　成本高
- 可移植性、可扩展性　不高
- 团队开发　较难
- 开发周期　较长

指令集模拟器 Simulator

- 一种软件仿真器，在宿主机上创建一个虚拟的目标机环境，再将应用系统下载到这个虚拟目标机上运行 / 调试
- 软件仿真的对象
  - 仿真处理器
  - 仿真外设
  - 仿真环境
- 软件仿真的级别
  - 指令级仿真开发
  - API级仿真开发



应用仿真开发环境示意图



结合了仿真的软件开发过程

Simulator

- 优点

最大好处就是可以不用真正的目标机，可以在目标机环境并不存在的条件下开发目标机上的应用系统，并且在调试时可以利用Host资源提供更详细的错误诊断信息

Simulator

- 缺点
  1. 和实际的运行环境差别很大
  2. 设备模拟的局限性较大
  3. 实时特性较差
  4. 对Host的资源要求较高
- 适用范围

  对时间特性没有严格要求、没有特殊外设、只需要验证逻辑正确的应用程序

嵌入式软件的测试

- 测试工具：用来辅助测试的工具，主要用来支持测试人员的工作，本身不能直接用来进行测试。测试工具一般都是通用工具，测试人员应根据实际情况对它们进行适当调整
- 嵌入式软件测试中经常用到的测试工具
  - 内存分析工具
  - 性能分析工具
  - 覆盖分析工具
  - 缺陷跟踪工具等

嵌入式软件的固化运行

- 当调试完成之后，程序代码需要被完全烧入到目标板的非易失性存储器（如ROM或闪存）中，并且在真实的硬件环境上运行，这个过程叫做固化
- 调试环境与固化环境的区别
  - **代码定位不同**
  - **初始化部分不同**

嵌入式软件的固化运行

阶段	调试环境	固化环境
编译	目标文件需要调试信息	目标文件不需要调试信息
链接	应用系统目标代码不需要Boot模块，此模块已由目标板上的监控器程序实现	应用系统目标代码必须以Boot模块作为入口模块
定位	程序的所有代码段、数据段都依次被定位到调试空间的RAM中	程序的各逻辑段按照其不同的属性分别定位到非易失性存储空间（ROM）或RAM中
下载	宿主机上的调试器读入被调试文件，并将其下载到目标机上的调试空间中，目标机掉电后所有信息全部丢失	在宿主机上利用固化工具将可固化的应用程序写入目标机的非易失性存储器，目标机掉电后信息不丢失
运行	被调试程序在目标监控器的控制下运行，并与后者共享某些资源，如CPU资源、RAM资源以及通信设备（如串口、网口等）资源	程序在真实的目标硬件环境上运行

嵌入式软件的固化运行

- Bootloader：当应用程序在真实的目标环境下运行时将首先执行该程序，它至少由系统加电时执行的代码组成
- Bootloader的主要功能：初始化CPU环境，使目标机硬件到已知的状态
  - 初始化芯片的引脚
  - 初始化系统外部控制寄存器
  - 初始化基本输入输出设备
  - 初始化MMU，包括片选控制寄存器等
  - 执行数据拷贝

嵌入式软件开发工具发展趋势

- 向着开放的、集成化的方向发展
- 具有系统设计、可视化建模、仿真和验证功能
- 自动生成代码和文档
- 具有更高的灵活性

Thank you

嵌入式软件的测试

内存分析工具

- 嵌入式系统的内存资源通常是受限的，内存分析工具可以用来处理在进行动态内存分配时产生的缺陷。当动态分配的内存被错误地引用时，产生的错误通常难以再现，出现的失效难以追踪，使用内存分析工具可以很好地检测出这类缺陷。
- 目前常用的内存分析工具有软件和硬件两种：
  - **基于软件的内存分析工具可能会对代码的执行性能带来很大影响，从而影响系统的实时性；**
  - **基于硬件的内存分析工具对系统性能影响小，但价格昂贵，并且只能在特定的环境中使用。**

