Embedded Systems / Eingebettete Systeme

Studiengang Informatik
Campus Minden

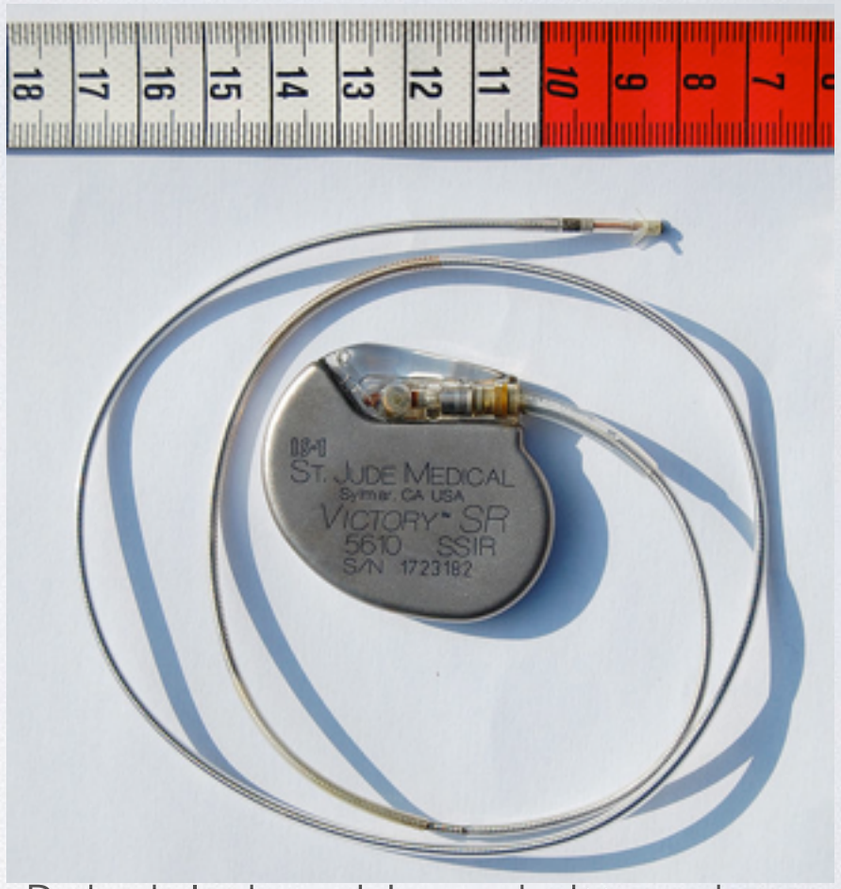
Matthias König

FH Bielefeld
University of
Applied Sciences

Beispiel einer Anwendung: Herzschrittmacher

- Batterie, Impulsgeber, Steuerelektronik, Elektrode
- Übliche Einstellung des Geräts ist drahtlos
- Elektrode als Sensor (für EKG) und Aktor (elektrischer Impuls)
- Auswertung eines Triggersignals aus EKG für Herzschlag

[Quelle: Wikipedia, Herzschrittmacher]



Beispiel eines Herzschrittmachers

[Quelle: Wikipedia (Fruitsmaak): http://de.wikipedia.org/w/index.php?title=Datei:St_Jude_Medical_pacemaker_with_ruler.jpg&filetimestamp=20071013142827]

Wiederholung

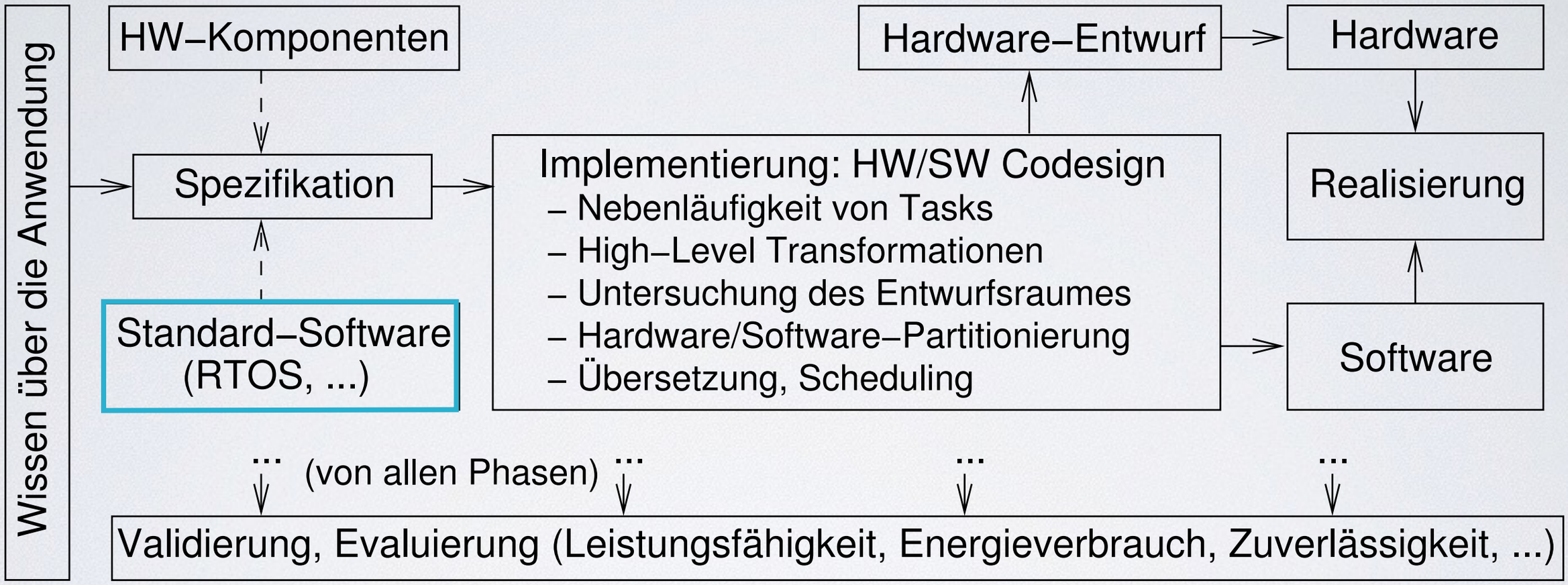
Schedulingalgorithmen für Echtzeit

- Unterscheidung nach
  - harte und weiche Deadlines
  - harte Deadlines weiterhin nach
    - periodisch/aperiodisch
    - präemptiv/nicht-präemptiv
    - statisch/dynamisch
- Beispiele für Verfahren: EDD, EDF, LL, RMS

[Quelle: Marwedel, Eingebettete Systeme]

Grundlagen von Standard-Software

Hardware/Software Codedesign



Entwurf Eingebetteter Systeme (nach Marwedel)

[Quelle: Marwedel, Eingebettete Systeme]

Standard-Software

- Wiederverwendbare Standardkomponenten
  - Hardware-Abstraction-Layer
  - Middleware
  - (Real Time) Operating Systems
- Oft auch als Intellectual Property IP bezeichnet (Geistiges Eigentum)

Hardware-Abstraction-Layer HAL

- Schicht zwischen (Anwendungs-)Software und Hardware
- "Standardisierte" Programmierschnittstelle unabhängig von unterliegender Hardware
- Kapselung von Hardware-spezifischen Programmcode
- Erlaubt einfache Portierung auf andere Hardware

Middleware

- Nutzbare Zusatzsoftware für Anwendungsprogrammierung
- Zwischenschicht/Vermittlungssoftware zwischen Betriebssystem und Anwendung
- Oft gemeint: Kommunikationsbibliotheken

Beispiel einer Middleware: CORBA

- Common Object Request Broker Architecture CORBA
- Standardisierte Schnittstellen auf entfernte Objekte
- Klient kommuniziert über lokale *Stubs* mit Objekt.
- Informationen an/über Objekt an *Object Request Broker* ORB
- ORB Lokalisierung des Objekts bekannt
- Nachrichten über Internet Inter-ORB Protocol
- Echtzeit-Erweiterung RT-CORBA verfügbar

[Quelle: Marwedel, Eingebettete Systeme]

Eingebettete Betriebssysteme

- Allgemeine Anforderungen
  - Konfigurierbarkeit (zur Compile-Zeit)
  - Abarbeitung von Peripherie in Tasks und nicht im Kernel
  - Schutzmechanismen für Sicherheit, aber normalerweise nicht für direkte Speicherzugriffe
  - Direktes Nutzen von Interrupts

[Quelle: Marwedel, Eingebettete Systeme]

