UNIVERSITY OF MICROCHIP
UOFM
MASTERS 2007
第八届中国技术精英年会

11031 FRT

如何在 Microchip 16位单片机上实现FreeRTOS.org

简介

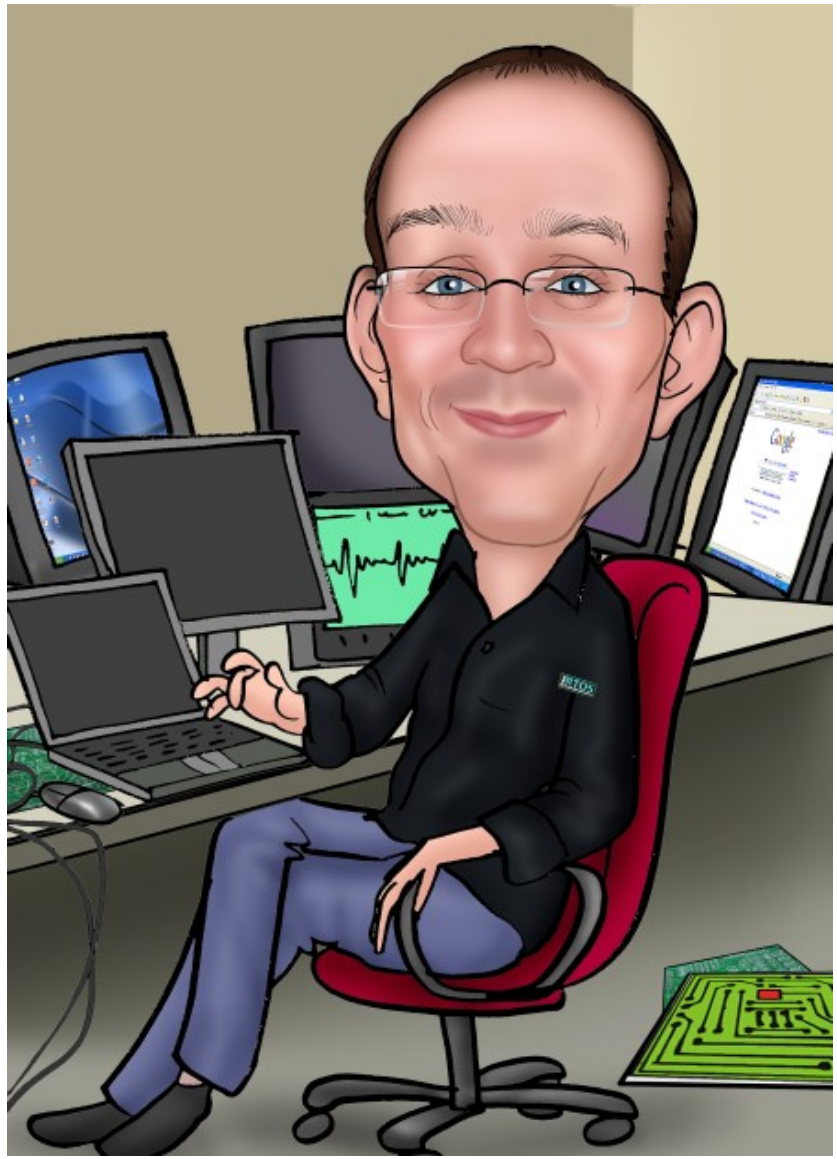
- **Richard Barry**

WITTENSTEIN high integrity systems

www.HighIntegritySystems.com

freeRTOS™

www.FreeRTOS.org



学习目标

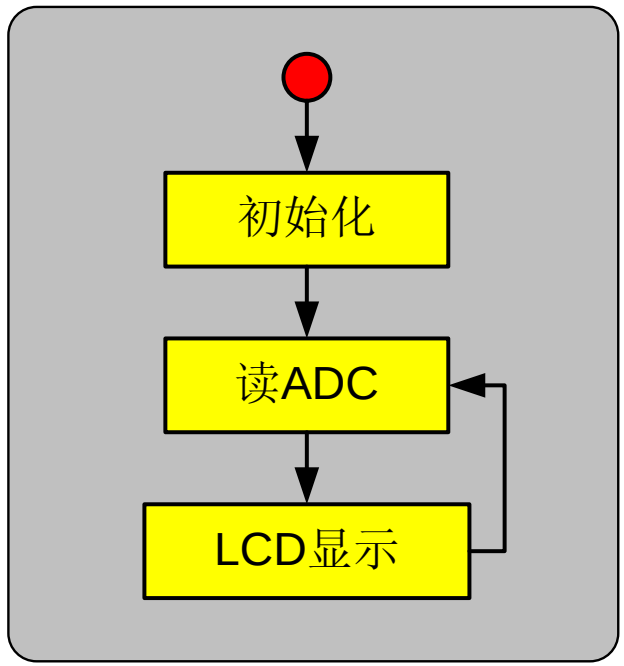
- 何为实时内核?
- 何时应使用实时内核?
- 在编写多任务应用程序时，用户应了解哪些内容?
- 如何开始使用实时内核?

课程安排

1. 单任务应用
   - 将传统架构扩展到更大的应用平台
   - Lab1 – 研究传统解决方法
2. 多任务基础知识
3. 其他概念及需要考虑的问题
4. 使用**FreeRTOS.org**实现一个多任务应用

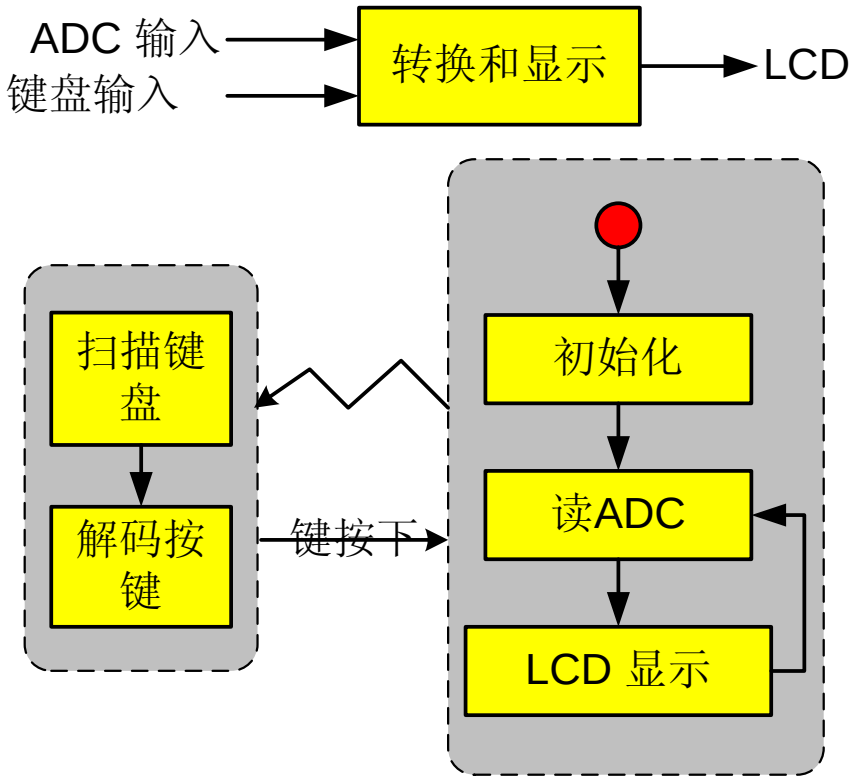
简单的连续处理应用

- 输入-处理-输出
- 上层-环路架构
- 自由运行或受控周期循环



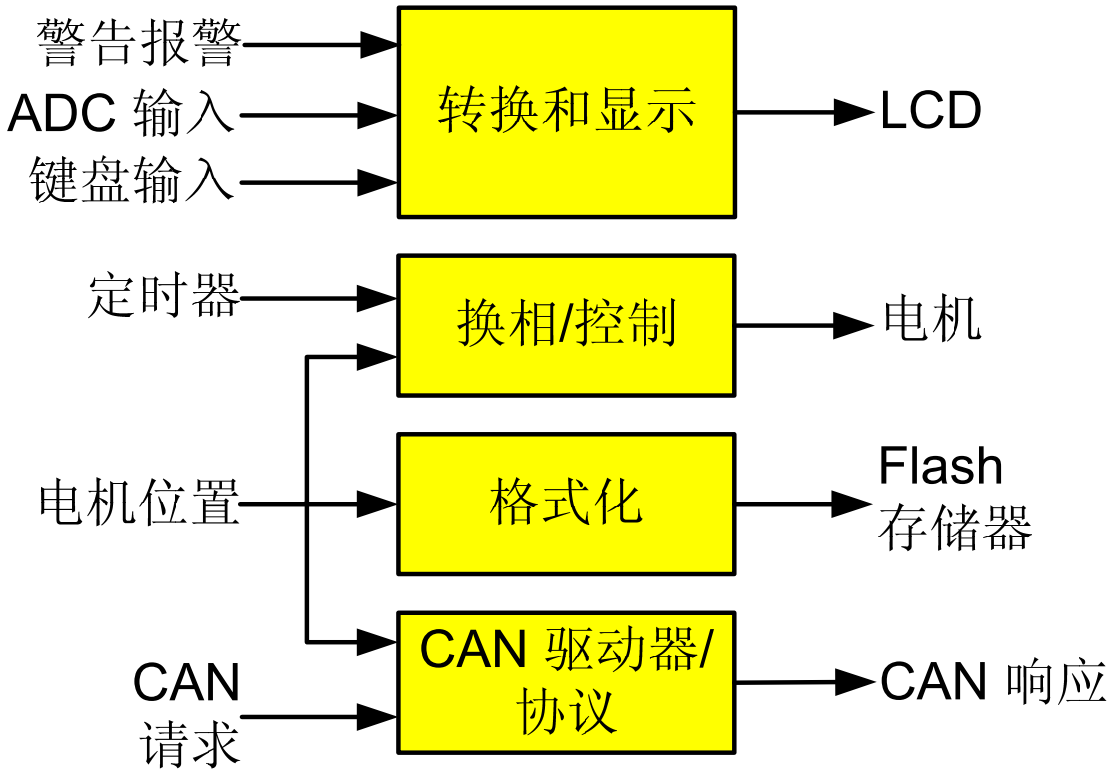
简单的前台/后台处理

- 前台上层环路
- 中断中进行的后台周期性处理



键盘扫描需要已知频率以实现较好的响应率和消抖性（**de-bouncing**）

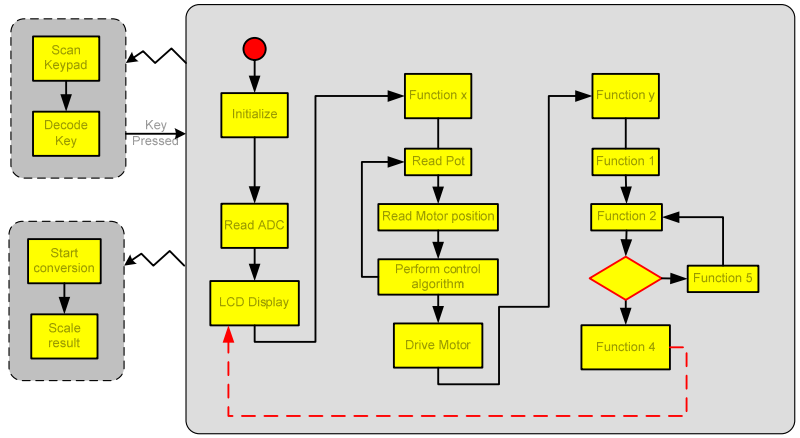
变得更为复杂



- 三个**LCD**报文源
- 包含快速和慢速硬件
- 包含连续性、周期性性、事件驱动、高优先级和低优先级的混合需求

修改后的设计

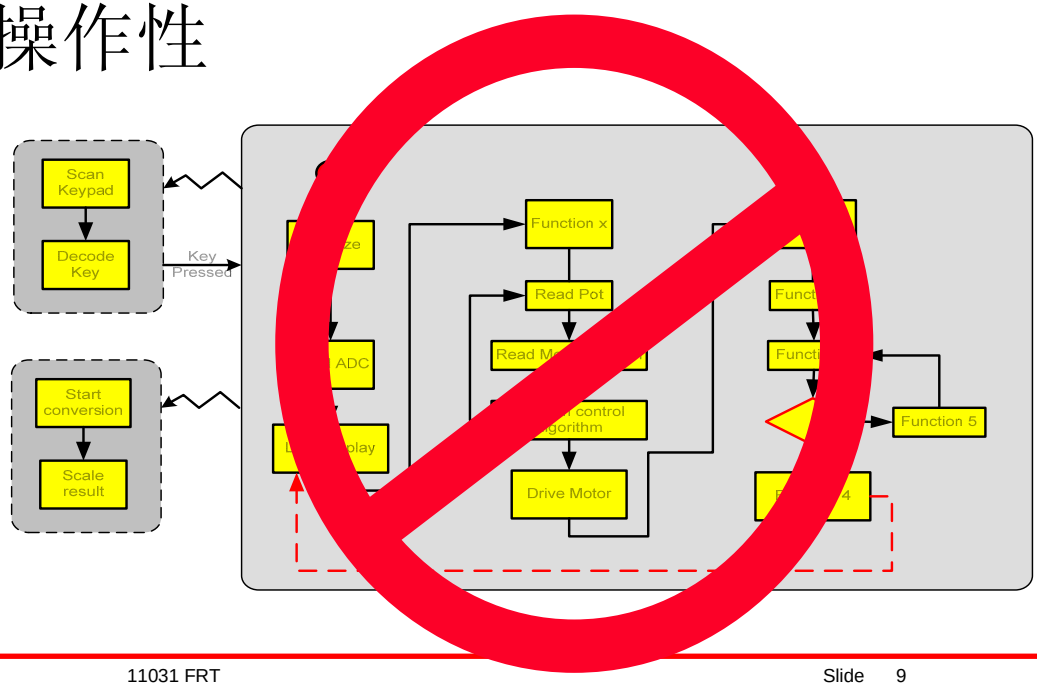
- 扩展架构
  - 中断中进行更多处理
  - 单个环路中对一个函数实现多次调用



- 它可以运行，但如何对它进行调整?
  - 市场部门刚发现我的新16位处理器具有很多剩余功能

潜在的问题是什么?

- 代码重用(实现过于依赖应用本身)
- COTS构件的可封装性
- 团队同时研发的可操作性
- 易测试性
- 确定性



Lab #1 目标

- 熟悉开发设置
  - 在我们深入学习RTOS之前
- 通过对一个非**RTOS**解决方案进行分析以对一些将要讨论的问题有充分了解
  - 随后我们将在下一部分采用基于FreeRTOS.org的方案进行解决

Lab #1 总结

回顾 **Lab #1** 中的问题

Part 1 结论: 可维护性

- 很难将时间的安排同功能性完全分割开来

COTS组件的集成	
代码的重用	
混合处理需求	
同时进行的团队研发	
可测试性	
混合有硬和软实时任务	
可扩展性 (不易受到应用方案出现改变的影响)	
允许硬件发生变化 (速度变化 – 尽管能更好的利用**h/w**定时器)	

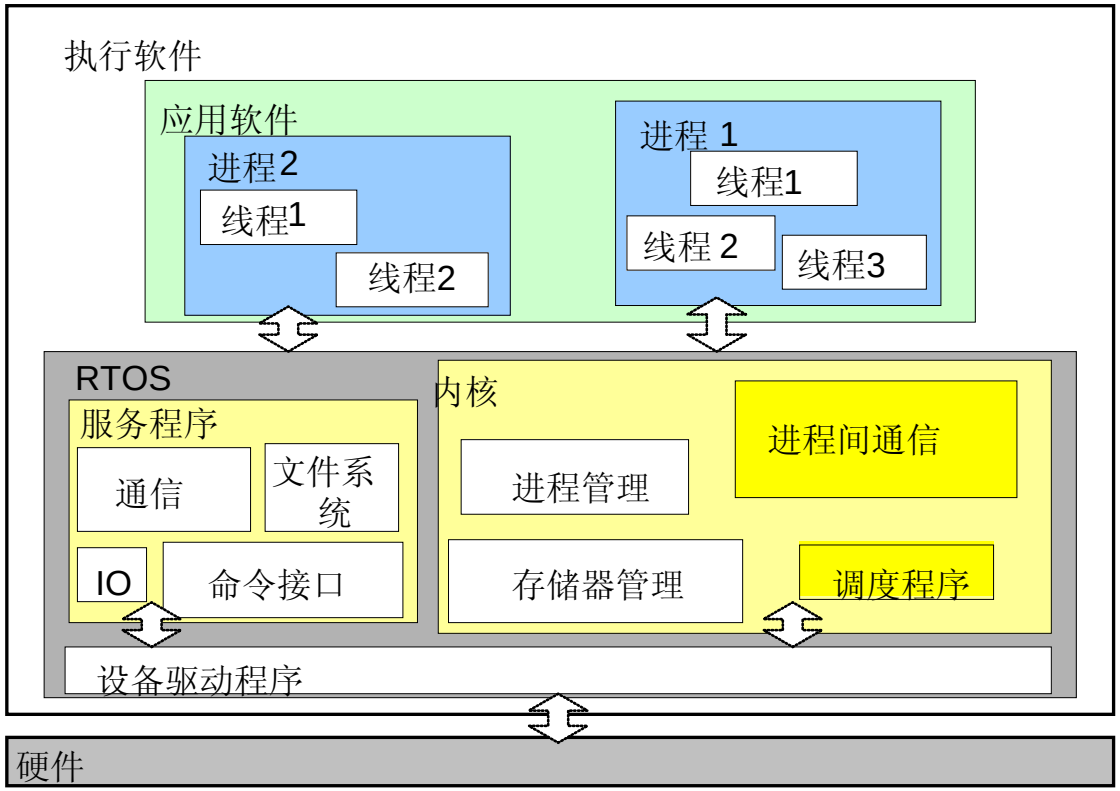
课程安排

1. 单任务应用
2. 多任务基础
   - 多任务范例
   - 实时内核的类型和特征
   - 任务和任务状态的特征
   - 将单一任务应用转换到多任务应用的示例
   - 多任务解决方案的利与弊
   - LAB #2 – 分析多任务解决方案
3. 其他的概念和需考虑的问题
4. 使用**FreeRTOS.org**创建一个多任务应用