嵌入式软件的测试

性能分析工具

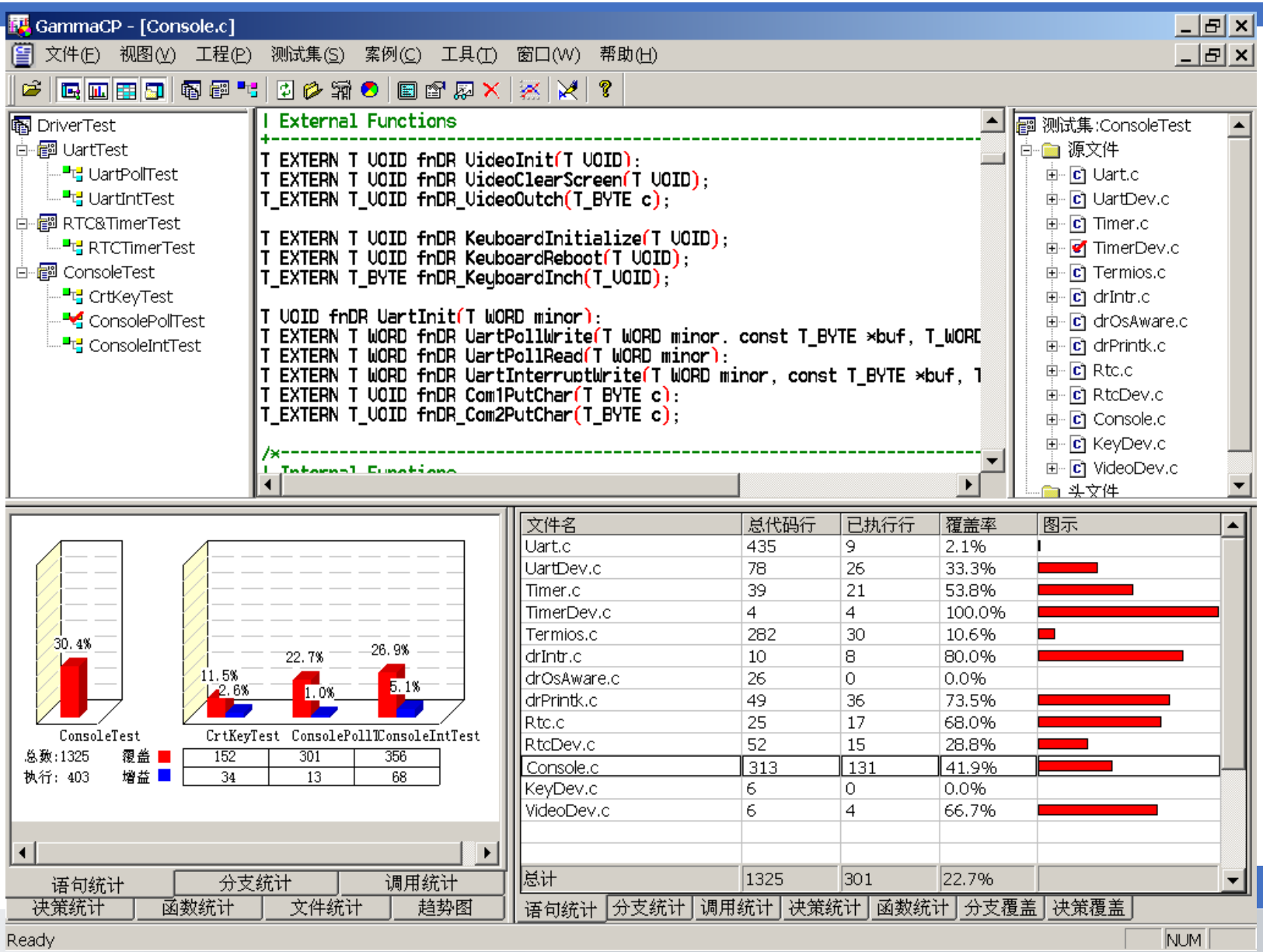
- 嵌入式系统的性能通常是一个非常关键的因素，开发人员一般需要对系统的某些关键代码进行优化来改进性能
- 性能分析工具
  - 可以提供有关数据，帮助确定哪些任务消耗了过多的执行时间，从而可以决定如何优化软件，以获得更好的时间性能
  - 引导开发人员发现在系统调用中存在的错误以及程序结构上的缺陷