Echtzeitbetriebssysteme RTOS

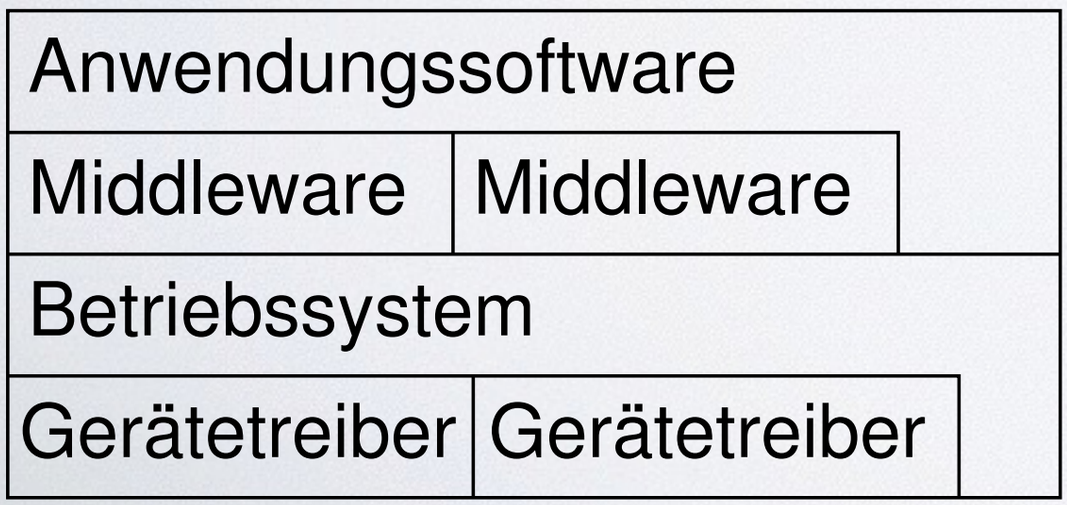
- Anforderungen an Echtzeitbetriebssystem (real-time operating system):
  - Vorhersagbares Zeitverhalten
    - Garantierte obere Schranke für Laufzeit
    - Deterministisches Scheduling
  - Zeitverwaltung/Scheduler bei Betriebssystem
    - Priorisierung und hohe Präzision für Zeitdiensten
    - Schnelles/vorhersagbares Umschalten (~Sekundenbruchteile)

[Quelle: Marwedel, Eingebettete Systeme]

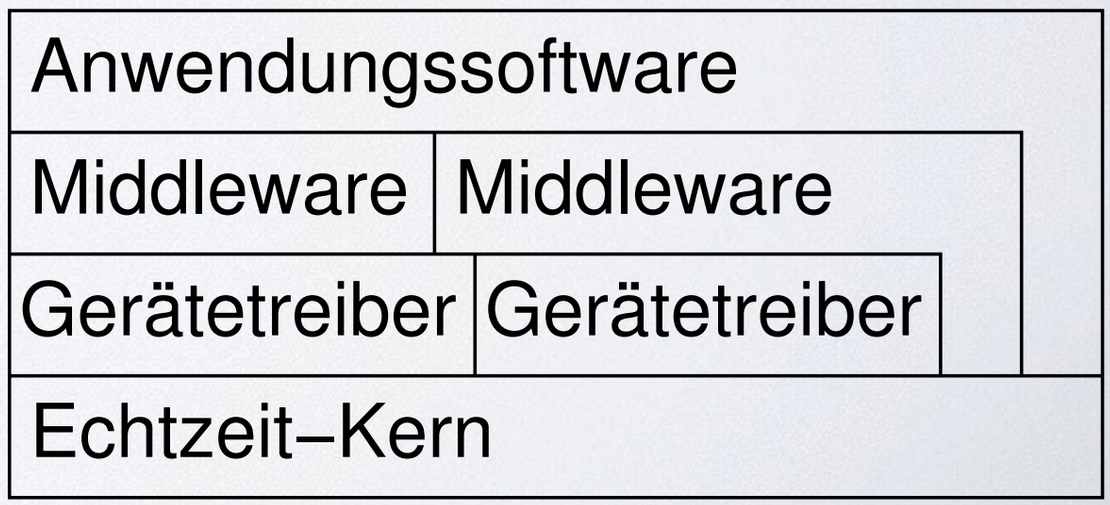
Echtzeitbetriebssysteme RTOS

- Gerätetreiber außerhalbs des Echtzeitbetriebssystem-Kerns
- Aufgaben des Kerns:
  - Task-, Speicher- und Zeitverwaltung
  - Intertask-Synchronisation und -Kommunikation

Allzweck-Betriebssystem



Echtzeitbetriebssystem



[Quelle: Marwedel, Eingebettete Systeme]

Echtzeitbetriebssysteme RTOS

- Vorteile:
  - Vorhandene Funktionalität, z.B.
    - Scheduler
    - Timer- und Interrupt-Handling
  - Gegebene Modularität
  - Portierbar- und Wartbarkeit

RTOS Standard: POSIX

- POSIX (Portable Operating System Interface) mit Echtzeit-Erweiterung
  - Internationaler Standard ISO/IEC-9945 bzw. IEEE 1003.1-2008
  - Vorgabe einer API für Unix-ähnliche Betriebssysteme
  - Echtzeit-Erweiterungen in Teilen
    - IEEE Std 1003.1b-1993 Realtime Extension
    - IEEE Std 1003.1c-1995 Threads
    - IEEE Std 1003.1d-1999 Additional Realtime Extensions
    - IEEE Std 1003.1j-2000 Advanced Realtime Extensions
    - IEEE Std 1003.1q-2000 Tracing

[Quelle: The Open Group, POSIX FAQ]

RTOS Standard: POSIX

The Open Group Base Specifications Issue 7
IEEE Std 1003.1, 2013 Edition

NAME

pthread_setschedprio - dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

```
[TPS] #include <pthread.h>

int pthread_setschedprio(pthread_t thread, int prio);
```

DESCRIPTION

The *pthread_setschedprio*() function shall set the scheduling priority for the thread whose thread ID is given by *thread* to the value given by *prio*. See *Scheduling Policies* for a description on how this function call affects the ordering of the thread in the thread list for its new priority.

If the *pthread_setschedprio*() function fails, the scheduling priority of the target thread shall not be changed.

RETURN VALUE

If successful, the *pthread_setschedprio*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setschedprio*() function may fail if:

[EINVAL]
The value of *prio* is invalid for the scheduling policy of the specified thread.
[EPERM]
The caller does not have appropriate privileges to set the scheduling priority of the specified thread.

The *pthread_setschedprio*() function shall not return an error code of [EINTR].

[Quelle:: http://pubs.opengroup.org/onlinepubs/9699919799/]

RTOS-Standard: OSEK/VDX

- Gremium: Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (OSEK) / Vehicle Distributed eXecutive (VDX)

- ISO 17356: Standard für Steuergeräte im Fahrzeug

  - Skalierbarkeit durch sogenannte *Conformance Classes* unterschiedlicher Komplexität, z.B. BCC1 für maximal acht Tasks und eine geteilte Ressource

  - OSEK-OS, u.a. statischer Kernel: keine dynamische Speicherallokation und keine dynamische Generierung von Tasks

  - eigene Beschreibungssprache OSEK Implementation Language

- Offen: z.B. openOSEK (www.openosek.org)

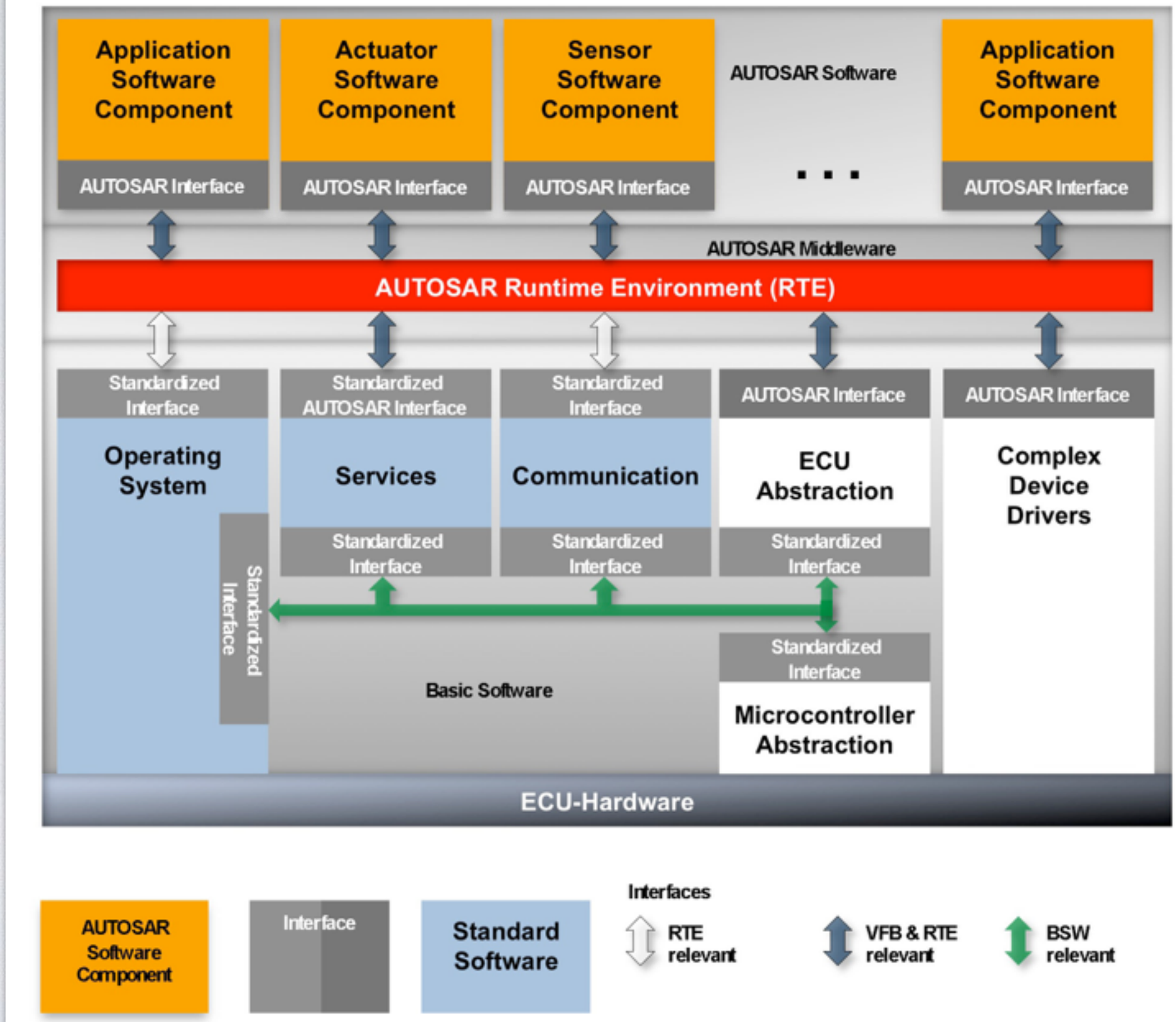
[Quelle: Buttazzo, Hard Real-Time Computing Systems]