多任务范例

- 多用户系统
  - Unix
  - VAX
- 现代台式计算机
  - Linux
  - Windows
  - MAC OS X

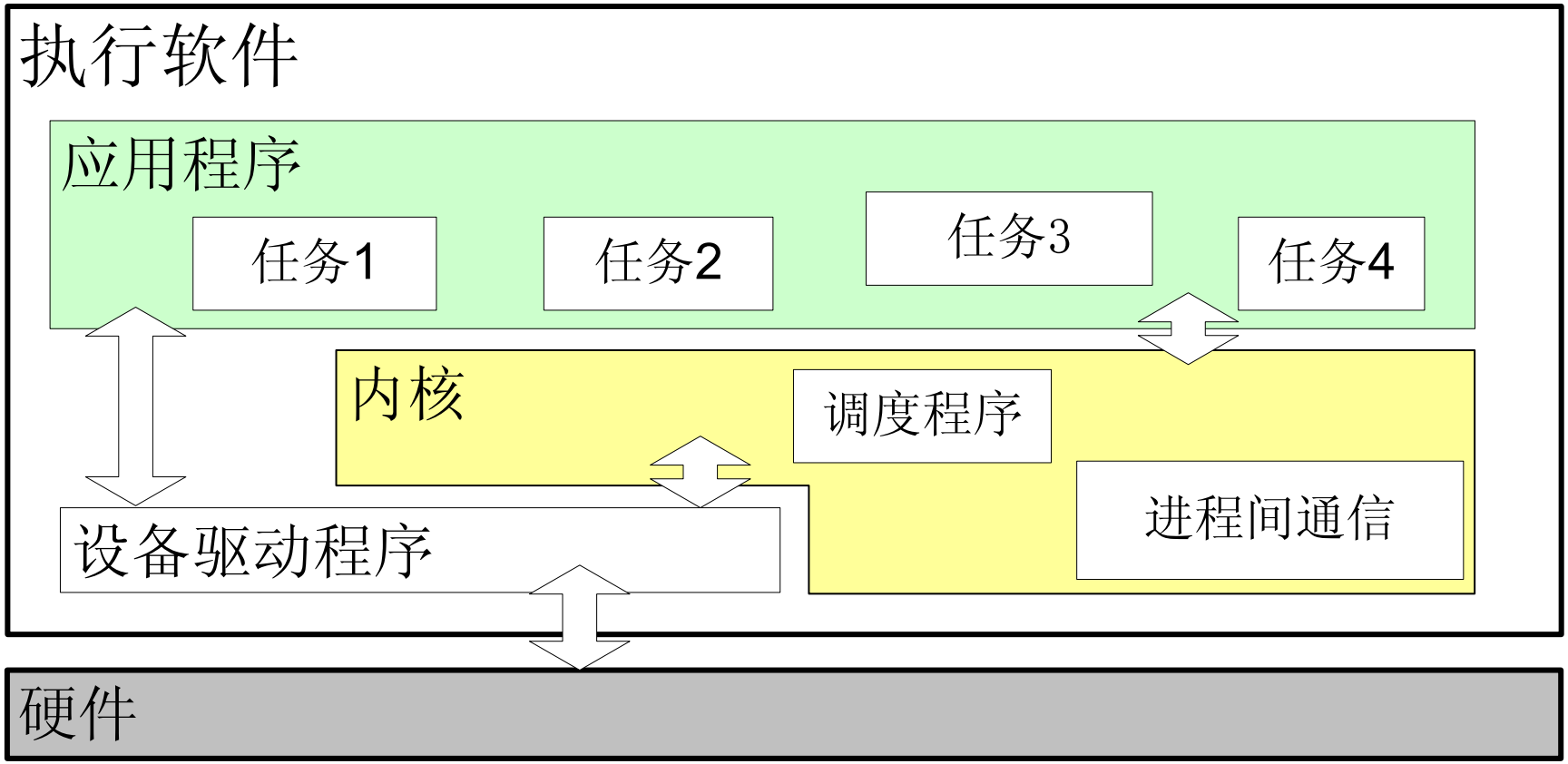
RTOS，内核还是调度程序?



术语

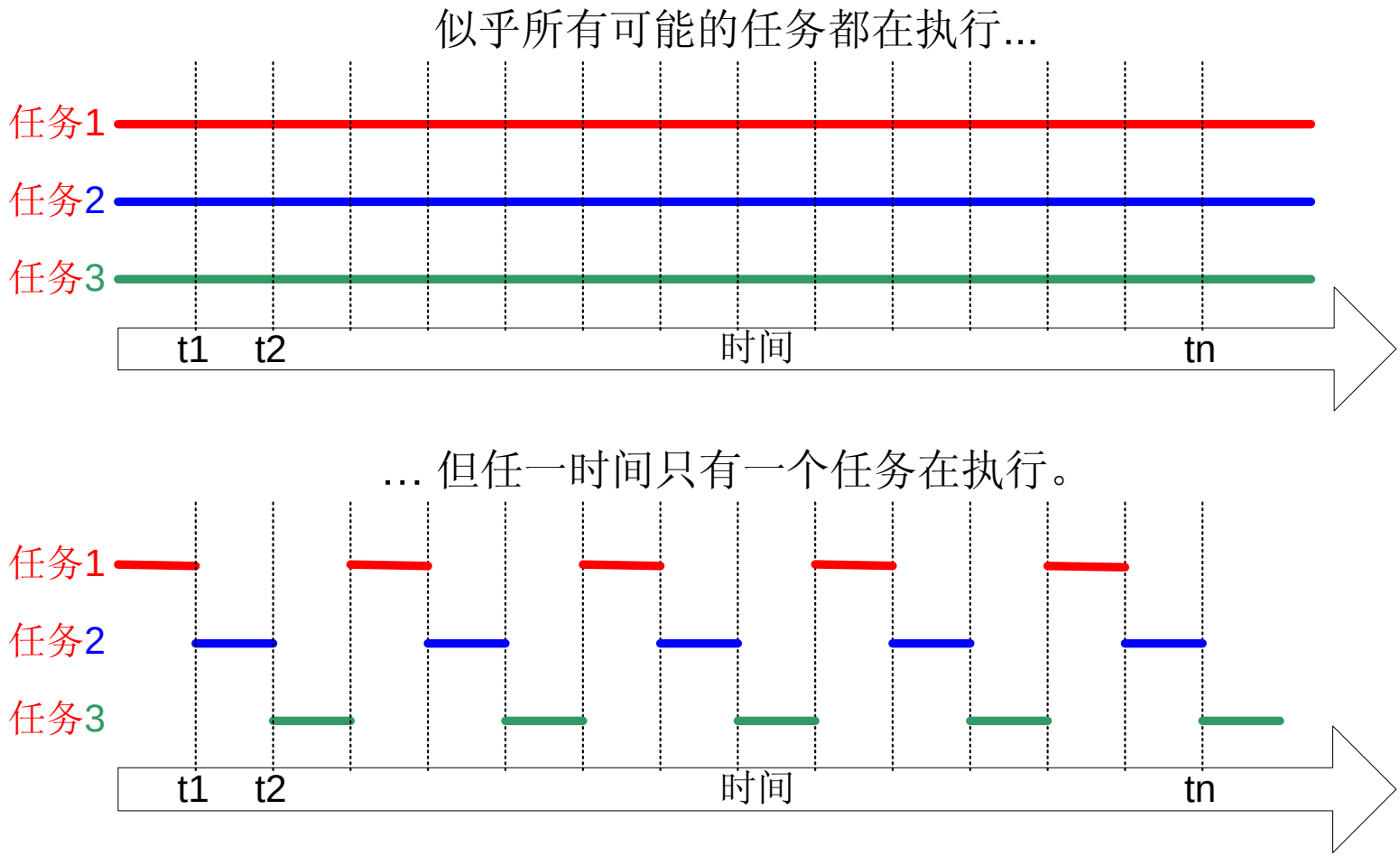
- 内核/服务程序的区分取决于内核的设计
  - Monolithic几乎所有都在内核空间中运行
  - Micro在用户空间中运行许多服务程序

‘Mini’ 实时内核



- 为在有限资源基础上实现具有实时性能的系统所设计

并行处理?



RTOS中的‘RT’

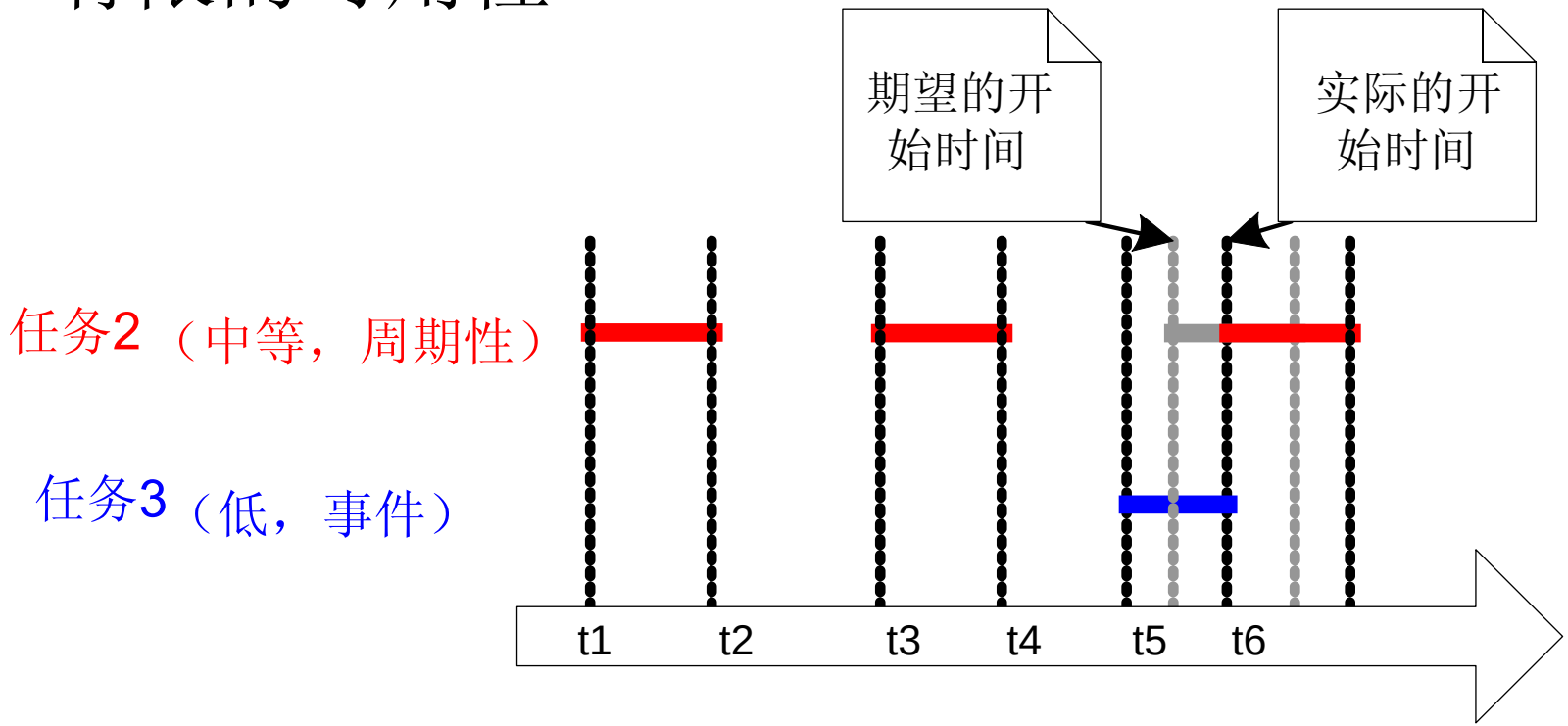
- 硬实时任务
  - 必须在特定时间内完成其功能
  - 如果错过最后期限将会导致一些不良的后果
- 软实时任务
  - 具有一个首选的完成时间
  - 错过最后期限不会导致灾难性的后果
- 如何实现?
  - 每一个任务都被分配一个优先级
  - 随后选择一个调度策略以确保满足实时要求

调度策略

- 动态优先级调度
  - 调度程序自己定义了每一个任务的优先级
    - 最后期限
    - 最后期限最近的优先级最高
    - 优先级老化
    - 等等
- 对于小型嵌入式系统并不总是实用的