嵌入式软件的测试

覆盖分析工具

- 在进行白盒测试时，可以使用代码覆盖分析工具追踪哪些代码被执行过
- 分析过程一般通过插桩来完成，插桩可以是在测试环境中嵌入硬件，也可以是在可执行代码中加入软件，或者是两者的结合
- 开发人员通过对分析结果进行总结，可以确定哪些代码被执行过，哪些代码被遗漏了
- 目前常用的覆盖分析工具一般都提供有关功能覆盖、分支覆盖、条件覆盖等信息

覆盖分析工具实例


GammaCP - [Console.c]
文件(F) 视图(V) 工程(P) 测试集(S) 案例(C) 工具(T) 窗口(W) 帮助(H)
DriverTest
UartTest
UartPollTest
UartIntTest
RTC&TimerTest
RTCTimerTest
ConsoleTest
CrtKeyTest
ConsolePollTest
ConsoleIntTest
| External Functions
T_EXTERN T_VOID fnDR_VideoInit(T_VOID);
T_EXTERN T_VOID fnDR_VideoClearScreen(T_VOID);
T_EXTERN T_VOID fnDR_VideoOutch(T_BYTE c);
T_EXTERN T_VOID fnDR_KeuboardInitialize(T_VOID);
T_EXTERN T_VOID fnDR_KeuboardReboot(T_VOID);
T_EXTERN T_BYTE fnDR_KeyboardInch(T_VOID);
T_VOID fnDR_UartInit(T_WORD minor);
T_EXTERN T_WORD fnDR_UartPollWrite(T_WORD minor, const T_BYTE *buf, T_WORD
T_EXTERN T_WORD fnDR_UartPollRead(T_WORD minor);
T_EXTERN T_WORD fnDR_UartInterruptWrite(T_WORD minor, const T_BYTE *buf, T
T_EXTERN T_VOID fnDR_Com1PutChar(T_BYTE c);
T_EXTERN T_VOID fnDR_Com2PutChar(T_BYTE c);
测试集:ConsoleTest
源文件
Uart.c
UartDev.c
Timer.c
TimerDev.c
Termios.c
drIntr.c
drOsAware.c
drPrintk.c
Rtc.c
RtcDev.c
Console.c
KeyDev.c
VideoDev.c
头文件
30.4%
11.5%
2.6%
22.7%
1.0%
26.9%
5.1%
ConsoleTest
CrtKeyTest
ConsolePollTest
ConsoleIntTest
总数:1325
执行: 403
覆盖
增益
152
301
356
34
13
68
文件名 总代码行 已执行行 覆盖率 图示
Uart.c 435 9 2.1%
UartDev.c 78 26 33.3%
Timer.c 39 21 53.8%
TimerDev.c 4 4 100.0%
Termios.c 282 30 10.6%
drIntr.c 10 8 80.0%
drOsAware.c 26 0 0.0%
drPrintk.c 49 36 73.5%
Rtc.c 25 17 68.0%
RtcDev.c 52 15 28.8%
Console.c 313 131 41.9%
KeyDev.c 6 0 0.0%
VideoDev.c 6 4 66.7%
总计 1325 301 22.7%
语句统计
分支统计
调用统计
决策统计
函数统计
文件统计
趋势图
语句统计
分支统计
调用统计
决策统计
函数统计
分支覆盖
决策覆盖
Ready
NUM