RTOS-Standard: AUTOSAR

- Gremium für AUTomotive Open System ARchitecture
- Fortsetzung von OSEK/VDX
- Beschreibung u.a.
  - Trennung von Basis-Software (für Infrastruktur des Steuergeräts) und Anwendungssoftware
  - Kommunikation der Basis- und Anwendungssoftware über virtuellen funktionalen Bus (Schnittstellen-API)
  - Runtime Environment (Middleware)

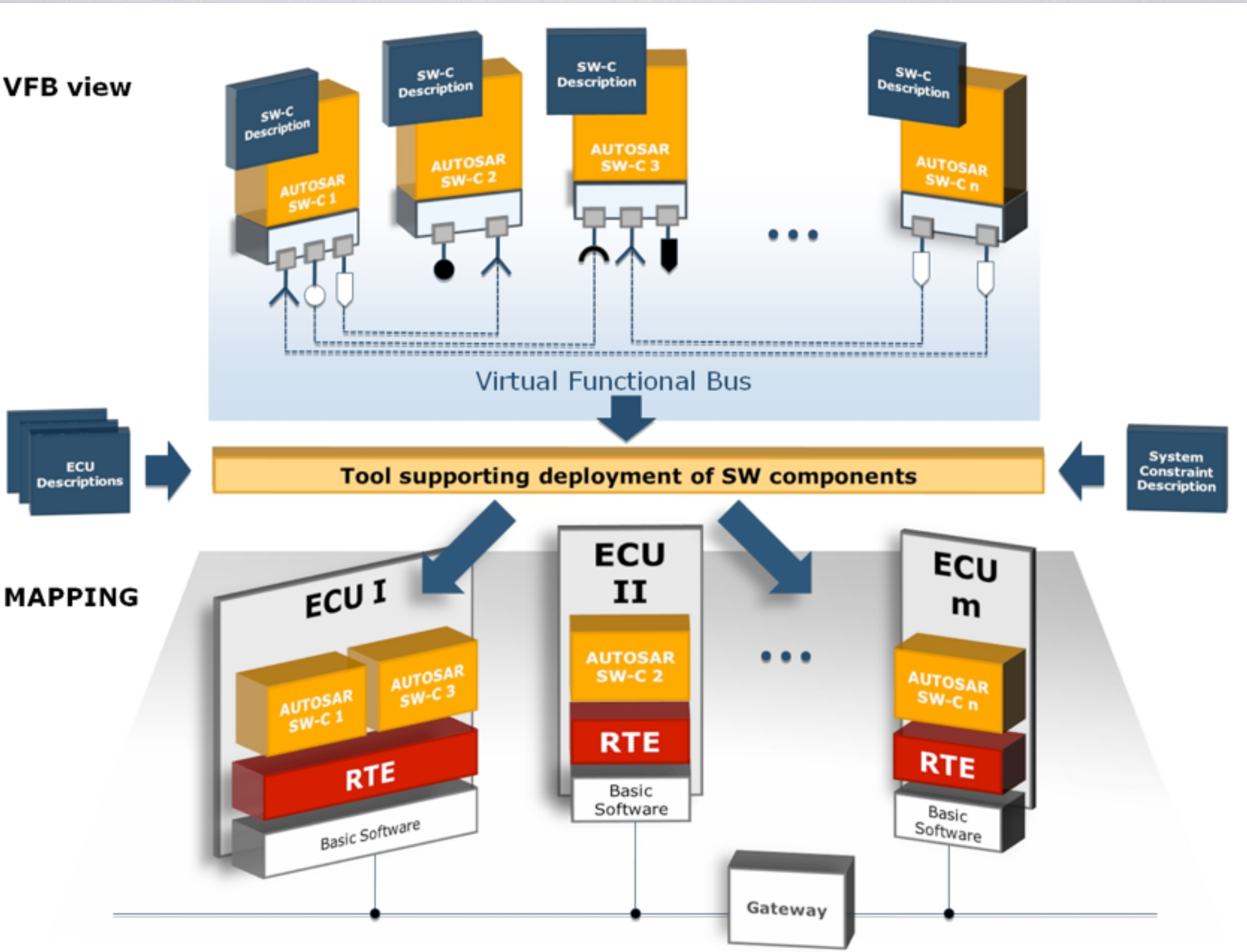
[Quelle: AUTOSAR]

AUTOSAR



[Quelle: http://www.autosar.org/gfx/AUTOSAR_ECU_softw_arch_b.jpg]

AUTOSAR



[Quelle: http://www.autosar.org/gfx/AUTOSAR_TechnicalOverview_b.jpg]

Echtzeiterweiterungen von Betriebssystemen

Hybride Betriebssysteme für Echtzeit- und andere Tasks

1. Komponenten-Austausch beim Standard-Betriebssystem, z.B. des Schedulers

   - Problem: Abhängigkeiten von Tasks nicht berücksichtigt

2. Standard-Betriebssystem läuft als Task eines Echtzeitkerns

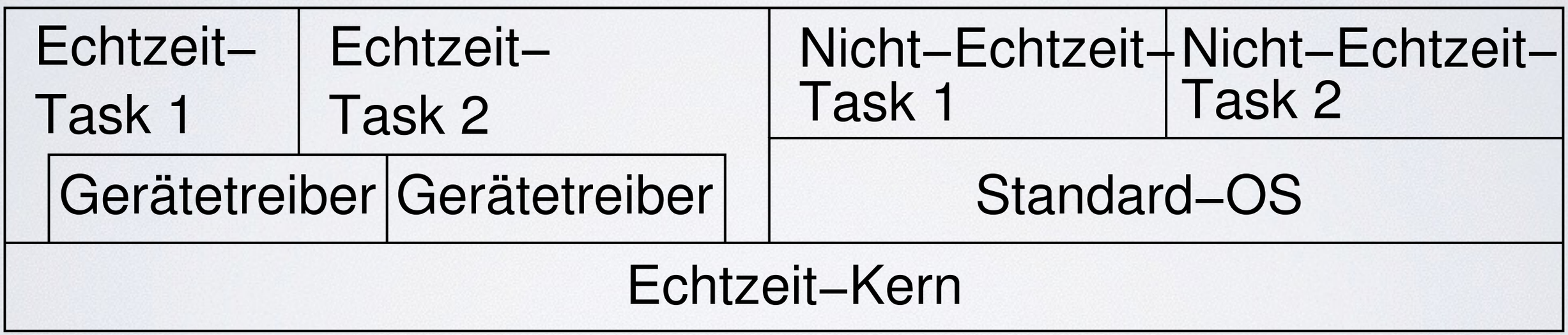
   - Probleme beim Standard-Betriebssystem beeinflussen den Echtzeitkern nicht.

   - Aufteilung von Geräten zwischen beiden Systemen ist möglicherweise notwendig.

[Quelle: Marwedel, Eingebettete Systeme]

Echtzeiterweiterungen von Betriebssystemen

Standard-Betriebssystem als Task eines Echtzeitskerns



[Quelle: Marwedel, Eingebettete Systeme]

Linux für Eingebettete Systeme

- Beide Ansätze hybrider Echtzeiterweiterungen vorhanden:
  - PREEMPT_RT (Scheduler ausgetauscht)
    - Präemptiver Kernel (aus User-Mode kann Kernel-Mode unterbrochen werden)
  - RTLinux
    - Linux als Task eines Real-Time Kernels