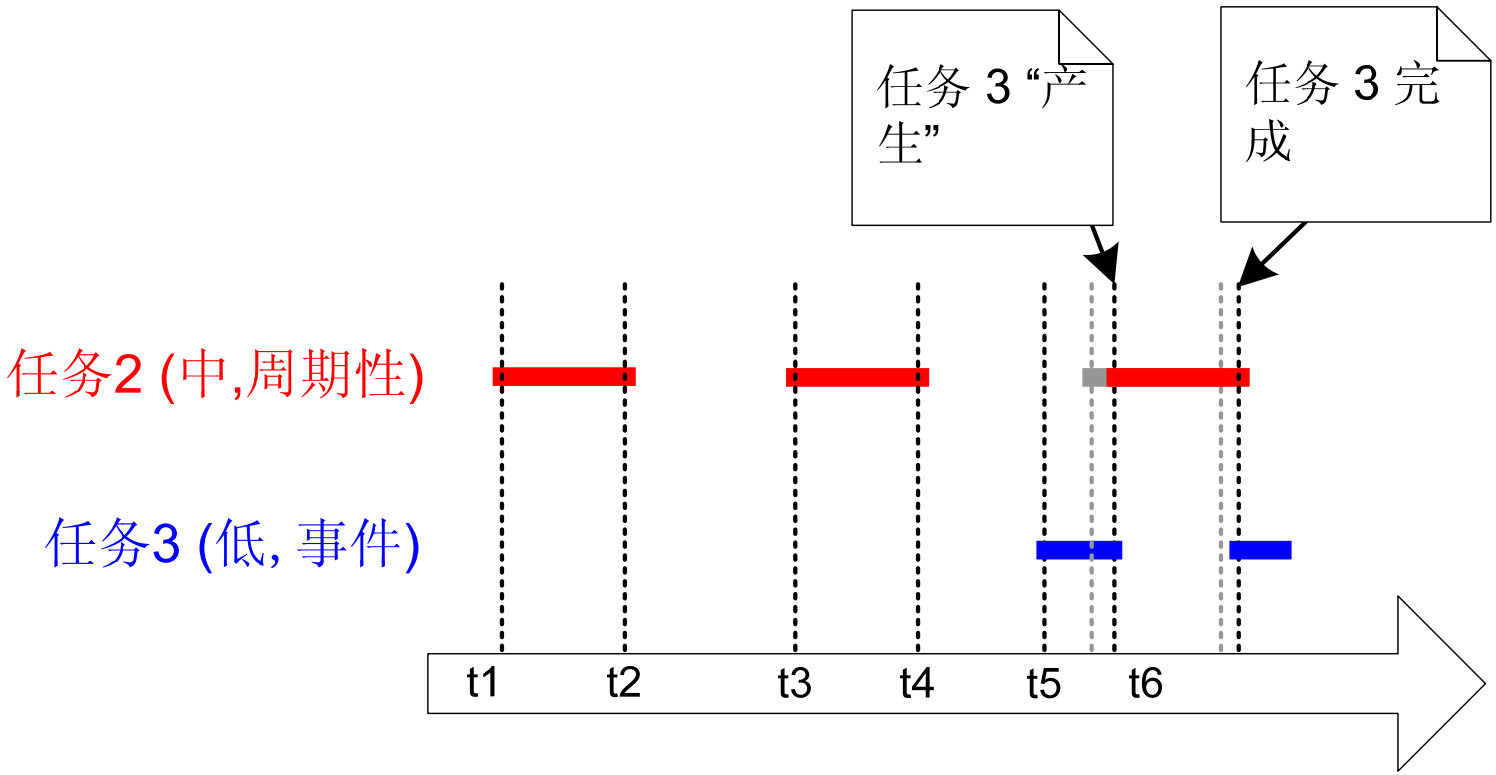
调度策略

- 运行至结束
  - 调度程序实现非常简单，但...
  - 有限的可用性



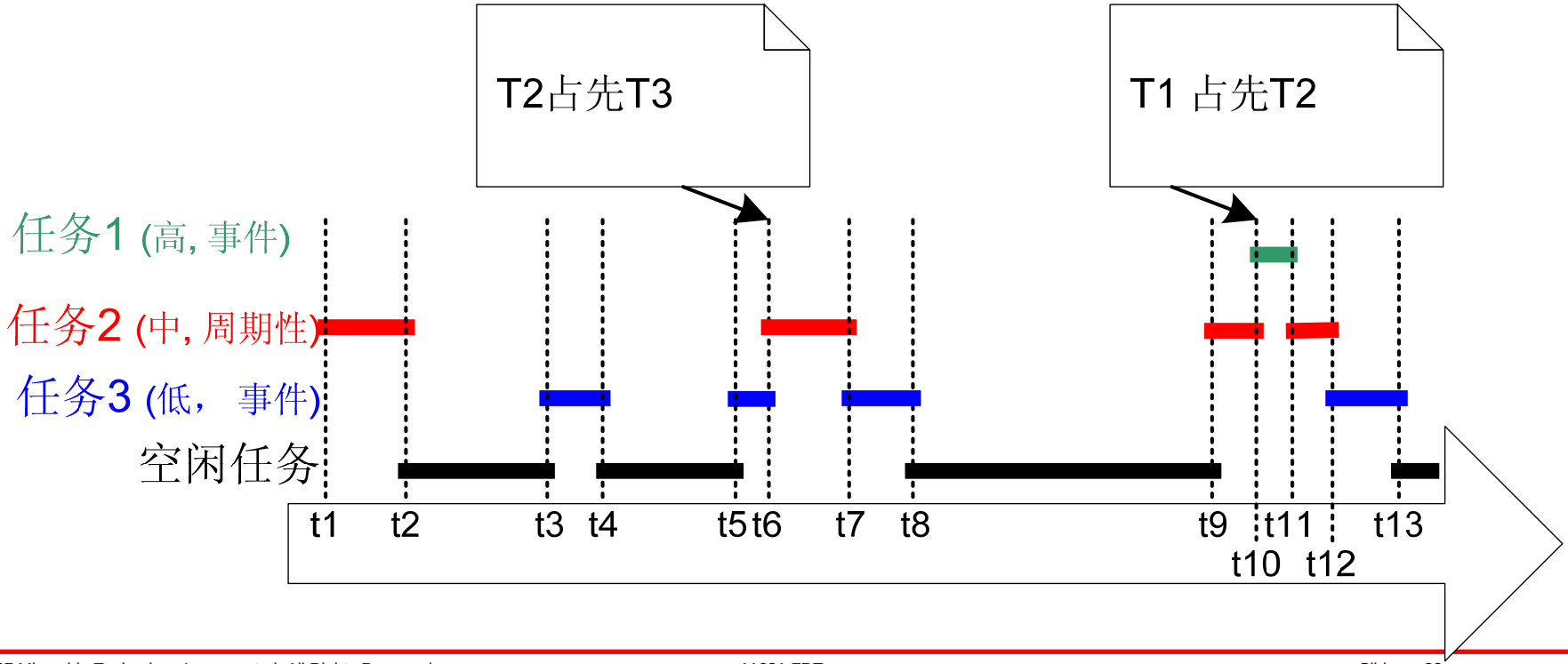
调度策略

- 基于优先级的合作式调度



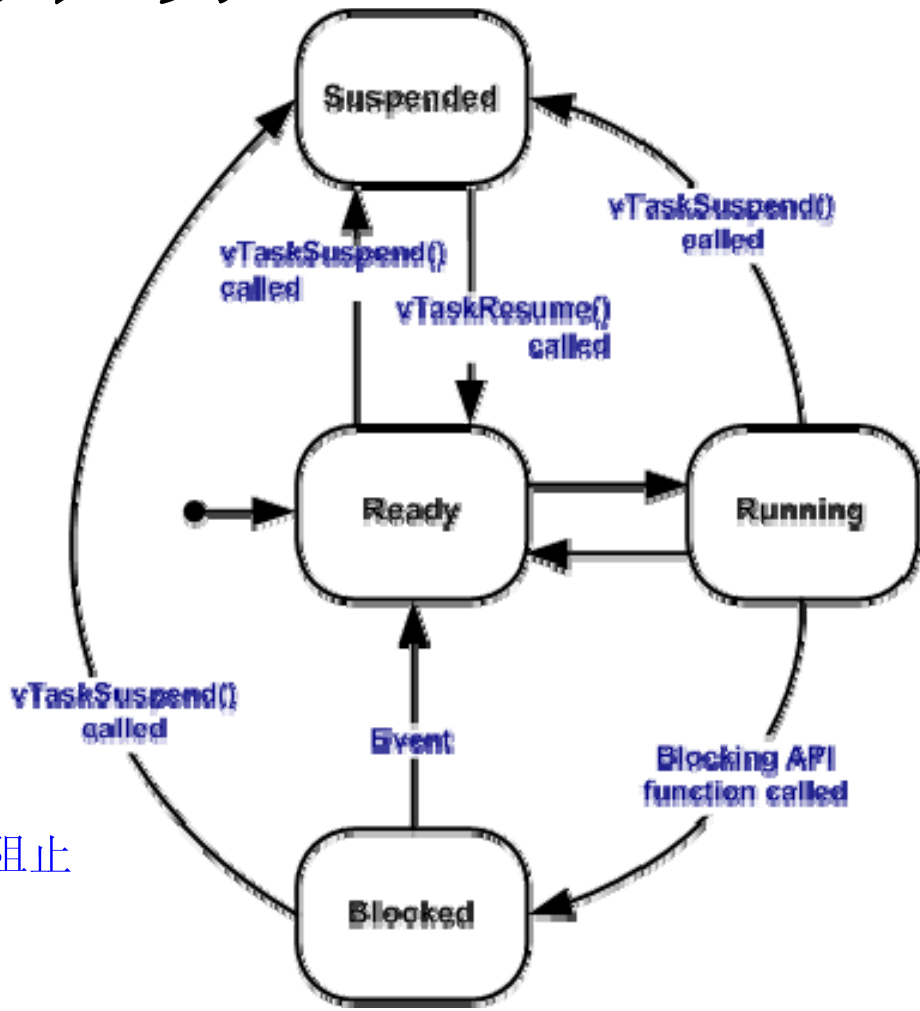
调度策略

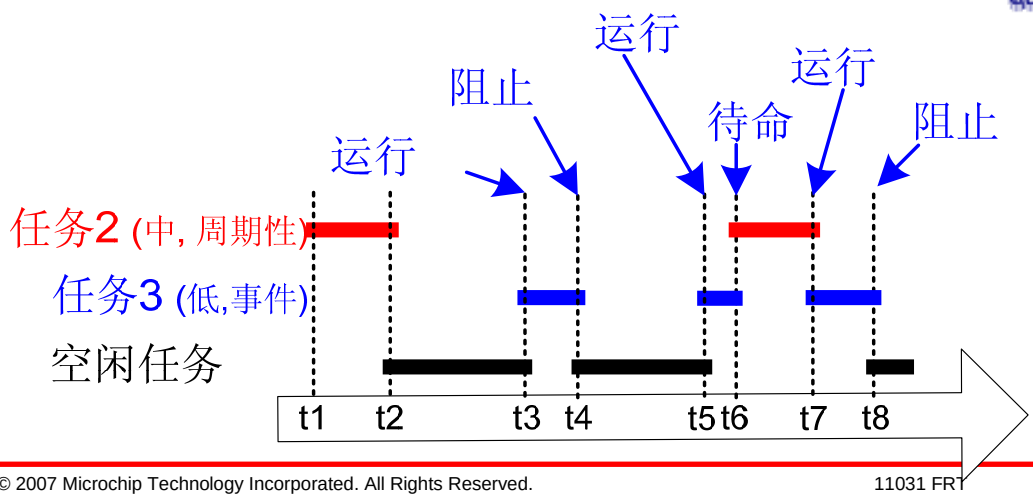
- 固定优先级占先调度
  - 优先级由应用而非内核设定
  - 引入了空闲任务，清楚显示了空闲的处理时间
  - Tick和外部事件可产生占先



何为任务?

- 自治的
  - 无需了解调度程序活动
  - 顺序的过程?
  - 在“虚拟处理器”上运行?





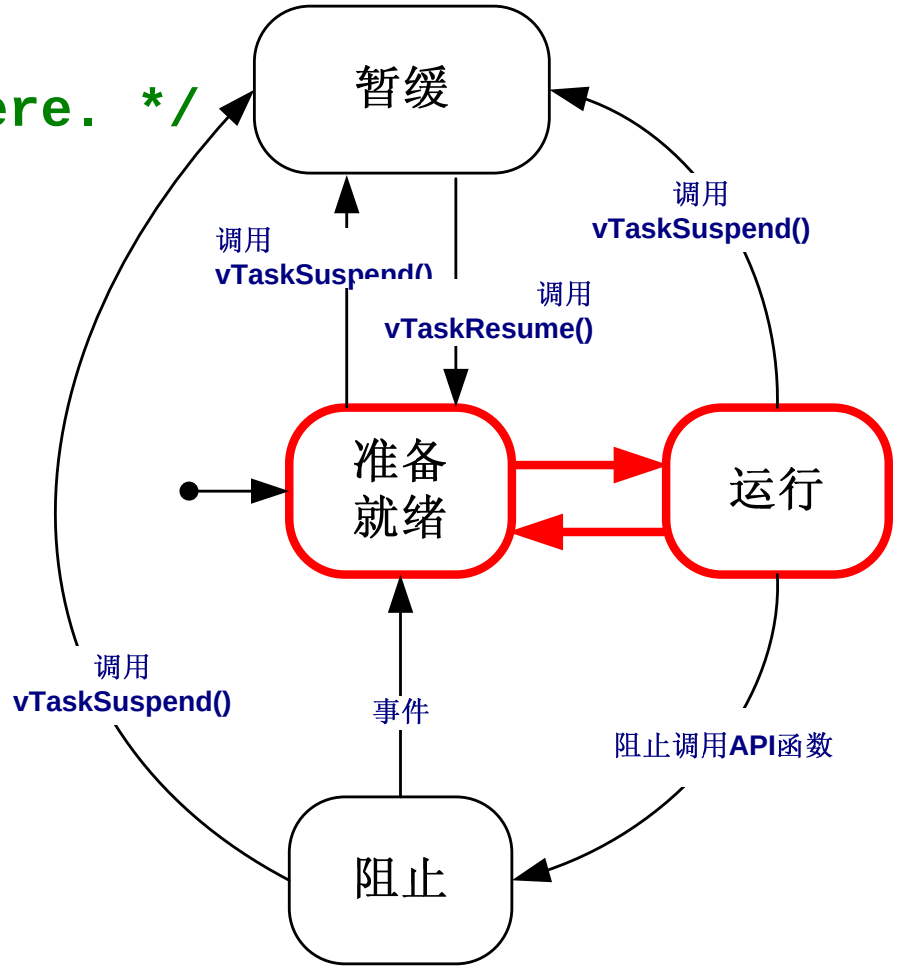
重温处理特性

- 使用任务来实现
  - 连续处理
  - 时间触发（或周期性）处理
  - 事件驱动处理
  - “单事件（One shot）”处理
- **Hook 函数**
  - 带来的便利
    - **Tick hook**
    - **空闲hook**

连续处理

```
void aTask( void * pvParameters )
{
  for( ;; )
  {
    /* Task processing goes here. */
  }
  vTaskDelete( NULL );
}
```

- 如果它从未阻塞或放弃，则只能被占先

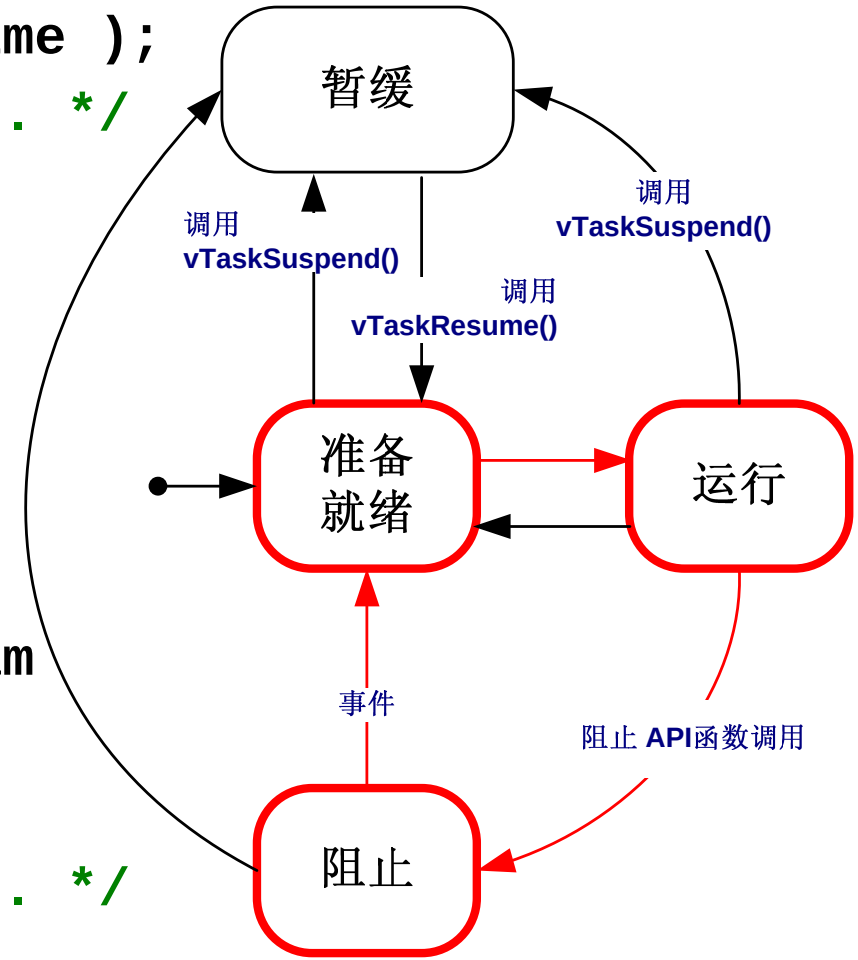


时间触发/周期性任务

```
void aFixedFrequencyPeriodicTask( void * pvParameters)
{
  for( ;; )
  {
    vTaskDelayUntil( aRelativeTime );
    /* Task processing goes here. */
  }
}
```

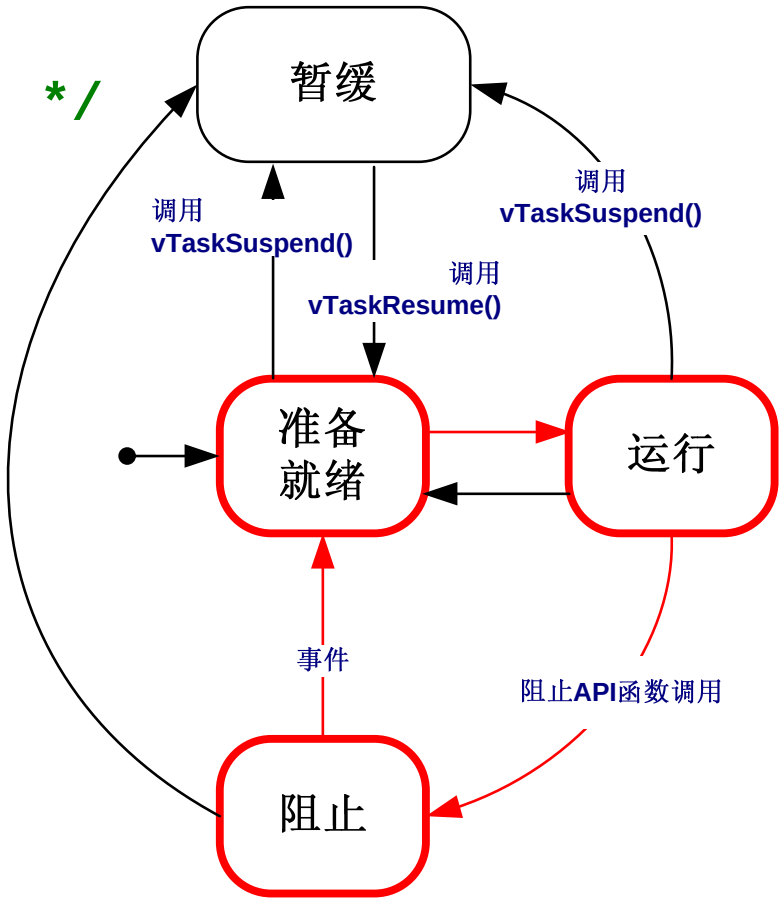
时间指定以‘tick’为单位

```
Void aPeriodicTask( void *pvParam
{
  for( ;; )
  {
    vTaskDelay( aTime );
    /* Task processing goes here. */
  }
}
```