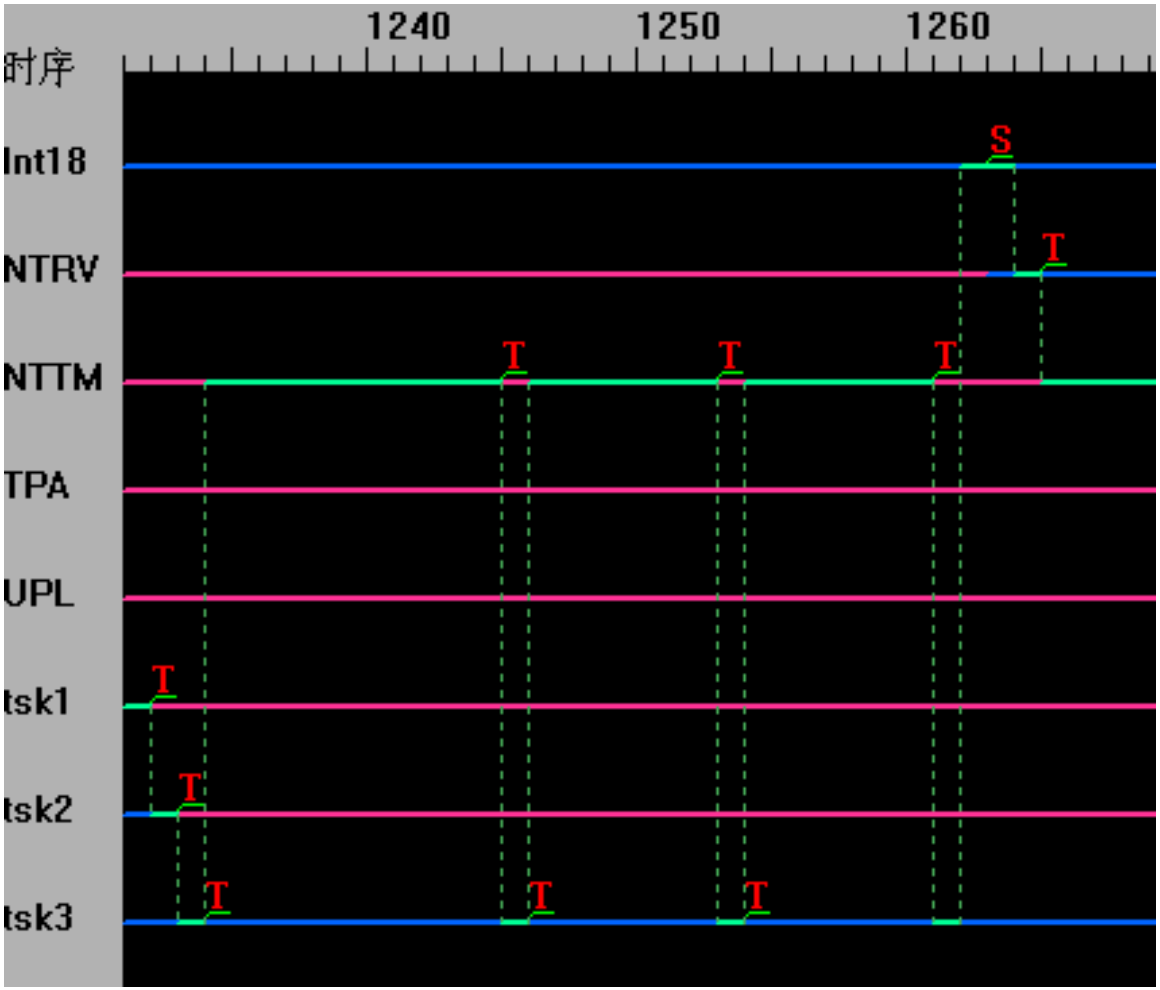
测试工具实例：逻辑分析仪

- 工作机理：在不打断被测程序运行流程的基础上，对程序运行中的相关信息进行采集和分析，然后通过真实再现程序运行的逻辑流程和分析程序运行数据，帮助用户优化系统设计和解决出现的问题
- 与调试工具的对比
  - **调试器：照相机**
  - **逻辑分析仪：摄像机**