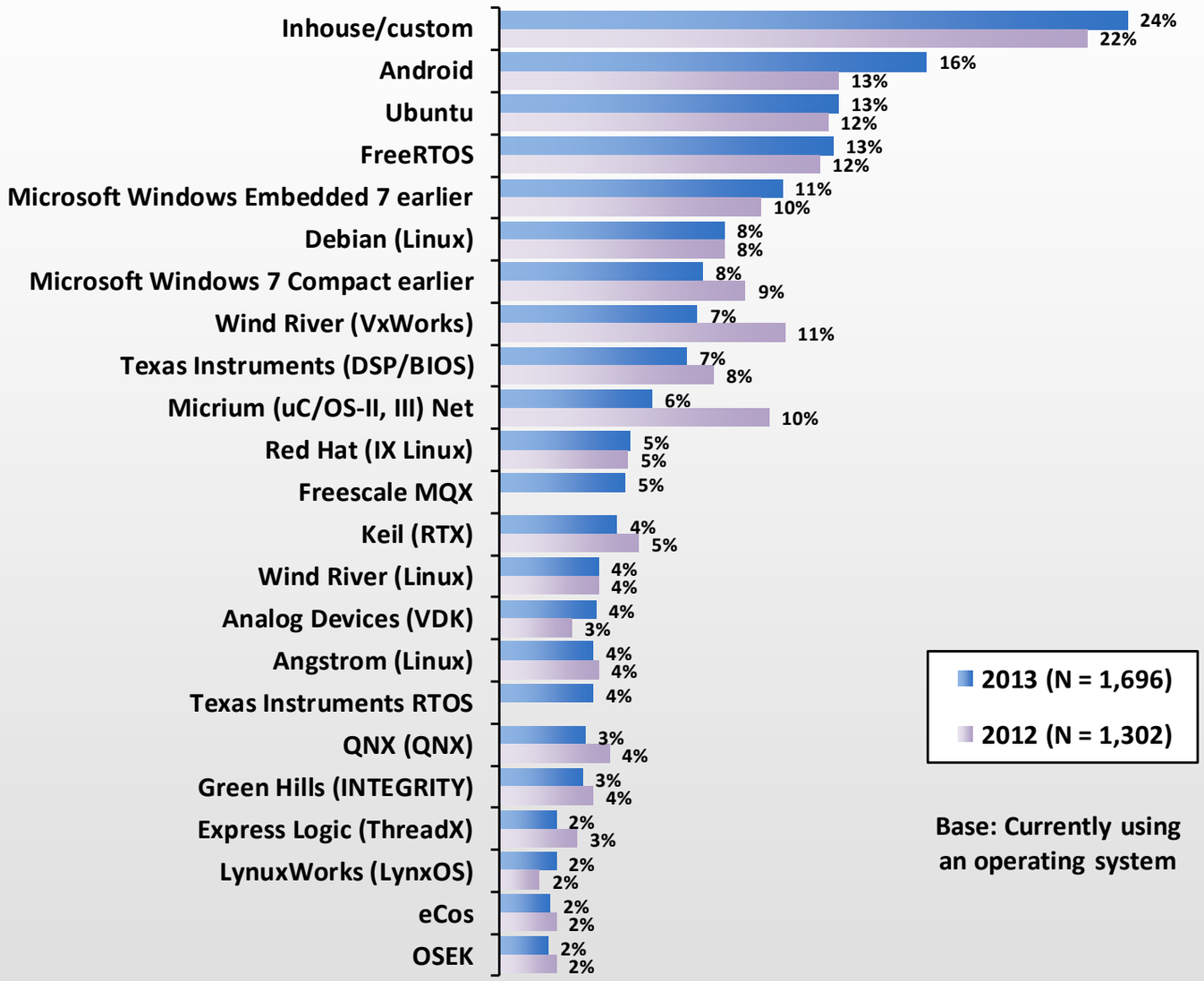
Kommerzielle RTOS

- Vielzahl Anbieter von Real-time Operating Systems , oft für spezielle Hardware-Plattformen

- Eine Auswahl:

  - FreeRTOS
  - MQX von Freescale
  - VxWorks von Wind River
  - RTX von Keil
  - QNX von QNX
  - Texas Instruments RTOS

Please select ALL of the operating systems you are currently using.



Base: Currently using an operating system

Only Operating Systems that had 2% or more are shown.

designwest
April 22-25, 2013
McEnery Convention Center
San Jose, CA

[Quelle: UBM Tech, 2013 Embedded Market Study]

RTOS: Beispiel Festlegung Tasks bei MQX

```
TASK_TEMPLATE_STRUCT MQX_template_list[] = {
{INIT_TASK, init_task, 1500, 9, "init", MQX_AUTO_START_TASK, 0, 0},
{LED1_TASK, led1_task, 1500, 10, "led1", 0, 0, 0},
{LED2_TASK, led2_task, 1500, 11, "led2", 0, 0, 0},
{LED3_TASK, led3_task, 1500, 12, "led3", 0, 0, 0},
{LED4_TASK, led4_task, 1500, 13, "led4", 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0}
};//Eintrag: ID, Callback, Stackgröße, Priorität, Name, Startupeintrag
```

```
task_id = _task_create(0, LED1_TASK, 0);
if (task_id == MQX_NULL_TASK_ID) ...
```

```
void led4_task(uint_32 initial_data){
  while (TRUE){}
  _time_delay(4444);
  SetOutput(4,value);
  value = value^1; }
```

[Quelle: Ramos, Freescale,AN3905]

FreeRTOS

- FreeRTOS
  - Unterstützung verschiedener Hardware-Plattformen
  - Single-Core, (Multi-Core ist selbst zu implementieren.)
  - Kleiner Kernel (ab 5 KB)
  - Open Source unter GPL, aber nutzbar für kommerzielle Projekte
- OpenRTOS
  - Gleicher Code wie FreeRTOS mit kommerzieller Lizenz
  - Support und zusätzliche Bibliotheken
- SafeRTOS
  - Gleicher Ausgangscode aber zertifiziert für IEC 61508 SIL 3

[Quelle: FreeRTOS]

FreeRTOS: Features

- Präemptives und kooperatives Scheduling
- Flexibles Management von Taskprioritäten
- Queues
- Semaphoren und Mutexes
- Tick und Idle Hook Funktionen
- Stack Overflow checking

[Quelle: FreeRTOS]

Develop – freertos_blinky/example/src/freertos_blinky.c – LPCXpresso – /Users/matthias/Documents/LPCXpresso_7.0.0/workspace

Quick Access | Develop | Debug

Project E | Periphera | Registers

- freertos_blinky
  - Binaries
  - Includes
  - example
    - inc
    - src
      - cr_startup_lpc175x_6x.c
      - freertos_blinky.c
      - sysinit.c
  - freertos
    - inc
      - croutine.h
      - FreeRTOS.h
      - list.h
      - mpu_wrappers.h
      - portable.h
      - portmacro.h
      - projdefs.h
      - queue.h
      - semphr.h
      - StackMacros.h
      - task.h
      - timers.h
    - src
      - FreeRTOSCommonHooks.c
      - FreeRTOSCommonHooks.h
      - heap_3.c
      - list.c
      - port.c
      - queue.c
      - tasks.c
  - Debug
  - freertos_blinky Debug.launch
  - freertos_blinky Release.launch

Einbindung FreeRTOS

freertos_blinky.c | Welcome

```
 2  * @brief FreeRTOS Blinky example
31
32 #include "board.h"
33 #include "FreeRTOS.h"
34 #include "task.h"
35
36 /*****************************************************************************
37  * Private types/enumerations/variables
38  ****************************************************************************/
39
40 /*****************************************************************************
41  * Public types/enumerations/variables
42  ****************************************************************************/
44 /*****************************************************************************
45  * Private functions
46  ****************************************************************************/
47
48 /* Sets up system hardware */
49 static void prvSetupHardware(void)
50 {
51     SystemCoreClockUpdate();
52     Board_Init();
53
54     /* Initial LED0 state is off */
55     Board_LED_Set(0, false);
56 }
57
58 /* LED1 toggle thread */
59 static void vLEDTask1(void *pvParameters) {
60     bool LedState = false;
61
62     while (1) {
63         Board_LED_Set(0, LedState);
```

Qui | Var | Bre | Out | Ex

Start here

- Import project(s)
- New project...
- Build all projects [Debug]
- Build 'freertos_blinky' [Debug]
- Clean 'freertos_blinky' [Debug]
- Debug 'freertos_blinky' [Debug]
- Edit 'freertos_blinky' project settings
- Import project(s) from XML Description

Console | Problems | Memory | Profile+ | Data Watch | Int Statistic | Int Trace+ | Host Strings | Instruction

CDT Build Console [freertos_blinky]

```
Finished building: ../example/src/sysinit.c

Building target: freertos_blinky.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"/Users/matthias/Documents/LPCXpresso_7.0.0/workspace/lpc_chip_175x_6x/Debug" -L"/Users/matt
Finished building target: freertos_blinky.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "freertos_blinky.axf"; # arm-none-eabi-objcopy -O binary "freertos_blinky.axf" "freertos_blinky.bin" ;
   text    data     bss     dec     hex filename
  18224       8     664   18896    49d0 freertos_blinky.axf


18:57:11 Build Finished (took 2s.860ms)
```

/freertos_blinky/freertos/inc

NXP LPC1769 (freertos_blinky)

FreeRTOS: Task

- Tasks sind einfache C-Funktionen, Prototyp:

```
void aTaskFunction(void *pvParameters);
```

  - Ohne Rückgabewert
  - Mit Zeiger auf übergebene Parameter

- Jeder Task bekommt eigenen Stack.