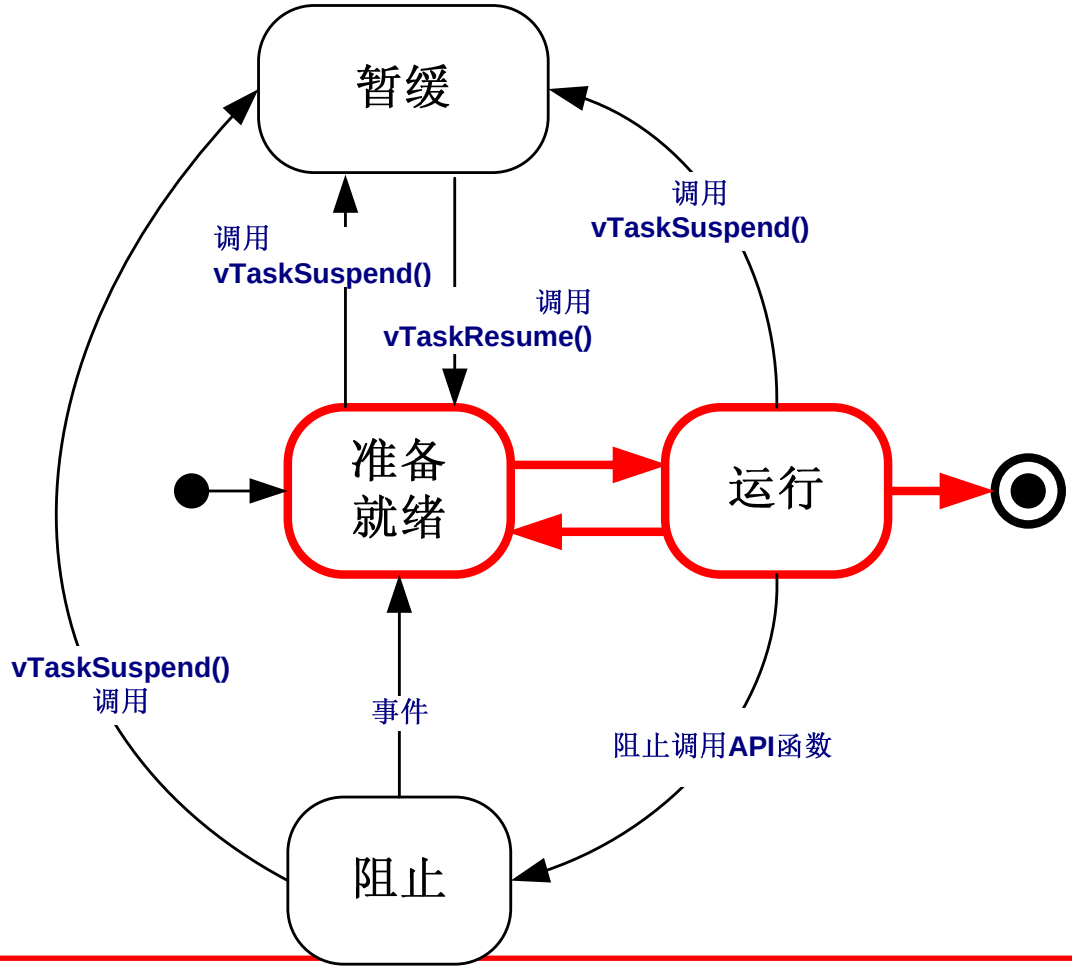
事件驱动任务

```
void aPeriodicTask( void * pvParameters)
{
  for( ;; )
  {
    xSemaphoreTake( aSemaphore,
                    aTimeOut );

    /* Task processing goes here. */
  }
}
```



单事件（One Shot）任务

```
void aOneShotTask( void * pvParameters )
{
  /* Task processing goes here. */
  vTaskDelete( NULL );
}
```



Hook 函数

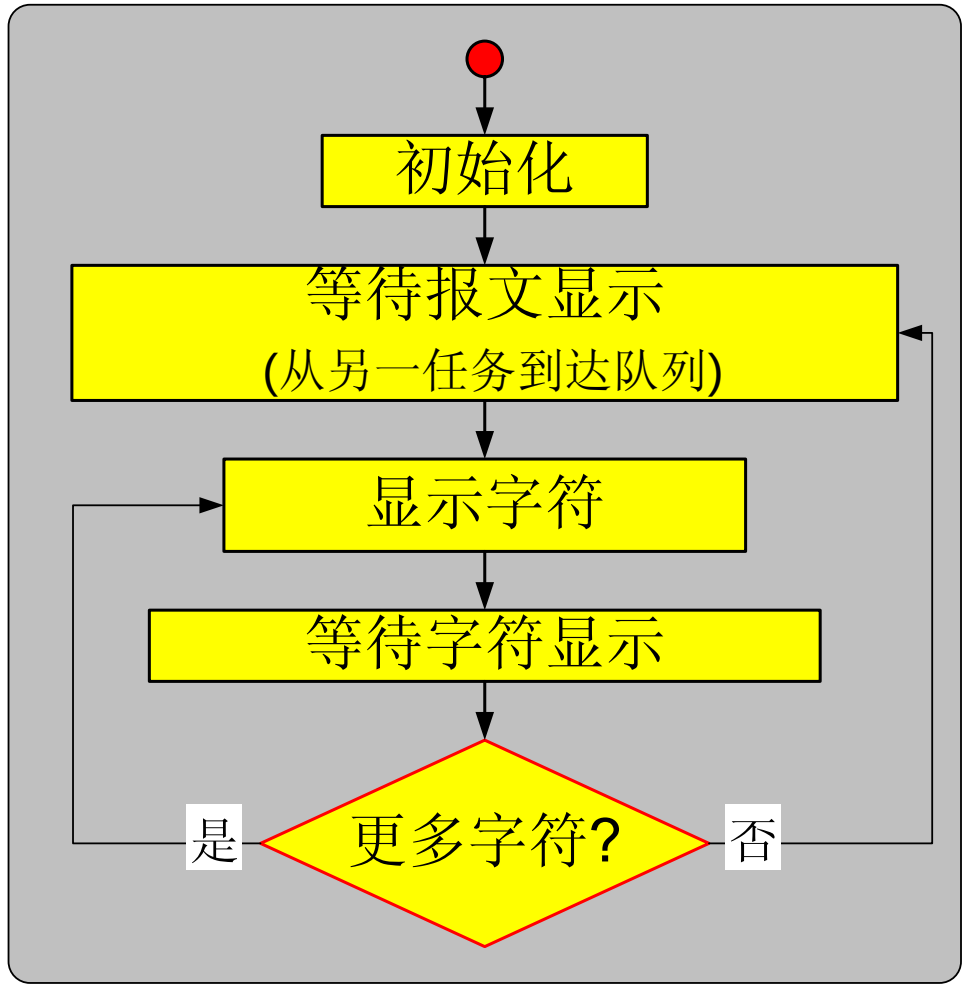
- 使用周期性**ISR** 对时间进行测量
- 在每一次**tick**递增时执行**“tick hook”**
- 从空闲任务中可调用**“Idle hook”**
  - 低优先级处理
  - 测量空闲时间
  - 使处理器处于低功耗模式

```
void vApplicationIdleHook( void )
{
    /* Perform low priority
    processing here. */
}
```

```
void vApplicationTickHook( void )
{
    /* Perform quick periodic
    processing here - timers,
    give semaphores, etc. */
}
```

重温LCD

- 将LCD句柄重新组织为自治任务
- 没有来自LCD句柄的入口或出口
- 不关心报文由何处产生



重温LCD

- LCD ‘任务’
- 报文可从任何地方发送

```
static void vLCDTask( void *pvParameters )
{
xLCDMessage xMessage;

    /* Initialize */
    prvSetupLCD();

    for( ;; )
    {
        /* Wait until a message to display arrives. */
        xQueueReceive( xLCDQueue, &xMessage, portMAX_DELAY );

        /* Write message to the LCD. */
        prvLCDPutString( xMessage.pcMessage );
    }
}
```

重温LCD

- **LCD**初始化（原始但有效的技术）
- 单一顺序函数，并不取决于处理器的速度

```
static void prvSetupLCD( void )
{
    /* Setup the PMP. */
    PMCON = 0x83BF;
    PMMODE = 0x3FF;
    PMAEN = 1;
    PMADDR = 0x0000;
    vTaskDelay( lcdSHORT_DELAY );

    /* Set the default function. */
    PMDIN1 = lcdDEFAULT_FUNCTION;
    vTaskDelay( lcdSHORT_DELAY );

    /* Set the display control. */
    PMDIN1 = lcdDISPLAY_CONTROL;
    vTaskDelay( lcdSHORT_DELAY );

    /* Clear the display. */
    PMDIN1 = lcdCLEAR_DISPLAY;
    vTaskDelay( lcdSHORT_DELAY );

    /* Set the entry mode. */
    PMDIN1 = lcdENTRY_MODE;
    vTaskDelay( lcdSHORT_DELAY );
}
```

重温LCD

```
static void prvLCDPutString( portCHAR *pcString )
{
    /* Write out each character with appropriate
    delay between each. */
    while( *pcString )
    {
        PMADDR = 0x0001;
        PMDIN1 = *pcString;
        pcString++;
        vTaskDelay( lcdSHORT_DELAY );
    }
}
```

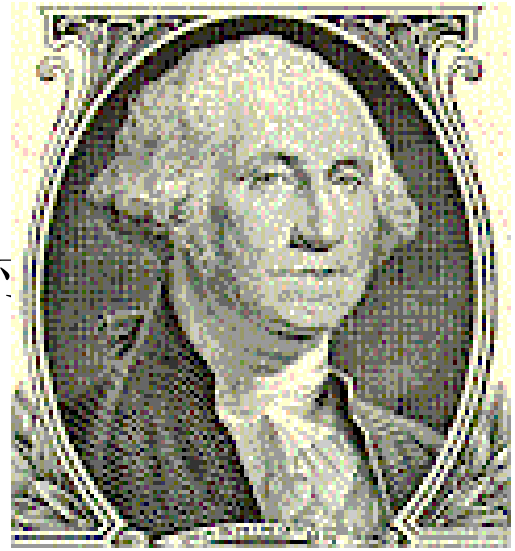
- 向LCD写入一个字符串
- 具有单一入口和出口的简单单一顺序函数

重温可维护性

COTS组件的集成（后面将展示文件系统实例）	
代码重用	
混合处理需求	
同时进行的团队研发	
可测试性	
混合有硬和软实时任务	
可升级性（不易受到应用改变的影响）	
允许硬件发生变化 （速度变化）	

代价是什么?

- **RAM**
  - 每一个任务存在自己的堆栈
  - 协同程序共用堆栈
  - 内核自身使用大约110字节
- 闪存
  - 所需最小闪存空间，通常未优化时
- 处理
  - 周期性tick中断
  - 环境切换时间
  - 但是…
    - …我们已经看见如何节省处理时间!
- 另外，这是一种新的应用设计思路



Lab #2 目标

- 评估使用这一替代解决方案实现前面的**LCD**示例
- 这种替代解决方案是否更好吗？

Lab #2 结论

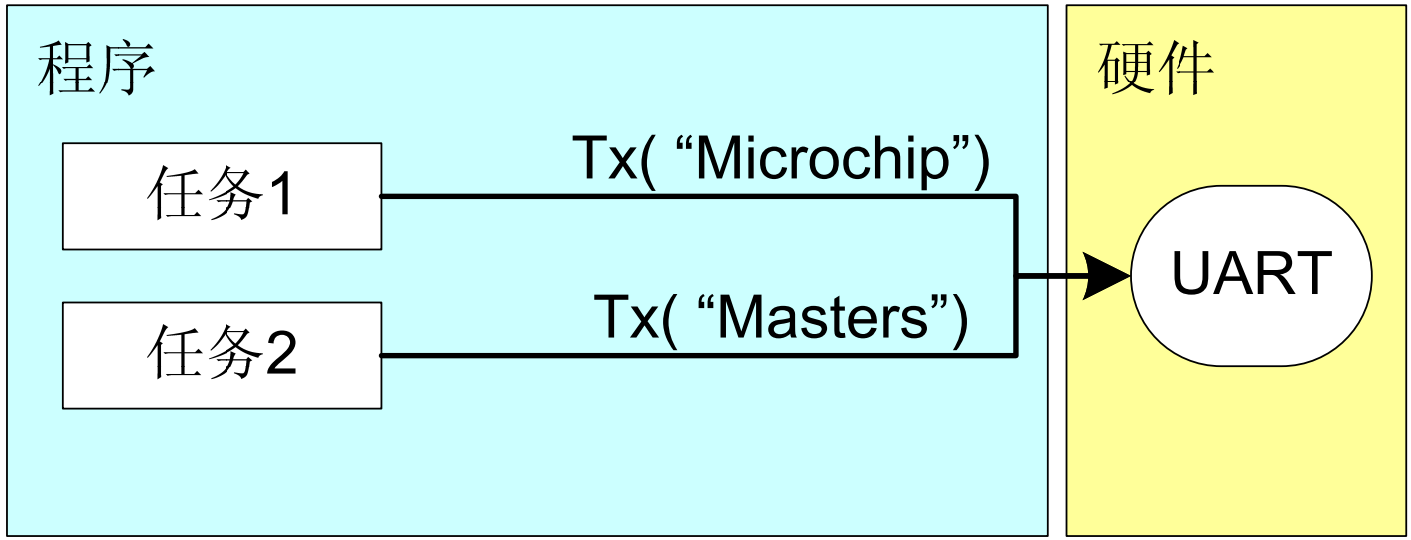
回顾**Lab #2**的问题

课程安排

1. 单一任务应用
2. 多任务基础
3. 其余概念和需要考虑的问题
   - 互斥
   - 任务间通信
   - 优先级
   - 中断句柄
4. 使用**FreeRTOS.org**创建一个多任务应用

互斥

- 如果两个任务同时对同一资源进行访问会发生什么?



互斥

- 硬件资源
  - 读、修改和写

```
69:                                 {
70:                                     PORTA |= uxLEDBit;
 021BC   801610      mov.w 0x02c2,0x0000
 021BE   97B8FF      mov.w [0x001e-2],0x0002
 021C0   700001      ior.w 0x0000,0x0002,0x0000
 021C2   881610      mov.w 0x0000,0x02c2
71:                                 }
```

互斥

- 数据
  - 即使单一变量！
    - 非原子访问

```
138:
139:                    /* A global variable! */
140:                    unsigned int x = 10;
141:
142:                    int main( void )
143:                    {
144:                     x += 5;
 00608   80EC80         mov.w 0x1d90,0x0000
 0060A   400065         add.w 0x0000,#5,0x0000
 0060C   88EC80         mov.w 0x0000,0x1d90
145:
```

互斥

- 重新进入?
  - 这些函数中哪个是“安全线程”?

```
int vAFunction1( void )
{
static int z = 0;

    z++
    return ++z;
}
//////////////////////////
int vAFunction2( int z )
{
    z++
    return z;
}
//////////////////////////
int vAFunction3( void )
{
int z = 0;

    z++
    return z;
}
```



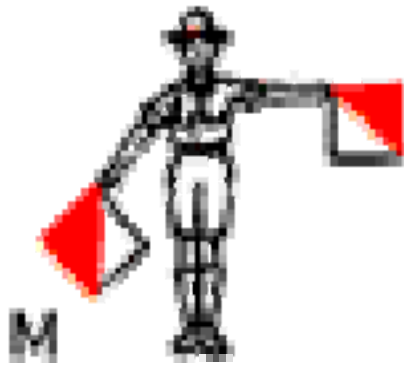
互斥技术

1. 好的设计习惯！
2. 在实时内核的帮助下
   a) 临界区
   b) 调度程序锁定
   c) 任务间通信 (我们的LCD实例使用了某种形式的互斥)

```
void aTask( void * pvParameters )
{
  for( ;; )
  {
    taskENTER_CRITCAL(); /* Or vTaskSuspendAll() */
        /* Access resource here. */
    taskEXIT_CRITICAL(); /* Or vTaskResumeAll() */
  }
}
```

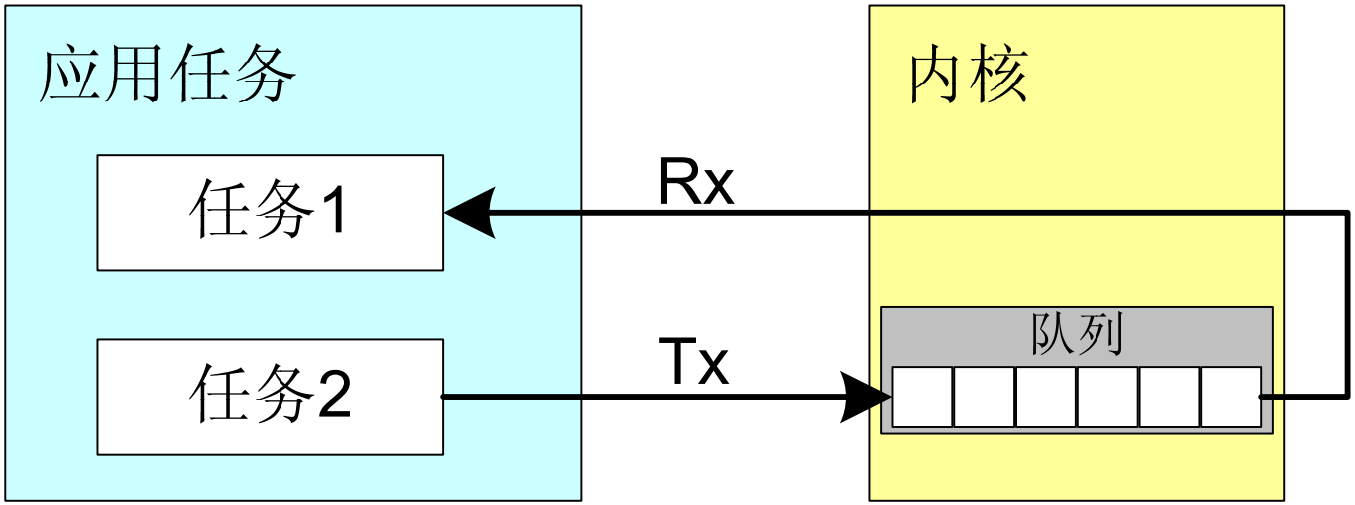
任务间(以及和ISR)的通信

- 队列和信号量的共性
  - 同步
  - ISR至任务(反之亦然)的通信
- 队列
  - 传递数据
- 信号量
  - 互斥



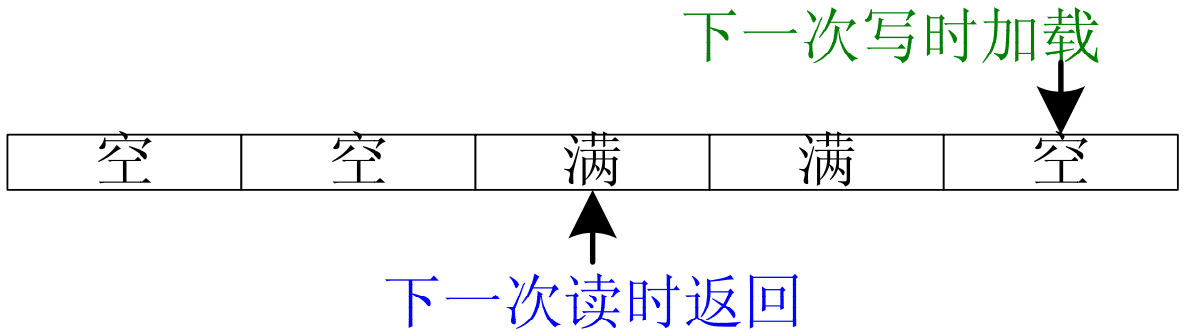
队列

- 安全地将数据从一个任务传递到另一个任务
- 内核负责互斥处理!