测试工具实例：逻辑分析仪

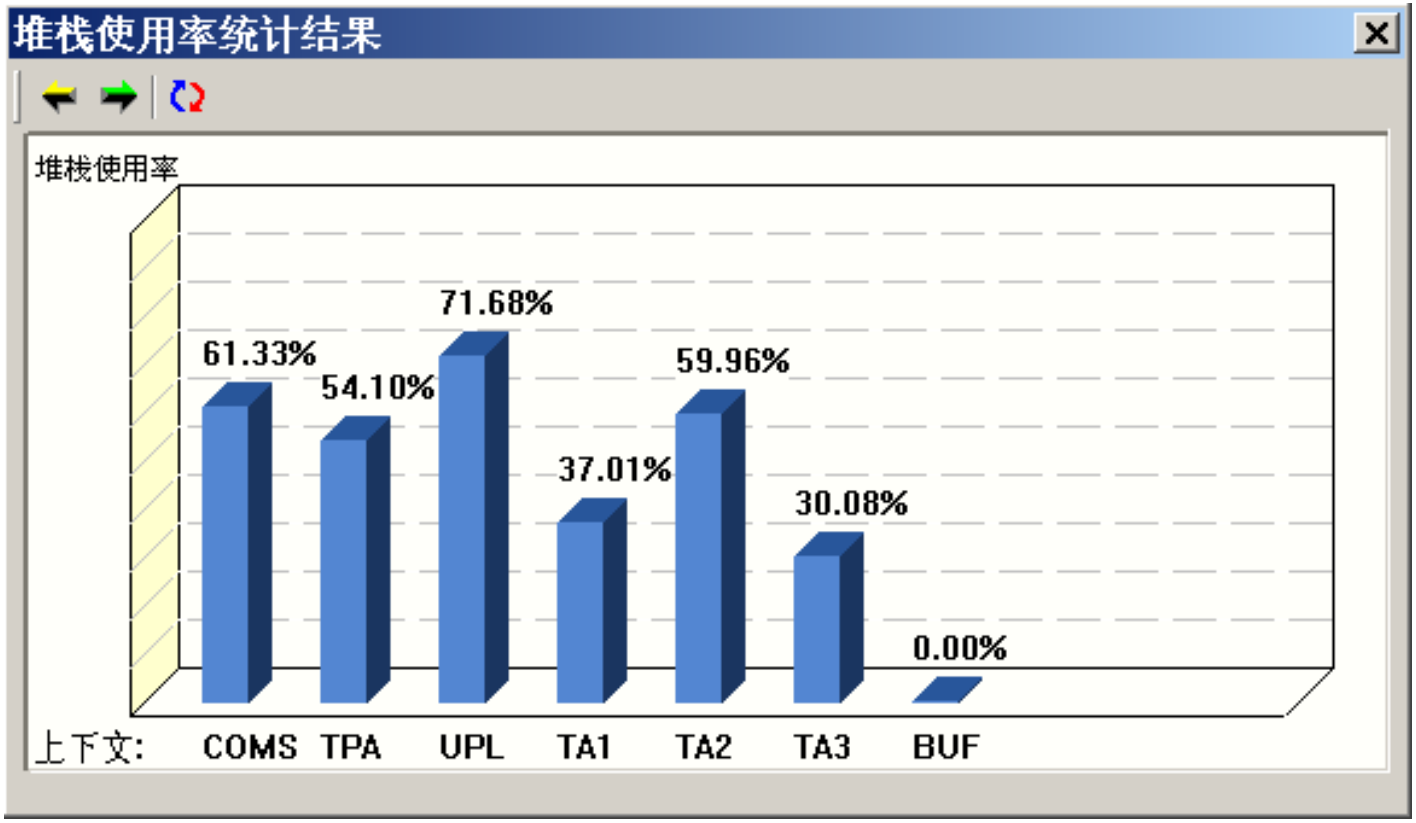
- 主要功能：
  - 真实再现程序运行流程
  - 发现系统死锁及软件造成的死机
  - 发现系统内存泄漏
  - 指导对任务的合理划分
  - 指导关键路径设计与验证
  - 指导合理分配任务堆栈
  - CPU使用率统计
  - 指导合理设计中断服务程序

测试工具实例：逻辑分析仪



嵌入式应用软件运行的逻辑流程

测试工具实例：逻辑分析仪



系统堆栈使用率分析