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task

```
void aTaskFunction(void *pvParameters) {
   // Each instance has own copy of this local variable.
   int aVariable = 0;

   // A task is implemented as infinite loop.
   while (1) {
      // Task code here

   }

   // If task breaks, it should be deleted.
   vTaskDelete( NULL );
}
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task-Generierung

```
// Return pdTrue or errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY
portBASE_TYPE xTaskCreate(
      pdTASK_CODE pvTaskCode, // C function for task
      const signed char * const pcName, // Task name
      unsigned short usStackDepth, // Stack size
      void *pvParameters, // Value passed to task function
      unsigned portBASE_TYPE uxPriority, // Priority of task
      xTaskHandle *pxCreatedTask // Handle of the task
);
```

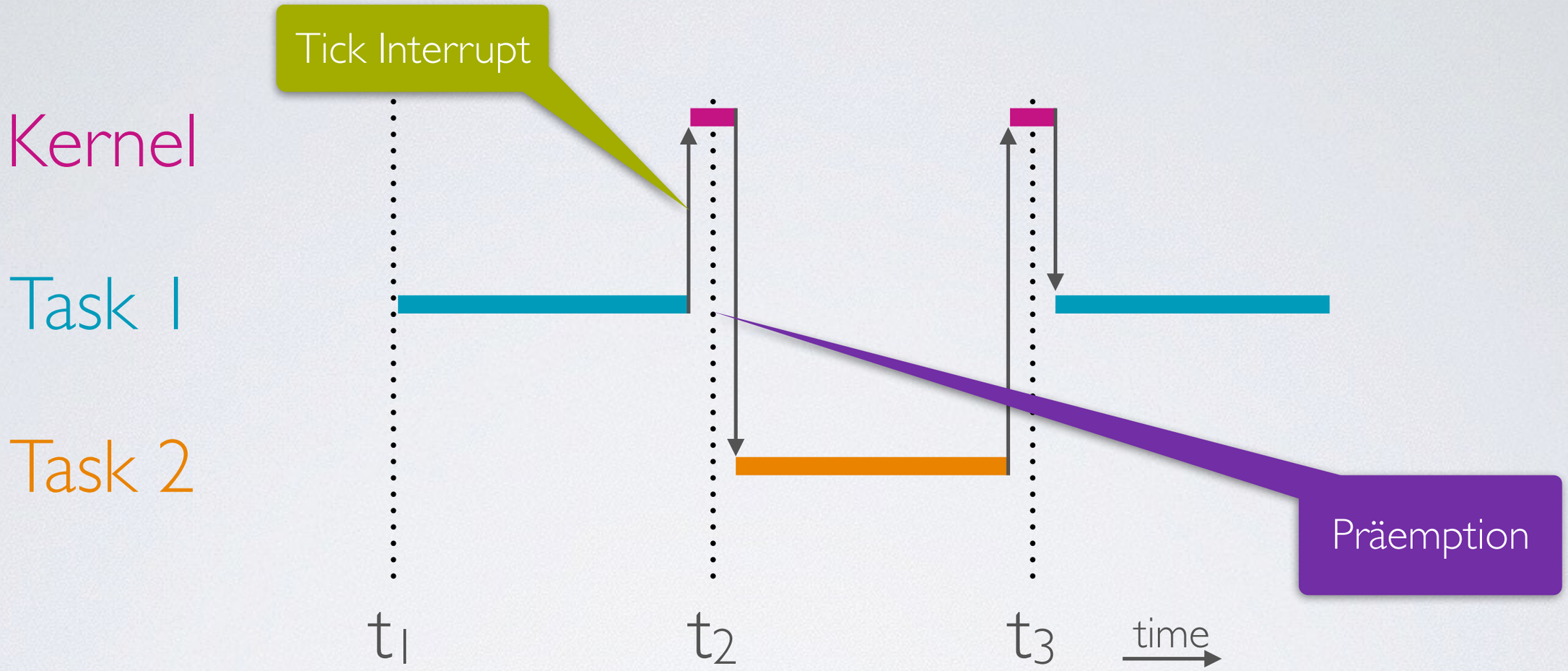
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task-Generierung

```
// Example 1: two tasks with same priority

// Generate task 1
xTaskCreate(aTaskFunction, (signed char *) "task1",
            configMINIMAL_STACK_SIZE, (void *) 1,
            (tskIDLE_PRIORITY + 1UL),
             (xTaskHandle *) NULL);

// Generate task 2
xTaskCreate(aTaskFunction2, (signed char *) "task2",
            configMINIMAL_STACK_SIZE, (void *) 2,
            (tskIDLE_PRIORITY + 1UL),
             (xTaskHandle *) NULL);

// Start the scheduler
vTaskStartScheduler();

// Should never arrive here
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Scheduling (Example 1)



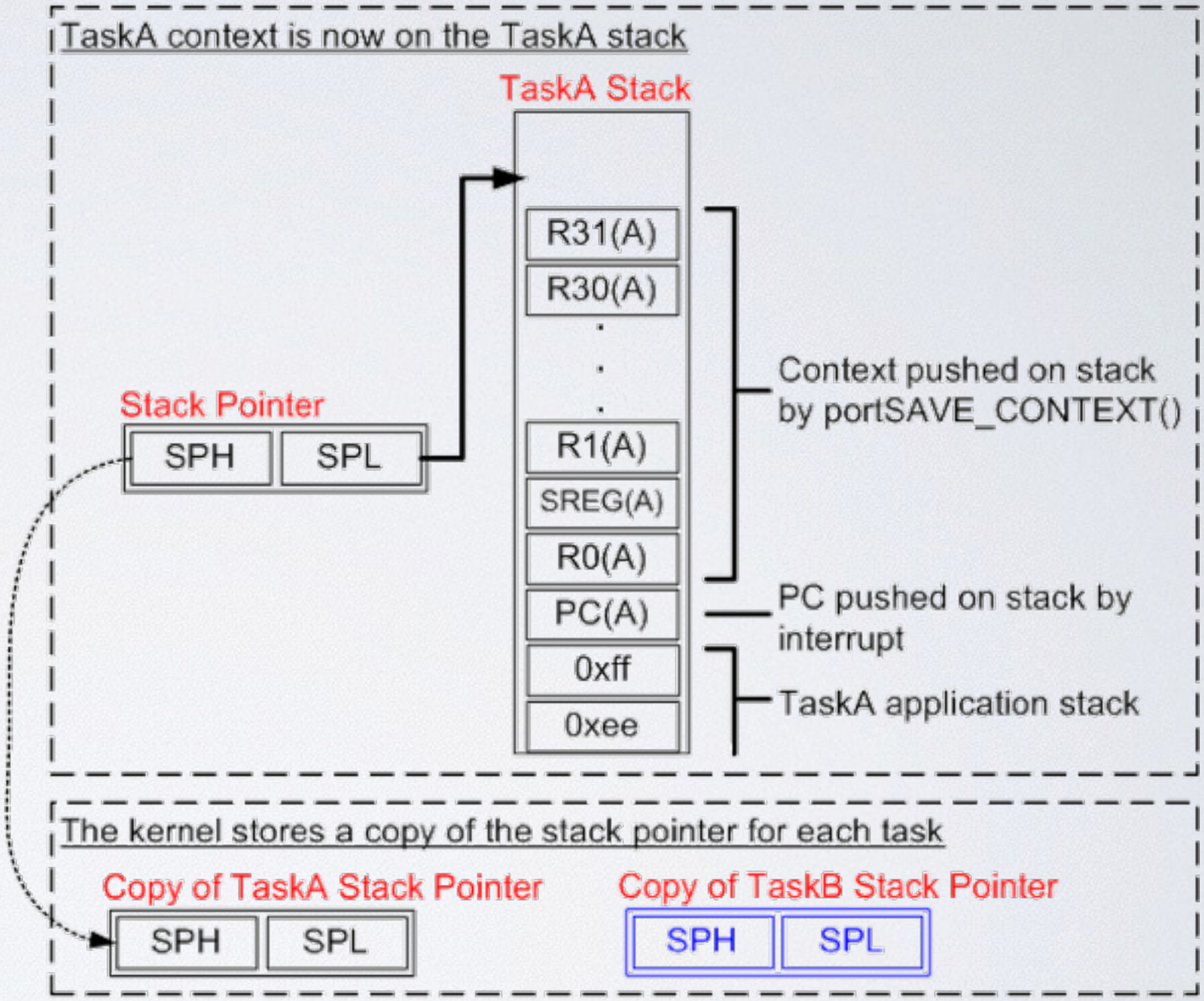
- Zwei Tasks mit gleicher Priorität
- Eine Zeitscheibe / Time Slice für jeden Task
- Overhead für Kernel

Time Slice definiert in Variable `configTICK_RATE_HZ`

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Context Switch (AVR)

```
/* Interrupt service routine
   for the RTOS tick. */
void
SIG_OUTPUT_COMPARE1A(void )
{
    vPortYieldFromTick();
    asm volatile ( "reti" );
}
/*-----------------------------*/

void vPortYieldFromTick( void )
{
    portSAVE_CONTEXT();

    vTaskIncrementTick();
    vTaskSwitchContext();
    portRESTORE_CONTEXT();

    asm volatile ( "ret" );
}
/*-----------------------------*/
```



[Quelle: http://www.freertos.org/implementation/AtoB3.gif]