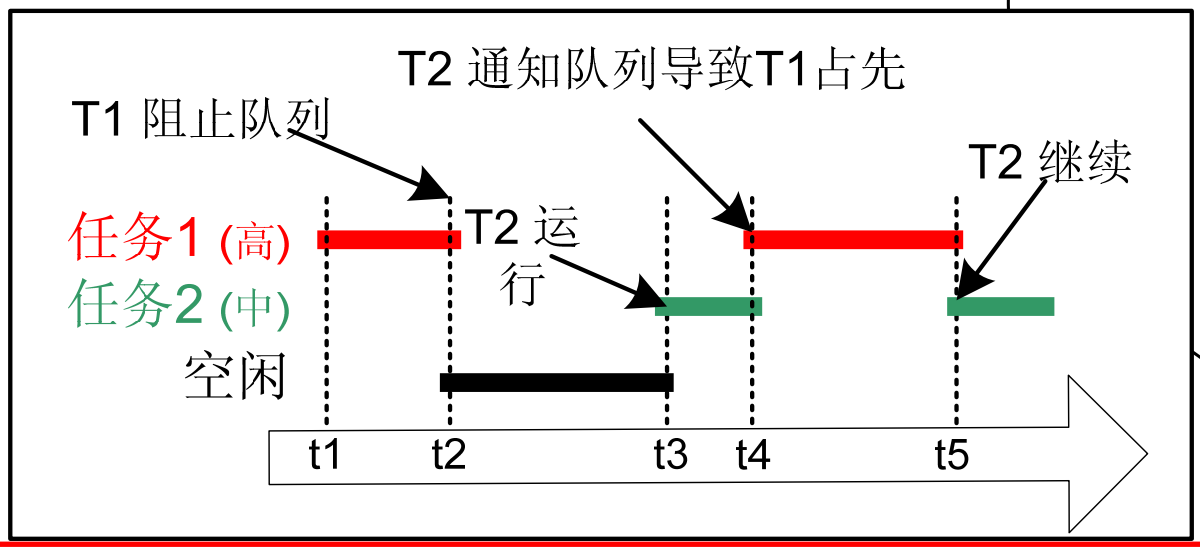
队列

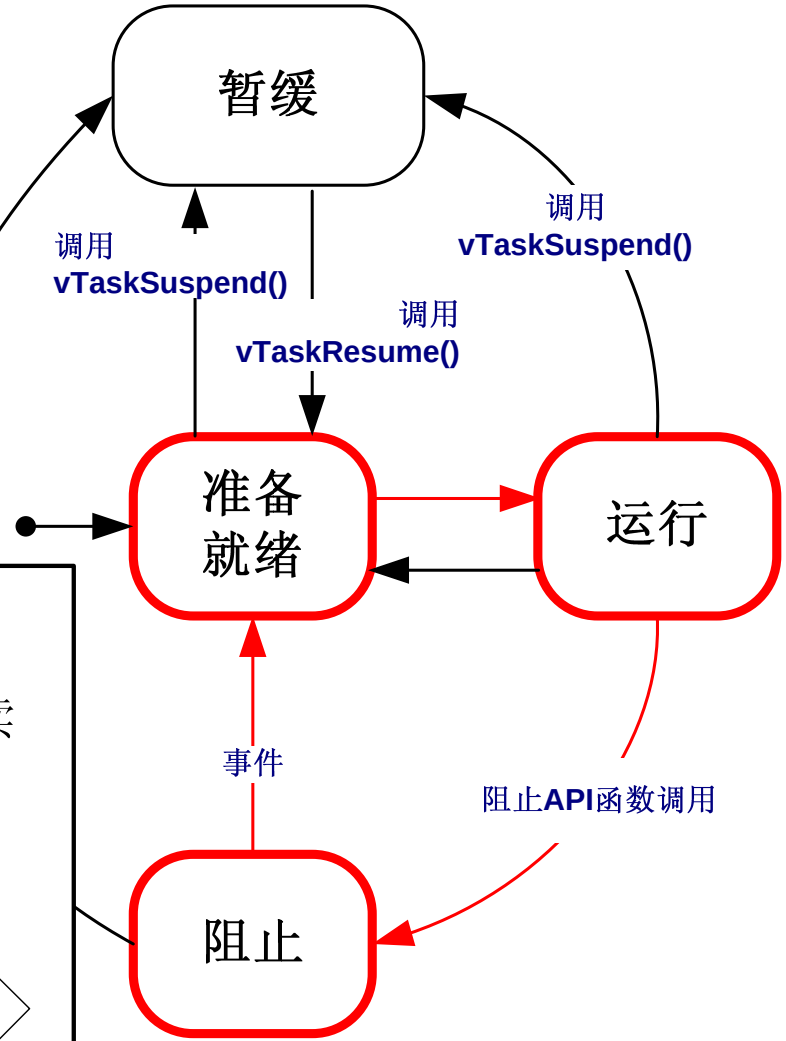
- 一个具有智能的FIFO缓冲器
- 数据被逐字节进行复制
  - 指针可进行排队但必须要小心（参见最后实验中的实例）！
- 队列可保持有限数量的数据项
- 当队列建立时每一队列中数据项的数目和大小是设定好的



队列阻塞

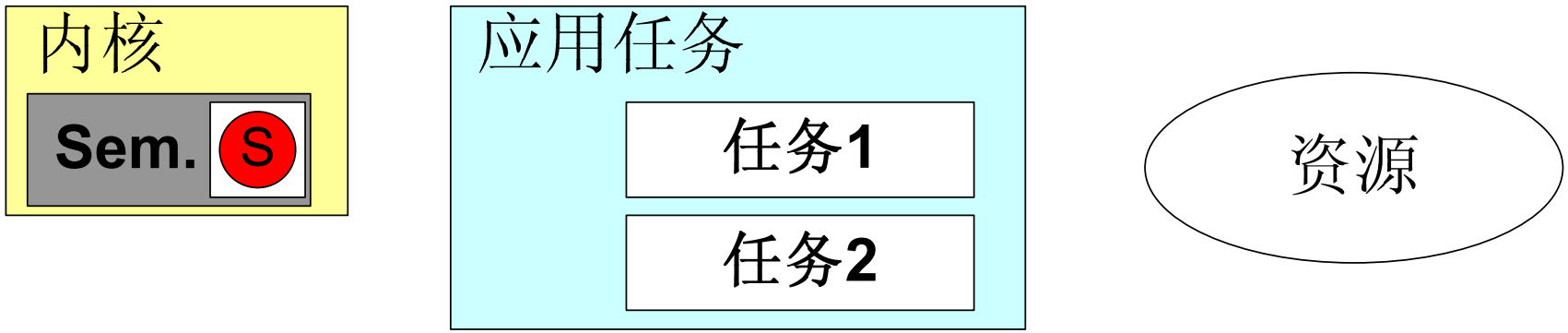
- 如果一个队列已满/空会怎么办?
  - 可指定一个可选的阻塞时间





二进制信号量

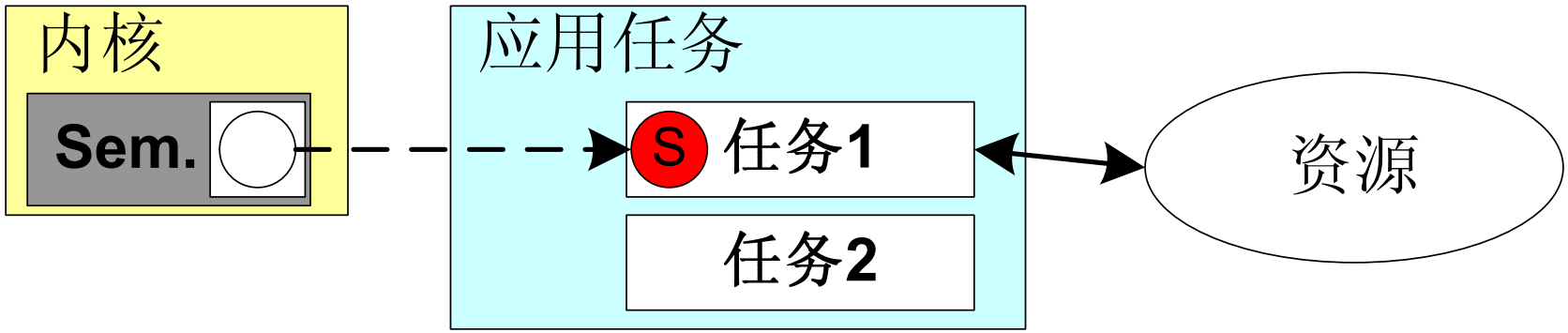
- 互斥机制



  - 如果任务不能获得信号量，则它也不能访问资源

二进制信号量

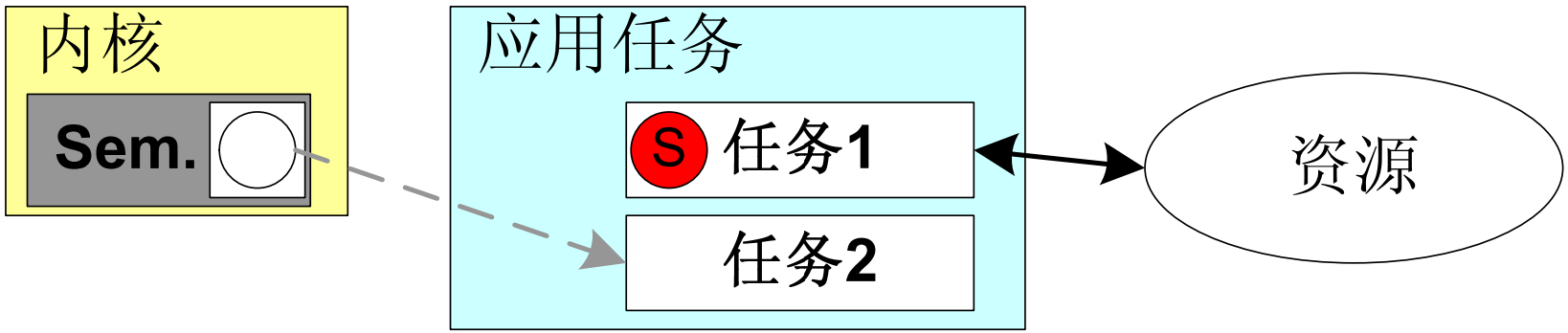
- 互斥机制



  - 任务1 “获得”信号量因此现在可以访问资源

二进制信号量

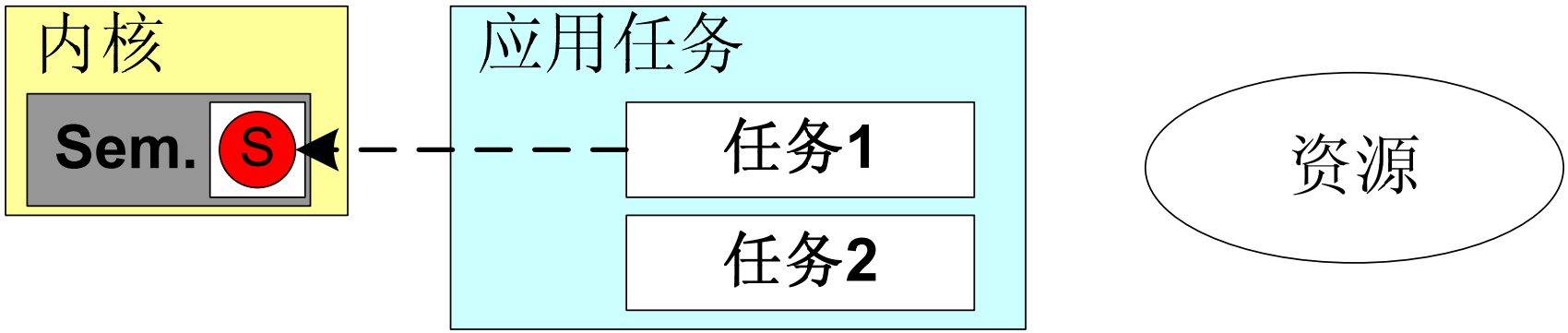
- 互斥机制



  - 任务2当前不能获得信号量 – 但可进行阻塞等待直至可以获得

二进制信号量

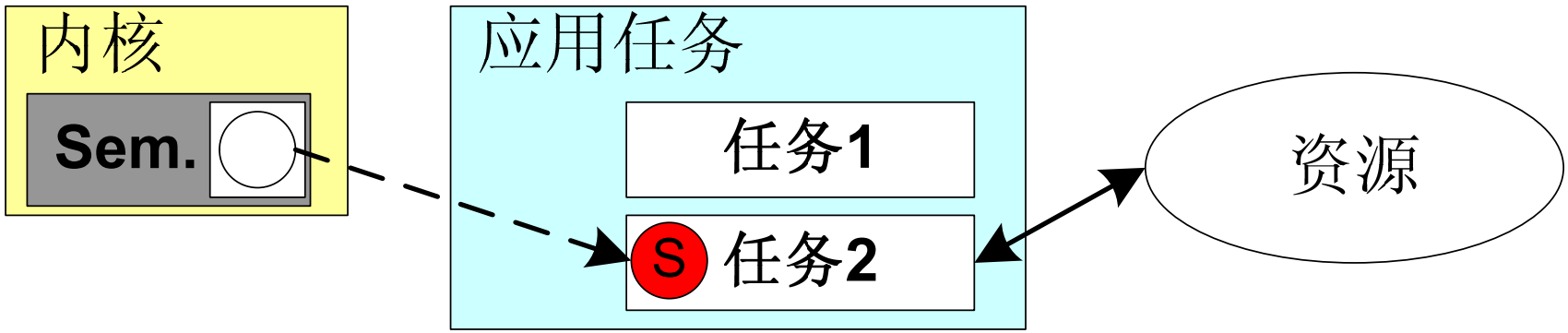
- 互斥机制



  - 任务1 归还信号量（但再不能对资源进行访问）

二进制信号量

- 互斥机制



  - 如果需要，那么任务 2可获得信号量

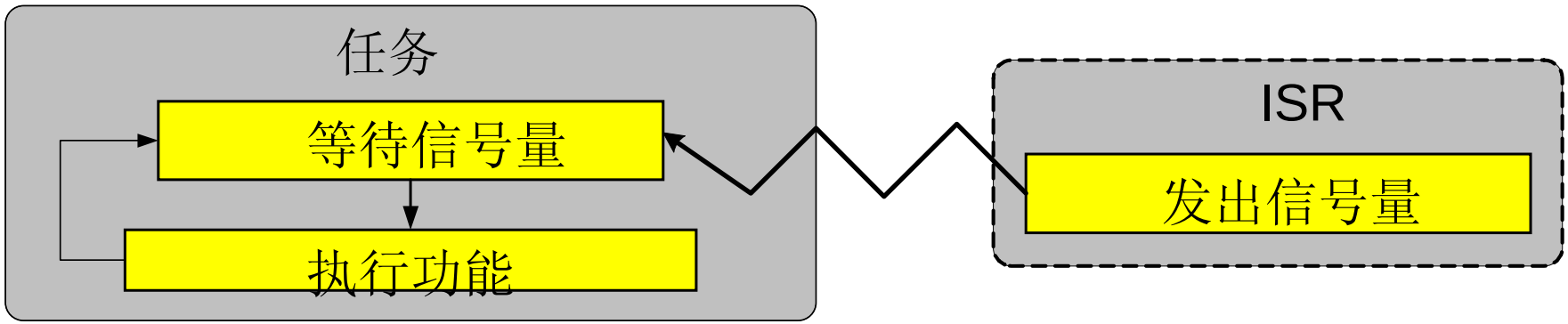
信号量

- 代码实例

```
void vTask1( void * pvParameters )
{
  for( ;; )
  {
    /* Need access, get semaphore. */
    xSemaphoreTake( xSemaphore, BLOCK_FOREVER );
    vLCDWrite( “Microchip” ); /* Access resource */
    /* Must remember to return semaphore. */
    vSemaphoreGive( xSemaphore );
  }
}

/*****************************************************/

void vTask2( void * pvParameters )
{
  for( ;; )
  {
    xSemaphoreTake( xSemaphore, BLOCK_FOREVER );
    vLCDWrite( “Masters” ); /* Access resource */
    vSemaphoreGive( xSemaphore );
  }
}
```

信号量

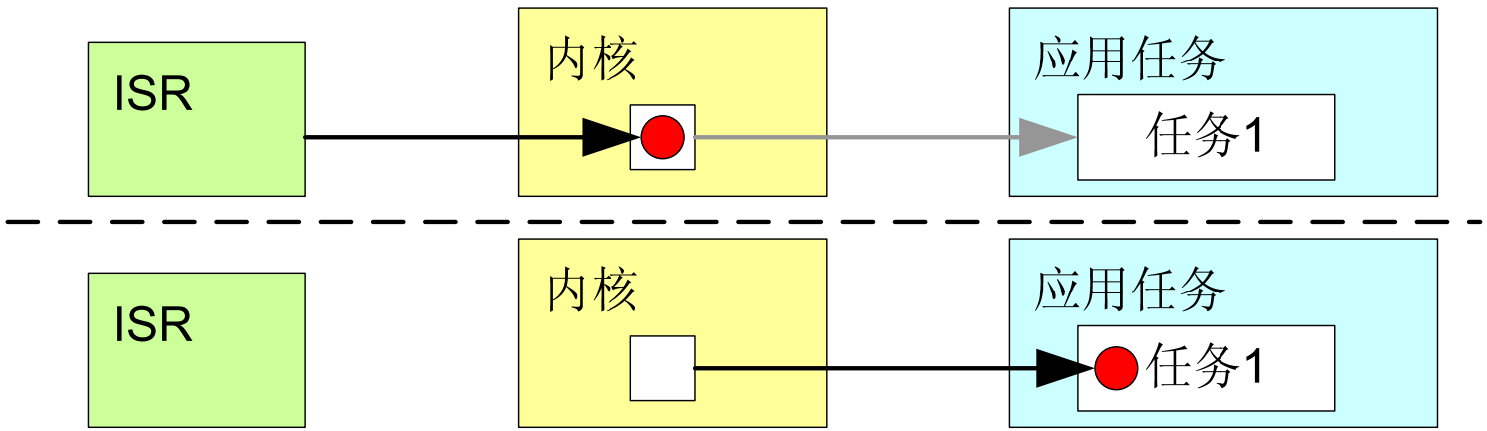
- 纯同步机制
- 与ISR同步的周期性处理
  - 注意：
    - 任务从不“归还”信号量
    - 即使任务执行周期很长，但仍处于运行状态



信号量

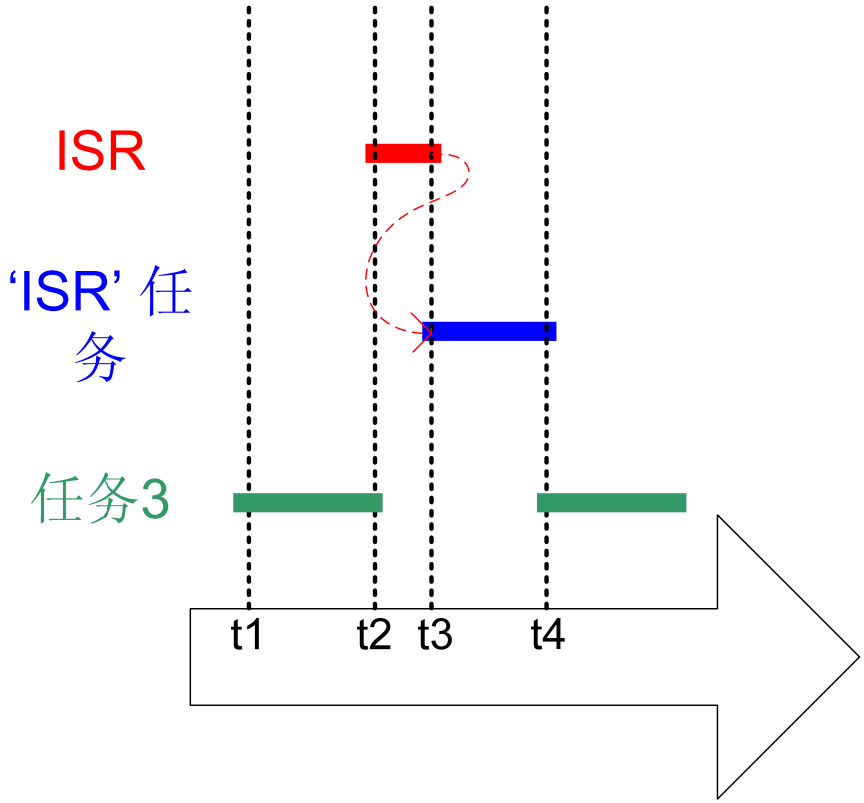
- 纯同步机制

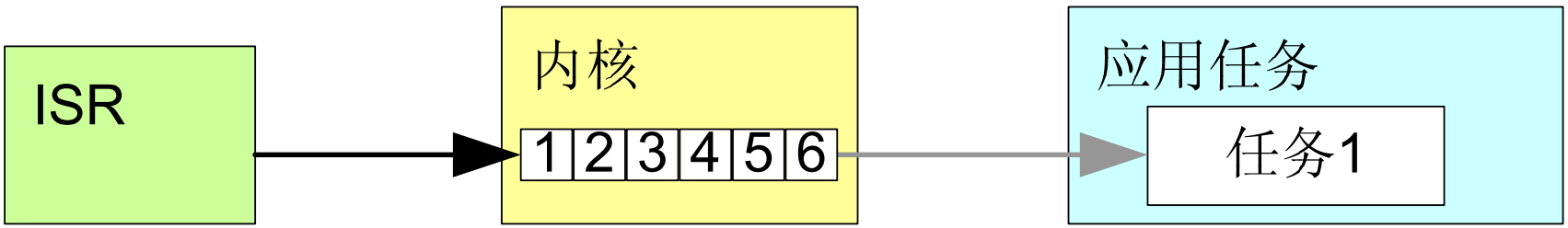
```
void vAnISR( void )
{
  xSemaphoreGiveFromISR( xSemaphore );
}

/*-----------------------------------------------------------*/

void vTask2( void * pvParameters )
{
  for( ;; )
  {
    xSemaphoreTake( xSemaphore, BLOCK_FOREVER );
    /* Processing done here. */
  }
}
```



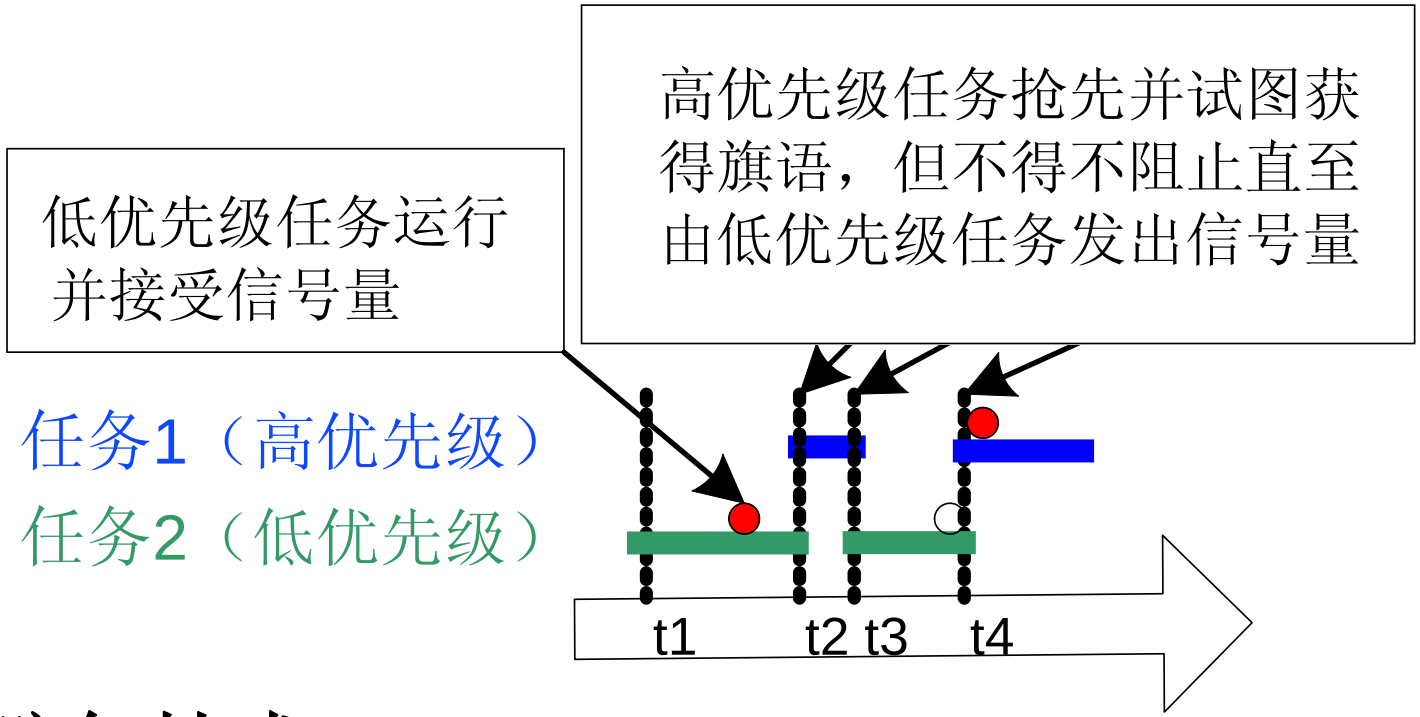
方便的中断处理

- 数据处理服从任务级别安排
- 在处理期间中断保持使能





何谓优先级反转?



- 最佳的避免技术?
  - 好的应用设计!
- 优先级继承机制
  - 仅能使影响最小化

死锁

- 当两个进程都在等待对方释放某个资源时
  - 任务1开始发送数据
  - 任务2优先级较高，也要发送数据，且来不及等待至任务1发送完毕

这在小型嵌入式设计中通常容易避免

活锁

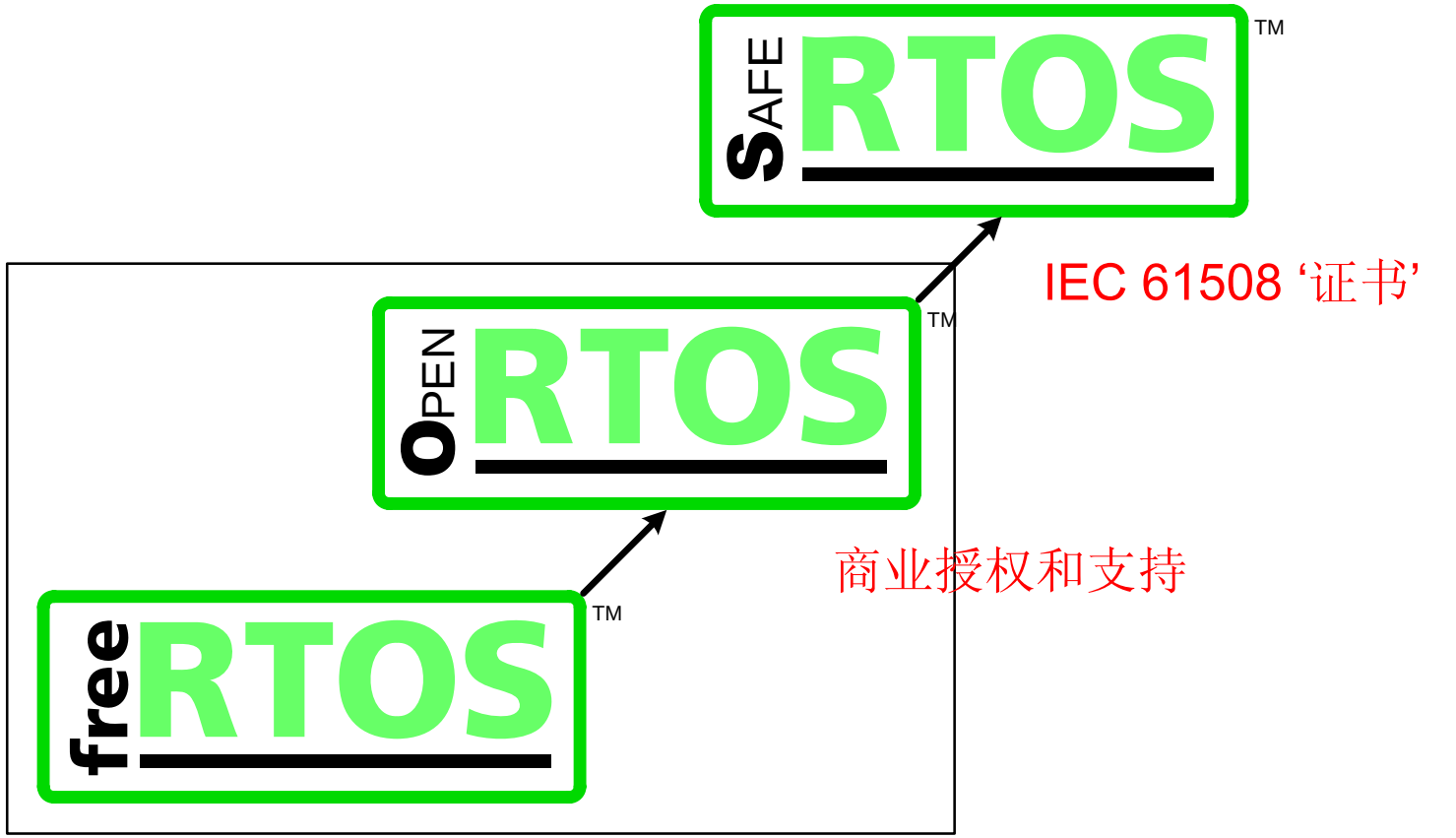
- 任务在运行但不能结束，因为它们必须在另一个任务结束后才能结束
  - “After you” 综合症

课程安排

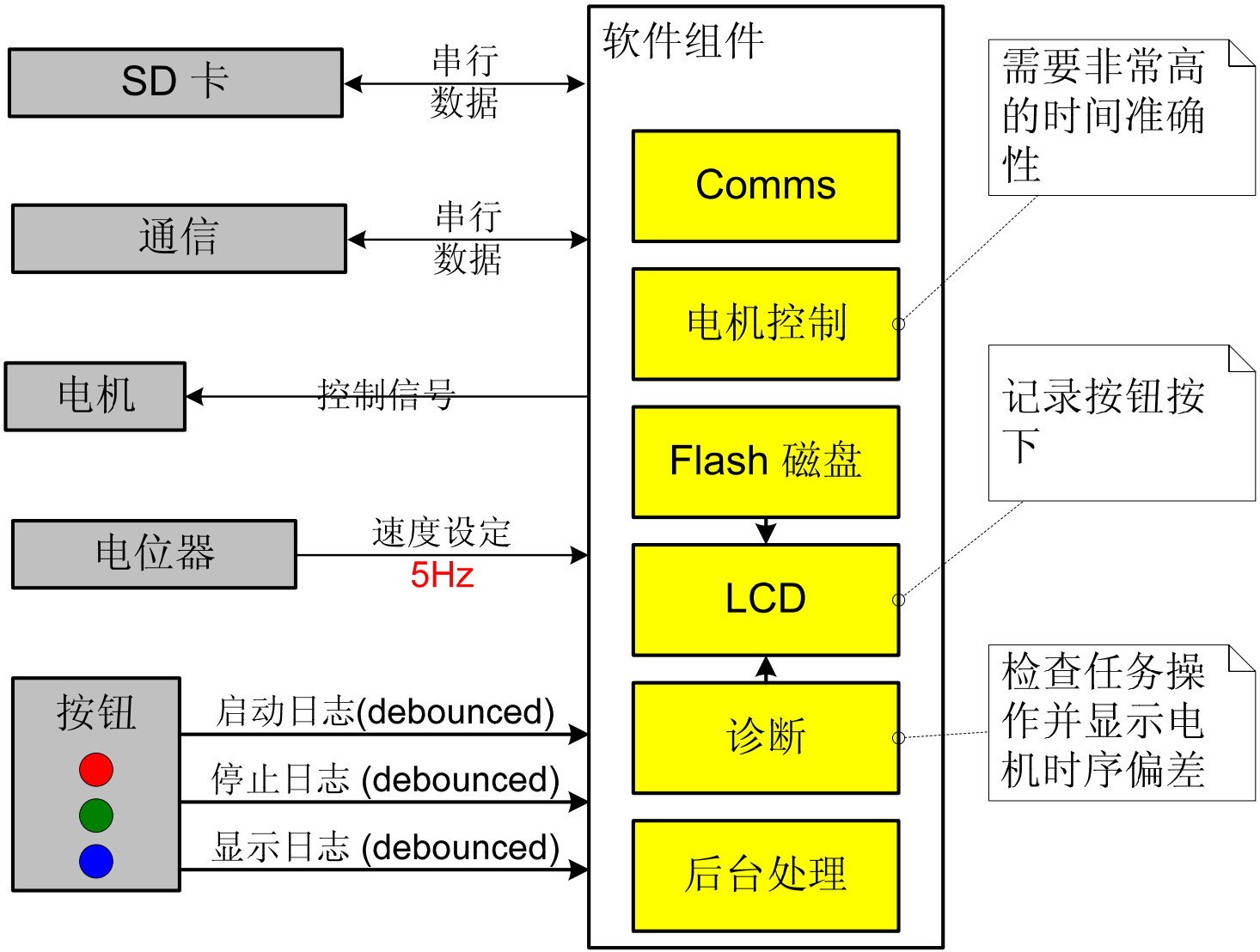
1. 单任务应用
2. 多任务基础
3. 其他概念何需考虑的问题
4. 使用**FreeRTOS.org**创建一个多任务应用
   - 关于FreeRTOS.org的更多信息
   - 创建一个FreeRTOS.org项目
   - 配置内核
   - 通过一个项目实例学习API
   - Lab #3 改变应用行为