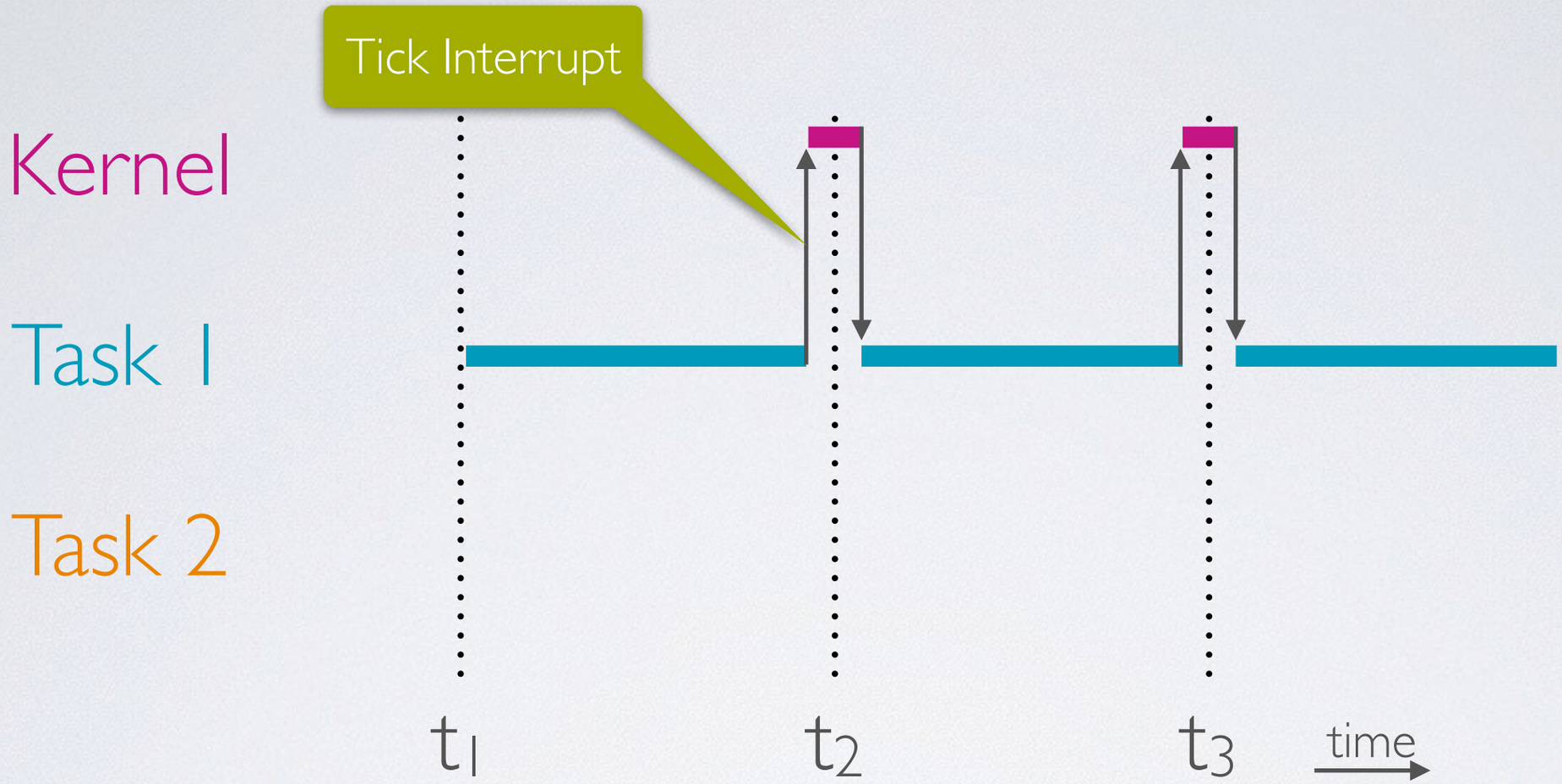
[Quelle: FreeRTOS]

FreeRTOS: Task-Generierung

```
// Example 2: task 1 with higher priority than task 2

// Generate task 1
xTaskCreate(aTaskFunction, (signed char *) "task1",
            configMINIMAL_STACK_SIZE, (void *) 1,
            (tskIDLE_PRIORITY + 2UL),
             (xTaskHandle *) NULL);

// Generate task 2
xTaskCreate(aTaskFunction2, (signed char *) "task2",
            configMINIMAL_STACK_SIZE, (void *) 2,
            (tskIDLE_PRIORITY + 1UL),
             (xTaskHandle *) NULL);

// Start the scheduler
vTaskStartScheduler();

// Should never arrive here
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Scheduling (Example 2)



- Task 1 mit höherer Priorität als Task 2
- Task 2 verhungert.
- Overhead für Kernel

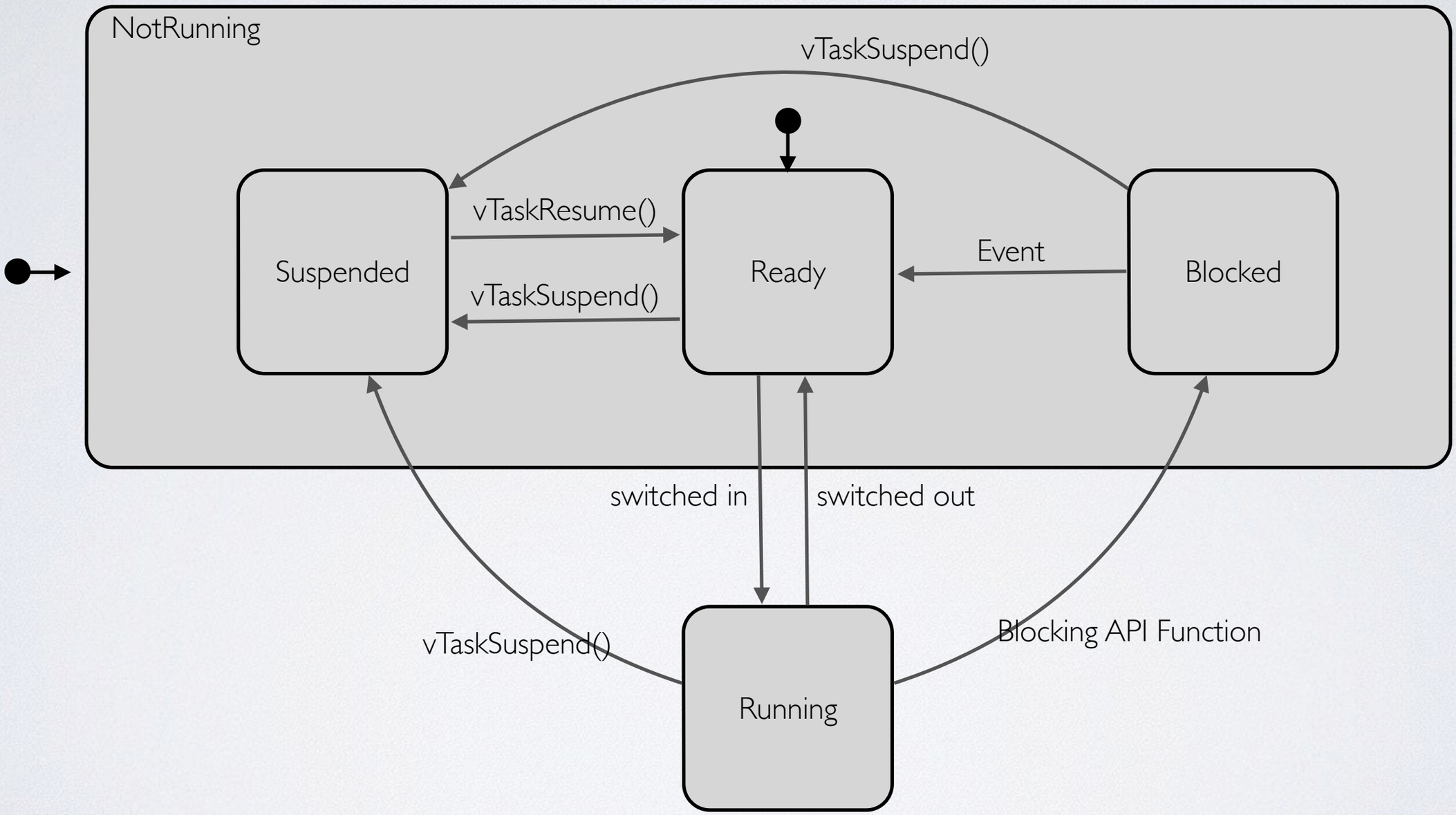
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task-Zustände

- *Not-Running*:
  - *Blocked*: Warten auf Zeit- oder Synchronisation-Ereignis
  - *Suspended*: ausgesetzt vom Scheduling mittels API Funktion
  - *Ready*: Bereit zur Ausführung
- *Running*: In Ausführung

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task-Zustände



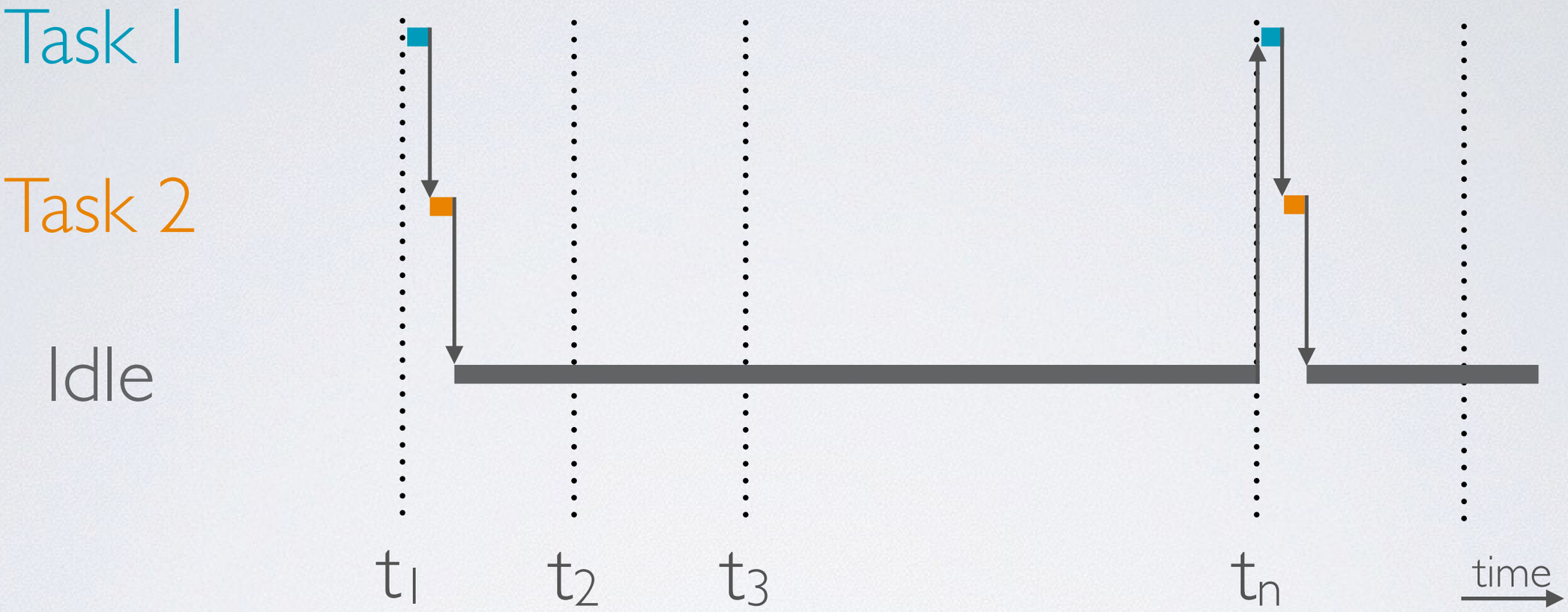
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task