FreeRTOS.org 设计目标

- 小型 + 便携式 + 快速 + 简单



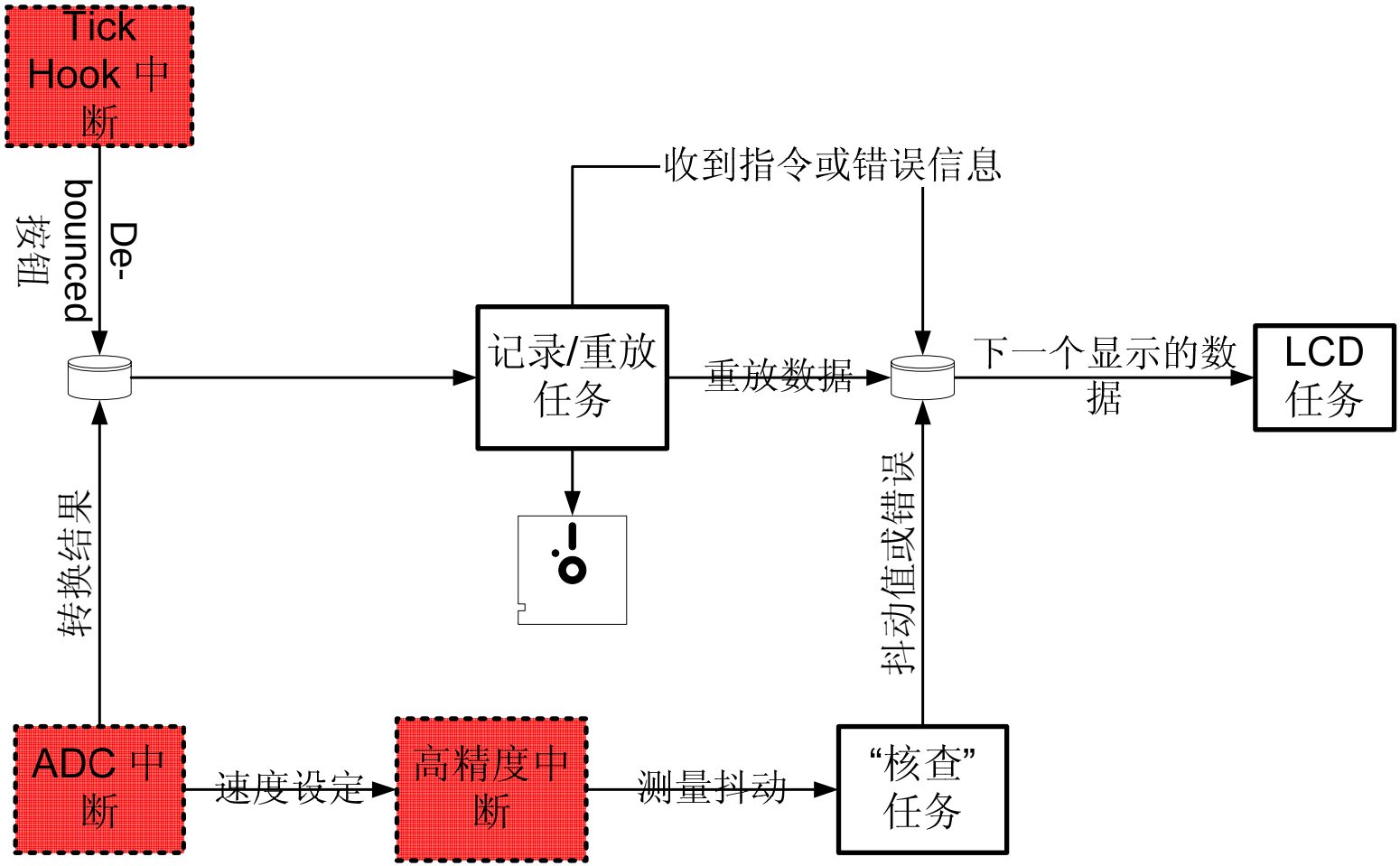
应用范例



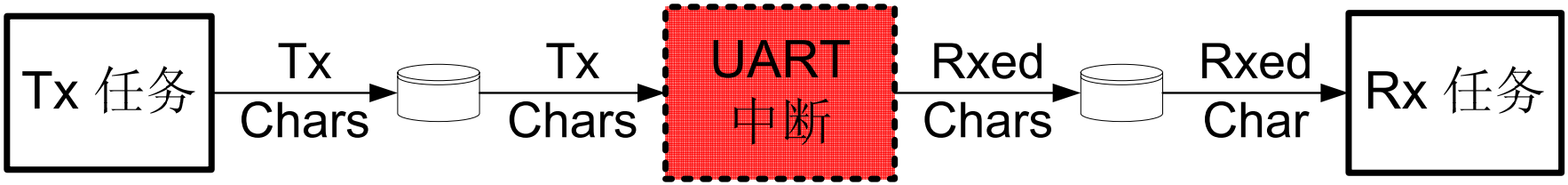
挑战

- 时序
  - 电机控制时序非常关键
  - Flash磁盘速度非常低
  - LCD速度非常低
- 互斥现象
  - 从多个源头对LCD进行写入
- 保持代码的简洁
  - 去除时序和接口的繁琐

主要功能



其余功能

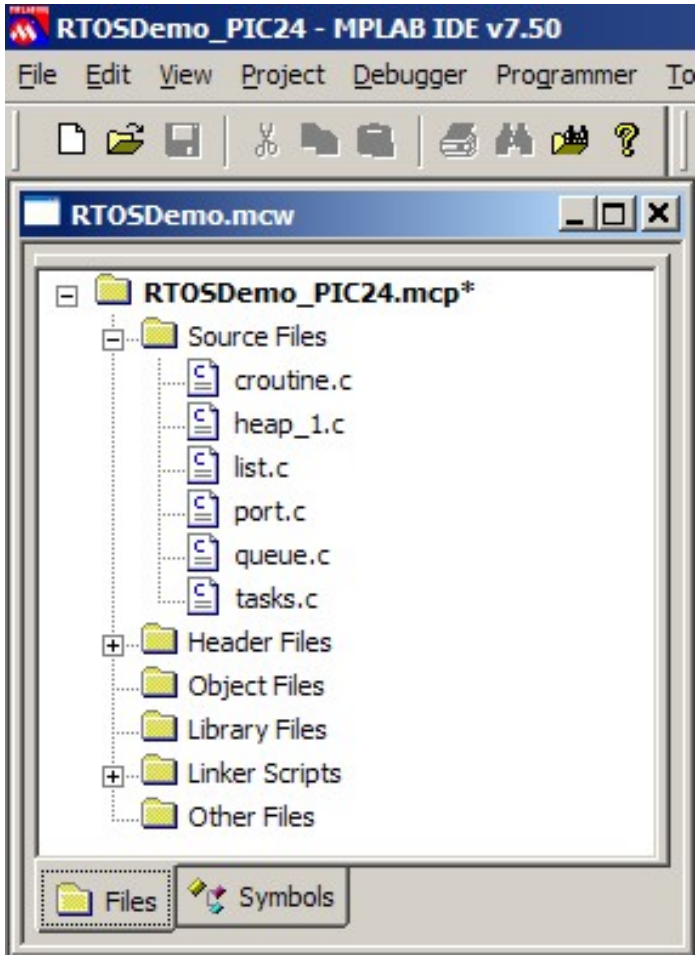


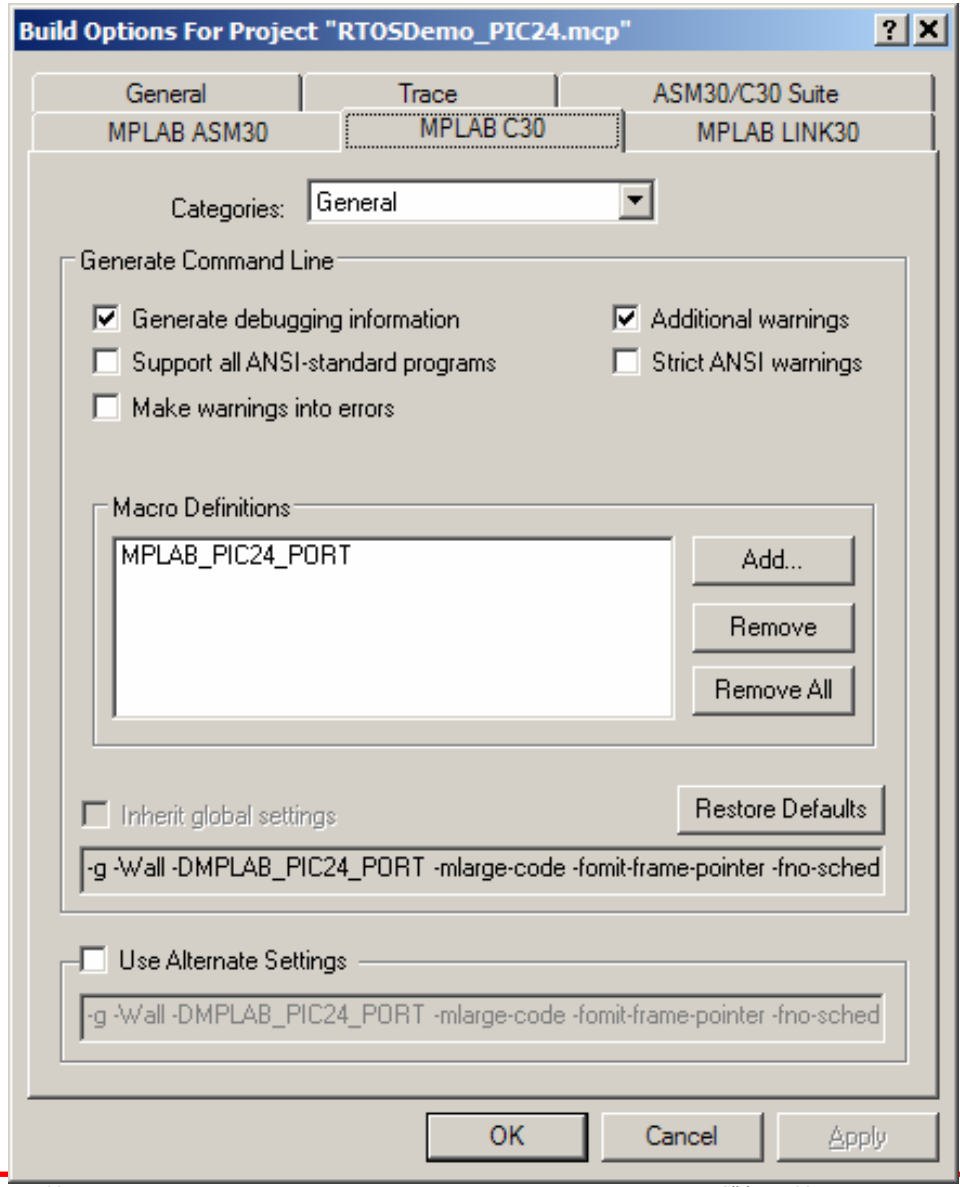
- 主要步骤
  i. 创建项目
  ii. 配置项目
  iii. 编写任务和ISR!
  iv. 编写main()：
    i. 设置硬件
    ii. 创建队列
    iii. 创建任务
    iv. 启动调度程序

“其他”任务

同时执行的程序

步骤 i – 创建项目





存储器管理

- 每当发生以下事件时实时内核需要存储器：
  - 任务创建时
  - 队列创建时
  - 信号量创建时
- 每一个系统的要求是不同的
- 提供了三个范例：
  - Heap_1.c – 简单，存储区不能释放
  - Heap_2.c – 存储区可被释放但也可分段
  - Heap_3.c – 依赖于标准的malloc/free

命名约定

```
/* prv = private to file, not an API function.                    */

void prvATask( void *pvParameters ) /* pv=pointer to void. */
{
unsigned portSHORT usValue = 0; /* us = unsigned short        */
portBASE_TYPE xReturn;          /* x = non standard type      */

    for( ;; )
    {
        xReturn = xQueueSend( pxQueueParameters->xQueue,
                              &usValue,
                              0 );
    }

    /* c = char,
       l = long. */
}
```

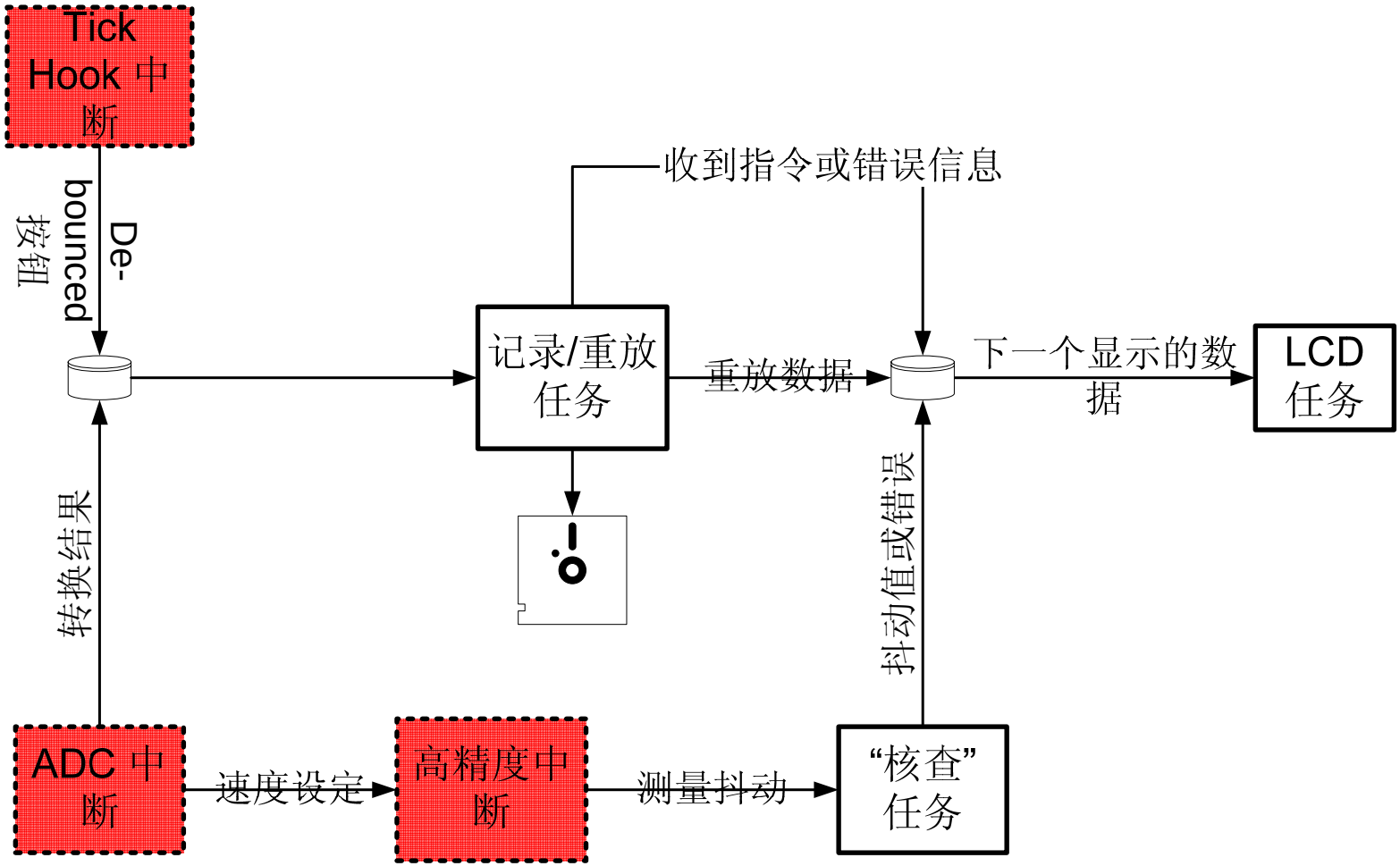
步骤 ii –配置项目

```
#define configUSE_PREEMPTION            1
#define configUSE_IDLE_HOOK             1
#define configUSE_TICK_HOOK             1
#define configTICK_RATE_HZ              1000
#define configCPU_CLOCK_HZ              16000000UL
#define configMAX_PRIORITIES            4
#define configMINIMAL_STACK_SIZE        105
#define configTOTAL_HEAP_SIZE           3800
#define configMAX_TASK_NAME_LEN         4
#define configUSE_TRACE_FACILITY        0
#define configUSE_16_BIT_TICKS          1
#define configIDLE_SHOULD_YIELD         1
```

步骤 ii –配置项目

```
/* Co-routine definitions. */
#define configUSE_CO_ROUTINES        1
#define configMAX_CO_ROUTINE_PRIORITIES 2

/* Set the following definitions to 1 to include
the API function, or zero to exclude the API
function. */

#define INCLUDE_vTaskPrioritySet          1
#define INCLUDE_uxTaskPriorityGet         0
#define INCLUDE_vTaskDelete               0
#define INCLUDE_vTaskCleanUpResources     0
#define INCLUDE_vTaskSuspend              1
#define INCLUDE_vTaskDelayUntil           1
#define INCLUDE_vTaskDelay                1

#define configKERNEL_INTERRUPT_PRIORITY 0x01
```

主要功能



写任务 – 例如记录任务

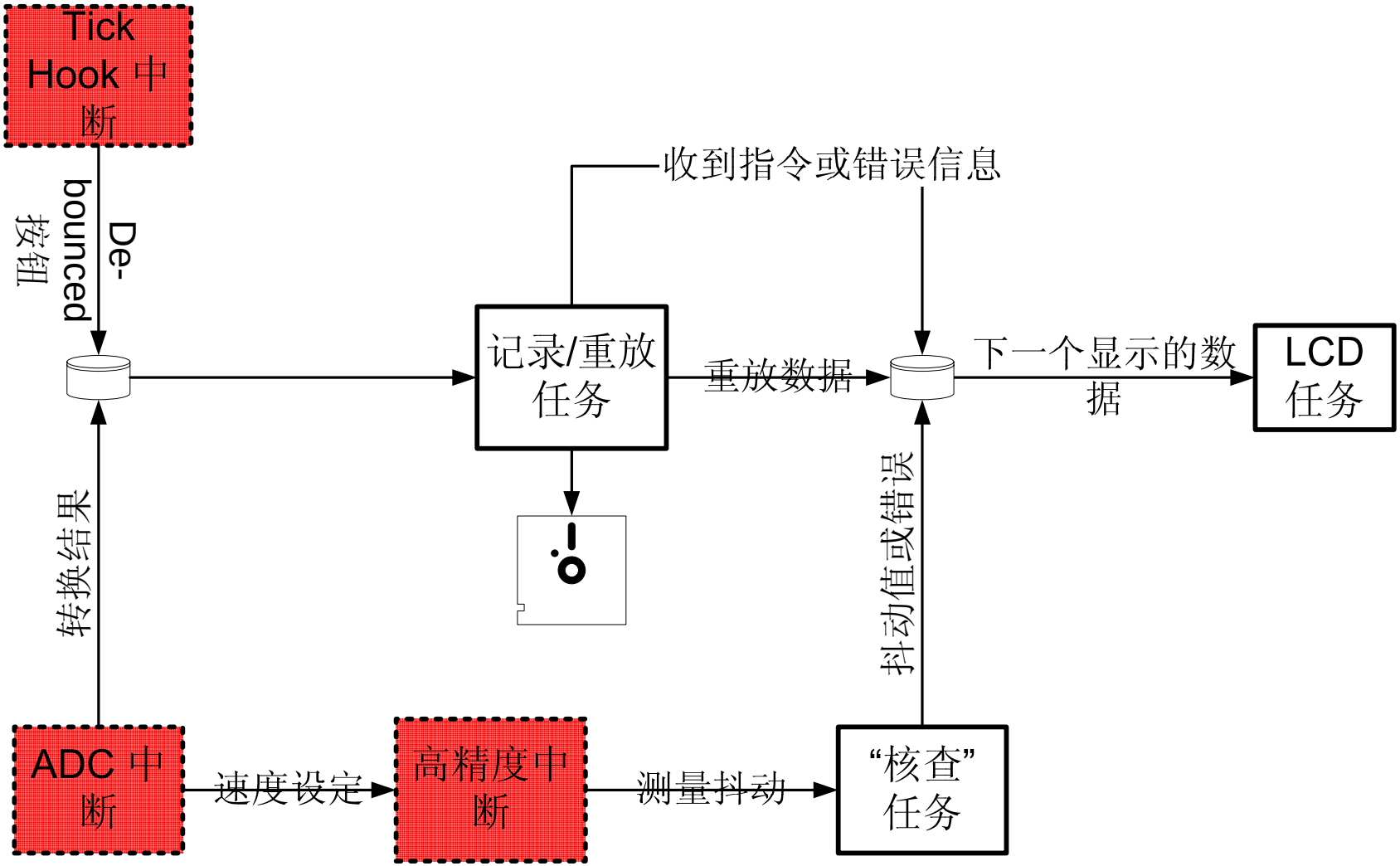
```
static void vLoggingTask( void *pvParameters )
{
xLoggingMessage xMessage;

    /* Initialise the card, if one is inserted. */
    while( !FAT16Init() )
    {
        /* Could not initialise the card.  Wait a bit then try again. */
        vTaskDelay( logINIT_DELAY );
    }

    for( ;; )
    {
        /* Wait for data or a command to be sent to this task. */
        xQueueReceive( xLoggingQueue, &xMessage, portMAX_DELAY );

        switch( xMessage.usCommand )
        {
            case logSTART_LOG    :   prvProcessStartLogCommand();
                                     break;
            case logSTOP_LOG     :   prvProcessStopLogCommand();
                                     break;
            case logREPLAY_LOG   :   prvProcessReplayLogCommand();
                                     break;
            case logVALUE        :   prvProcessLogValue( xMessage.usValue );
                                     break;
        }
    }
}
```

写任务 – 例如记录任务

```
static void vLoggingTask( void *pvParameters )
{
xLoggingMessage xMessage;

    /* Initialise the card, if one is ir
    while( !FAT16Init() )
    {
        /* Could not initialise the card
        vTaskDelay( logINIT_DELAY );
    }

    for( ;; )
    {
        /* Wait for data or a command to be sent to this task. */
        xQueueReceive( xLoggingQueue, &xMessage, portMAX_DELAY );

        switch( xMessage.usCommand )
        {
            case logSTART_LOG    :   prvProcessStartLogCommand();
                                     break;
            case logSTOP_LOG     :   prvProcessStopLogCommand();
                                     break;
            case logREPLAY_LOG   :   prvProcessReplayLogCommand();
                                     break;
            case logVALUE        :   prvProcessLogValue( xMessage.usValue );
                                     break;
        }
    }
}
```

```
typedef struct
{
    unsigned portSHORT usCommand;
    unsigned portSHORT usValue;
} xLoggingMessage;
```

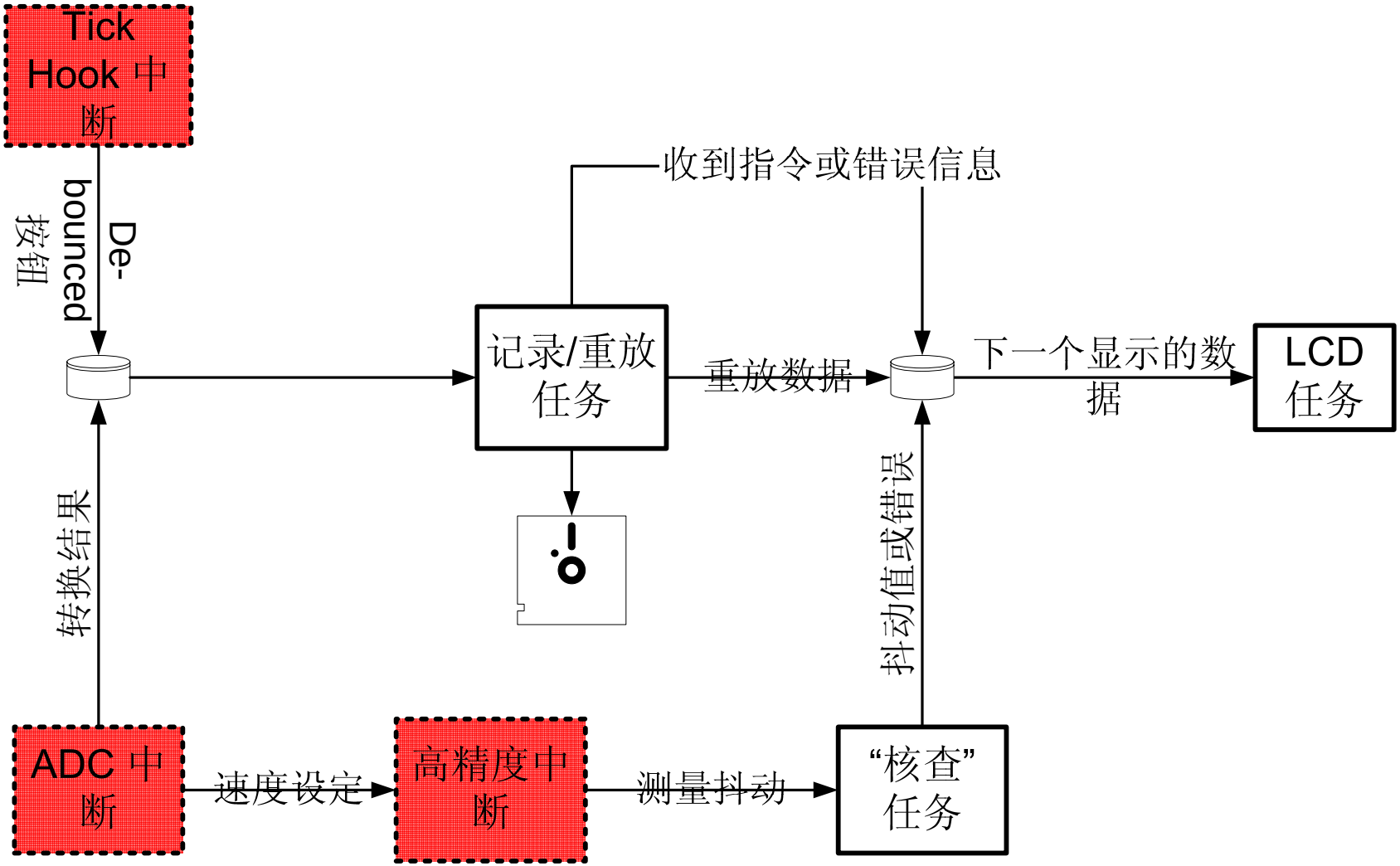
主要功能


Tick Hook 中断
按钮 De-bounced
收到指令或错误信息
记录/重放任务
重放数据
下一个显示的数据
LCD 任务
转换结果
抖动值或错误
ADC 中断
速度设定
高精度中断
测量抖动
“核查”任务

步骤 iii – 写任务
例如核查任务

```
static void vCheckTask( void *pvParameters )
{
xLCDMessage xMessage = { 0, cStringBuffer }; /* Other vars removed for brevity. */

    xLastExecutionTime = xTaskGetTickCount();

    for( ;; )
    {
        vTaskDelayUntil( &xLastExecutionTime, mainCHECK_TASK_PERIOD );

        if( ( xAreComTestTasksStillRunning() != pdTRUE ) ||
            ( xAreSemaphoreTasksStillRunning() != pdTRUE ) )
        {
            usErrorDetected = pdTRUE;
        }

        if( usErrorDetected == pdFALSE )
        {
            sprintf( cStringBuffer, "%dns max jitter",
                ( portSHORT ) ( usMaxJitter - mainEXPECTED_CLOCKS_BETWEEN_INTERRUPTS )
                * mainNS_PER_CLOCK );
        }
        else
        {
            xMessage.pcMessage = "FAIL";
        }

        xQueueSend( xLCDQueue, &xMessage, portMAX_DELAY );
    }
}
```

主要功能


Tick Hook 中断
De-bounced 按钮
收到指令或错误信息
记录/重放任务
重放数据
下一个显示的数据
LCD 任务
转换结果
抖动值或错误
ADC 中断
速度设定
高精度中断
测量抖动
“核查”任务

使用 Tick Hook
启动 ADC 转换（+ 消抖（debouncing））

```
void vApplicationTickHook( void )
{
static unsigned portSHORT usADCCount = 0, usButtonCount;
extern xQueueHandle xLoggingQueue;
xLoggingMessage xMessage;

    /* Start an ADC conversion every 200 milliseconds.
    The ADC conversion result is used to set the speed
    of the LED's (motor ) on the Explorer 16 board. */
    usADCCount++;
    if( usADCCount >= ( 200 * portTICK_RATE_MS ) )
    {
        /* Start an ADC conversion. */
        AD1CON1bits.SAMP = 1;

        /* Reset the count so we know when to start
        the next conversion. */
        usADCCount = 0;
    }

    /* Scan the buttons every 10 milliseconds  */
```

步骤 iv – 创建任务和队列
例如记录任务和队列

```
void vStartLoggingTask( void )
{
    /* Create the queue used to send commands
    and data to the logging task. */
    xLoggingQueue = xQueueCreate( logQUEUE_SIZE,
                                  sizeof( xLoggingMessage ) );

    /* Then create the task itself. */
    xTaskCreate(
        vLoggingTask,/* Points to function that implements task. */
        "Logging", /* Name of the task.  Useful for debugging.   */
        configMINIMAL_STACK_SIZE * 2, /* Size of allocated stack.*/
        NULL,      /* Parameters to be passed to the task.       */
        tskIDLE_PRIORITY, /* Priority of the task.               */
        NULL );    /* Can be used to obtain a handle to the task */
}
```

步骤 iv – 启动调度程序

```
int main( void )
{
    /* Configure any hardware required for this demo. */
    prvSetupHardware();

    /* Create the tasks. */
    prvCreateAllTasks();

    /* Configure interrupts.  Note the tick is configured
    automatically when the scheduler is started. */
    prvConfigureInterrupts();

    /* Finally start the scheduler. */
    vTaskStartScheduler();

    /* Will only reach here if there is insufficient
    heap available to start the scheduler. */
    return 0;
}
```

Lab #3 – 目标

- 通过试验熟悉讨论过的概念
- 熟悉完全配置的多任务应用

Lab #3 – 结论

回顾**Lab #3**中的问题

结论

- 我们已了解到:
  - 为什么传统小型应用的技术未必适合较大型的应用
  - 使用小型RTOS如何可以提供一种合适的选择
  - … 但使用RTOS需要使用一种不同的方式进行应用设计

重温学习目标

- 何为实时内核?
  - 我们已经了解调度策略如何定义内核的特性
- 实时内核何时是有用的?
  - 混合处理需求、最终期限、 可扩展性、抽象性、可测试性等

重温学习目标

- 在写多任务应用程序时需要了解哪些内容?
  - 任务是如何运行、如何处理互斥、任务间通信和同步等等。
- 如何开始使用一个实时内核?
  - 共需通过四个步骤来启动一个基于FreeRTOS.org的应用

最后的结论

- 总是为手头的工作选择最佳的工具!

参考文献

有关 FreeRTOS.org 源代码和文档，可登录:
www.FreeRTOS.org

有关 SAFERTOS 的信息，可登录:
www.SafeRTOS.com

有关商业授权和开发服务的具体信息，可登录:
www.HighIntegritySystems.com

感谢参与本课程!

本课程使用的开发工具

- 实验练习
  - Explorer 16 开发板 (DM240001)
  - 用于SD/MMC (AC164122)的 PICtail™ 开发板

商标

Microchip的名称和徽标组合、**Microchip**徽标、**Accuron**、**dsPIC**、**KeeLoq**、**KeeLoq**徽标、**microID**、**MPLAB**、**PIC**、**PICmicro**、**PICSTART**、**PRO MATE**、**rfPIC**和**SmartShunt**均为**Microchip Technology Incorporated**在美国和其他国家或地区的注册商标。

AmpLab、**FilterLab**、**Linear Active Thermistor**、**Migratable Memory**、**MXDEV**、**MXLAB**、**SEEVAL**、**SmartSensor**和**The Embedded Control Solutions Company**均为**Microchip Technology Incorporated**在美国的注册商标。

Analog-for-the-Digital Age、**Application Maestro**、**CodeGuard**、**dsPICDEM**、**dsPICDEM.net**、**dsPICworks**、**dsSPEAK**、**ECAN**、**ECONOMONITOR**、**FanSense**、**FlexROM**、**fuzzyLAB**、**In-Circuit Serial Programming**、**ICSP**、**ICEPIC**、**Mindi**、**MiWi**、**MPASM**、**MPLAB Certified**徽标、**MPLIB**、**MPLINK**、**PICkit**、**PICDEM**、**PICDEM.net**、**PICLAB**、**PICtail**、**PowerCal**、**PowerInfo**、**PowerMate**、**PowerTool**、**REAL ICE**、**rfLAB**、**Select Mode**、**Smart Serial**、**SmartTel**、**Total Endurance**、**UNI/O**、**WiperLock**和**ZENA**均为**Microchip Technology Incorporated**在美国和其他国家或地区的商标。

SQTP是**Microchip Technology Incorporated**在美国的服务标记。

在此提及的所有其他商标均为各持有公司所有。