```
// Example 3: Task 1 higher prio. than task 2
// but new task function for both.

void aTaskFunction(void *pvParameters) {
   // Each instance has own copy of this local variable.
   int aVariable = 0;

   // A task is implemented as infinite loop.
   while (1) {
      // Task code here
      // …

      // Delay for 250ms, go into Blocked state.
      vTaskDelay( 250 / portTICK_RATE_MS );
   }
}
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Scheduling (Example 3)



- Task 1 mit höherer Priorität als Task 2
- Tasks gehen in Warteposition für ein Zeitintervall von n Ticks.
- Achtung: Kernel Overhead Prozess ist nicht dargestellt.

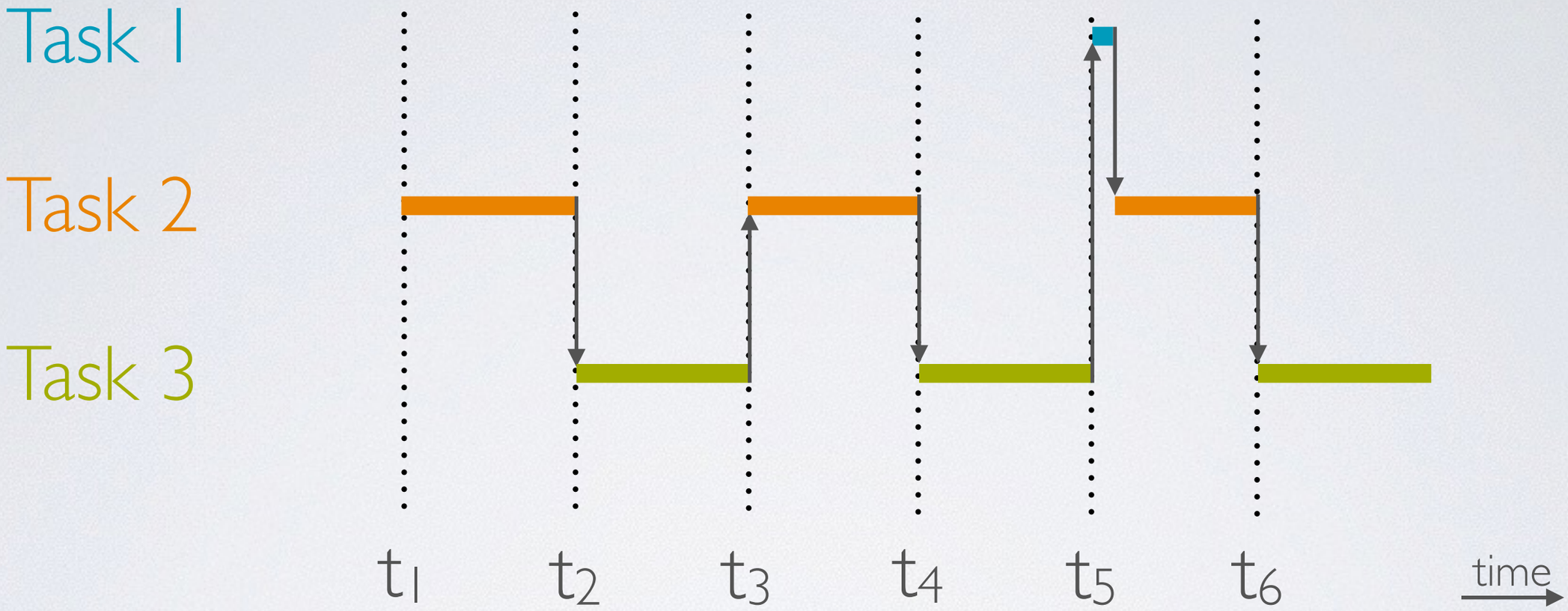
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Task

```
// Example 4: Task 1 higher prio. than task 2 and task 3
// Shown here the new additional task 1 function.

void periodicTaskFunction(void *pvParameters) {
  // Save current tick count (time).
  portTickType lastWakeTime = xTaskGetTickCount();

  // A task is implemented as infinite loop.
  while (1) {
    // Task code here
    // …

    // Call exactly every 5ms.
    vTaskDelayUntil(&lastWakeTime, (5/portTICK_RATE_MS));
  }
}
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Scheduling (Example 4)



- Task 1 mit höherer Priorität als Task 2 und Task 3
- Task 1 wird periodisch aufgerufen; Hier Time Slice = 1 ms.
- Achtung: Kernel Overhead Prozess ist nicht dargestellt.

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Scheduling

- *Fixed Priority Pre-emptive Scheduling*
  - Prioritäten nur vom Task selber änderbar
  - Präemption
    - Höchste Priorität zuerst
    - Bei gleicher Priorität Round Robin
- *Co-operative Scheduling*
  - Wechsel der Tasks nur freiwillig oder bei Blockierung

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Umsetzung RMS

- Rate Monotonic Scheduling mit FreeRTOS
  - Verhindern von eigenen Prioritätsänderung der Tasks
  - Zuordnen der Prioritäten entsprechend RMS, d.h. je kleiner die Periode je höher die Priorität eines Tasks
  - Keine gleichen sondern nur eindeutige Prioritäten
  - Einberechnen des Context Switchs

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Weitere Features

- Einige weitere Features:
  - Queues zur Kommunikation zwischen den Tasks
  - Semaphoren zur Unterstützung von Interruptabarbeitung
  - Mutexes zum Ressourcen Management

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Queues

- Queues
  - zur Kommunikation zwischen Tasks
  - üblicherweise in FIFO modus
  - eigenständig (gehören keinem Task)
  - Zugriff auf Queue eines Tasks optional mit Wartezeit
    - Task in *Blocked*-Zustand bis Lese/Schreib-Zugriff möglich oder bis Wartezeit abgelaufen
    - Höchste Priorität gewinnt, sonst längste Wartezeit bei mehreren Tasks in *Blocked*-Zustand

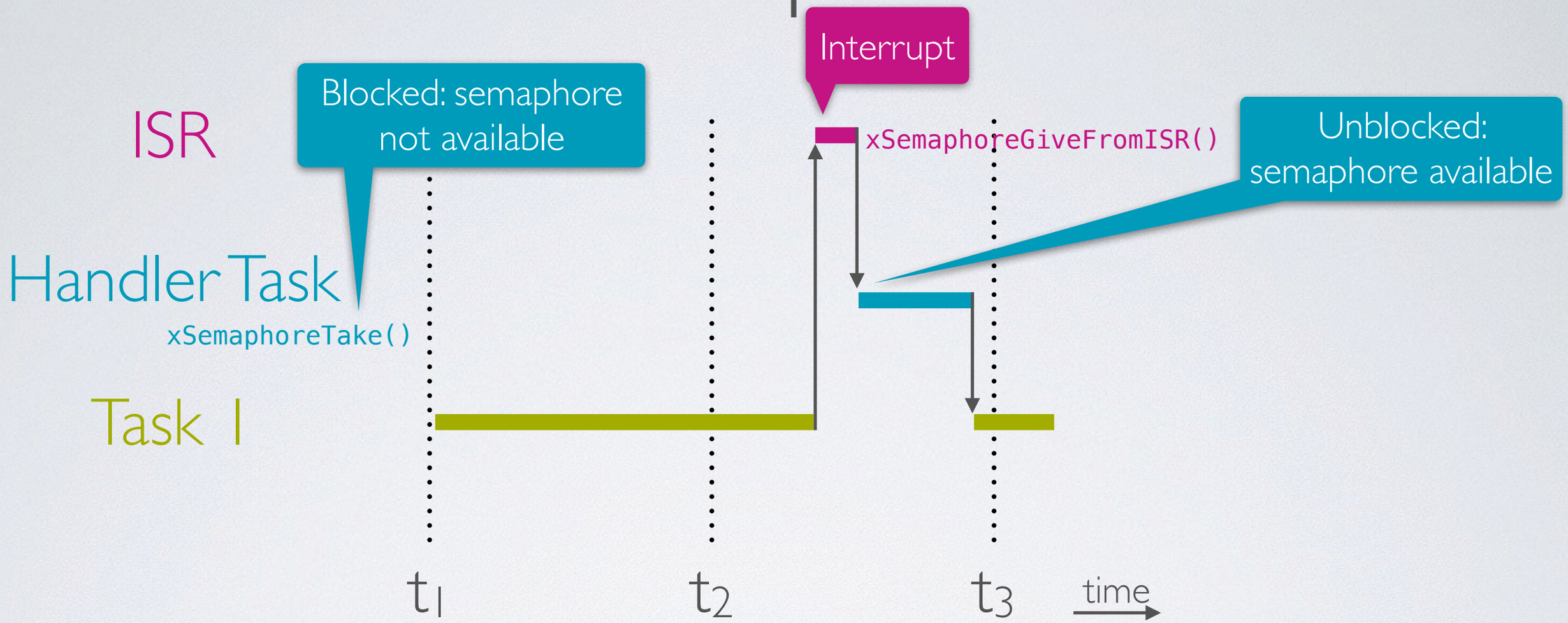
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Queues

```
// A few of the basic queue functions

// Create a queue of length for items of size and return handle
xQueueHandle xQueueCreate(unsigned portBASE_TYPE uxQueueLength,
                          unsigned portBASE_TYPE uxItemSize );

// Send item (size of item known by generated queue) to back of queue
// and wait some ticks for space on the queue if it is already full.
// Return value shows if send was successful.
portBASE_TYPE xQueueSendToBack(
                          xQueueHandle xQueue,
                          const void*  pvItemToQueue,
                          portTickType xTicksToWait );

// Read/receive item from queue into buffer. If queue is empty wait some ticks
// for new item. Return value shows if read was successful.
portBASE_TYPE xQueueReceive(
                             xQueueHandle  xQueue,
                             void*         pvBuffer,
                             portTickType  xTicksToWait );
```

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

FreeRTOS: Semaphoren und ISR

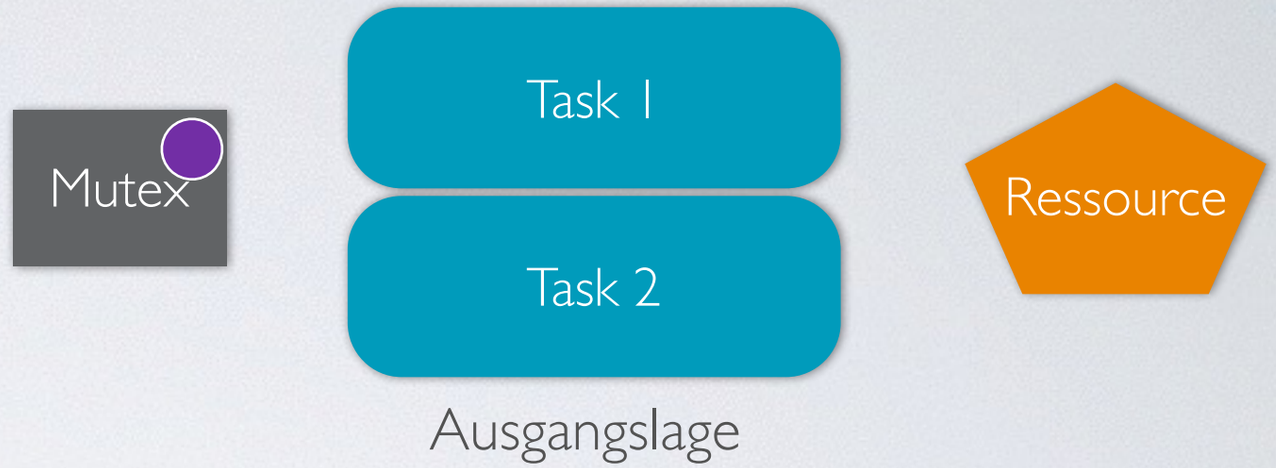


- Binäre Semaphoren zur Synchronisierung mit Interrupts
- Kurze Interrupt Service Routinen (ISR),
- Höhere Priorität für Handler Task als für Task 1
- Weiterleiten des Interrupts an Handler Task mittels Semaphore

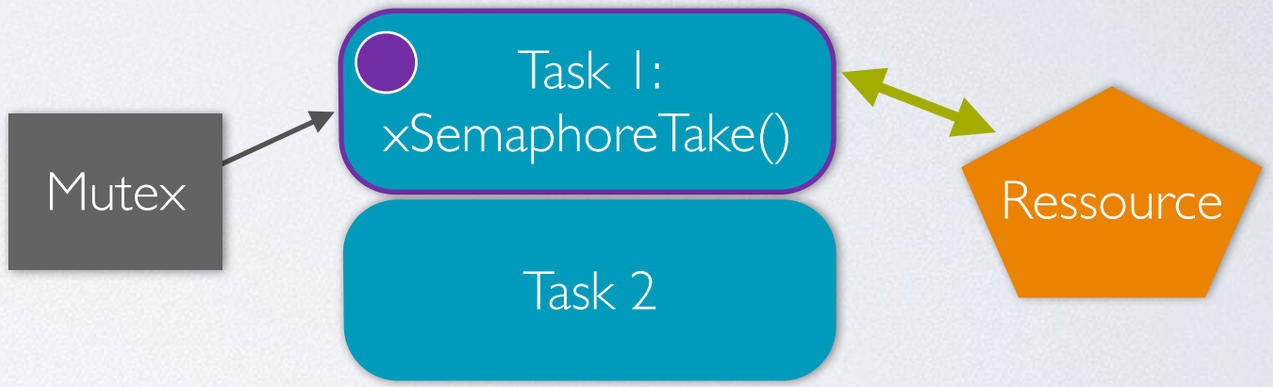
[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

Free RTOS: Mutexes

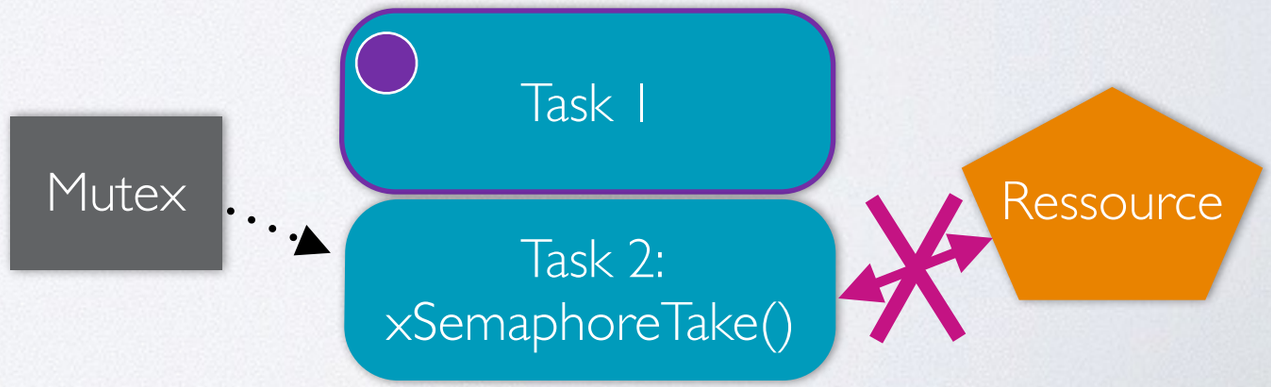
- Mutex (Mutual Exclusion)
- Spezielle binäre Semaphore
- Nehmen und Rückgeben von Token
- Token regelt Zugriff auf gemeinsame Ressource.
- Mit Prioritätsvererbung



Ausgangslage



Task 1 nimmt Mutex.



Task 2 möchte Mutex, muss aber auf Rückgabe warten.

[Quelle: Barry, Using the FreeRTOS Real Time Kernel]

Beispiel einer Anwendung: Mobiles Ticketing System

- Bordcomputer für Telematik und Ticketverkauf im Bus
- Ein Display für Fahrer, ein Display für Kunden, Drucker
- Anbindung an den IBIS-Wagenbus z.B. zur Steuerung der Anzeigen im Bus
- Anbindung an Geldwechsler
- GPS und Funk für Datenübertragung

[Quelle: Wikipedia, Elektronischer Fahrscheindrucker]

Zusammenfassung

- Wiederverwendbare Standardkomponenten
  - Hardware-Abstraction-Layer
  - Middleware
  - (Real Time) Operating Systems
- FreeRTOS als Beispiel eines RTOS

Literatur / Quellen

- AUTOSAR, *Technical Overview,* URL: http://www.autosar.org/index.php?p=1&up=2&uup=0
- R. **Barry**, Using the FreeRTOS Real Time Kernel, Real Time Engineers Ltd. 2011
- G.C. **Buttazzo**, *Hard Real-Time Computing Systems*, Springer-Verlag 2011
- FreeRTOS, URL: http://www.freertos.org
- P. **Marwedel**, Eingebettete Systeme, Springer-Verlag, 2008
- The **Open Group**, *POSIX 1003.1 Frequently Asked Questions (FAQ Version 1.14)* URL: http://www.opengroup.org/austin/papers/posix_faq.html
- A. **Ramos**, Writing Your First MQX Application, AN3905, Freescale, 2009
- UBM Tech, *2013 Embedded Market Study*, 2013
- Wikipedia, Elektronischer Fahrscheindrucker, URL: http://de.wikipedia.org/wiki/Elektronischer_Fahrscheindrucker
- Wikipedia, Herzschrittmacher, URL: http://de.wikipedia.org/wiki/Herzschrittmacher
- **Stand aller Internetquellen: 25.02.2014**