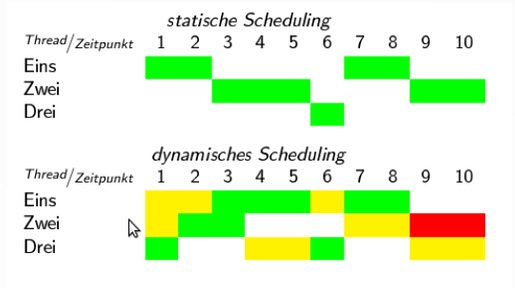
Der Embedded Spielplatz

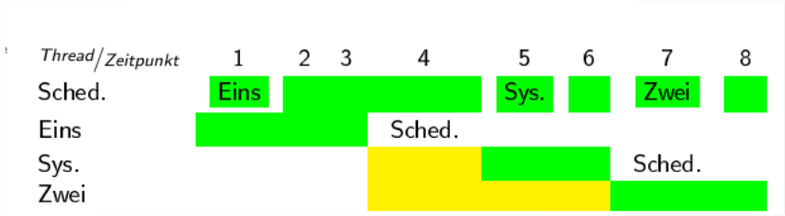
wenn Quellcode Samba tanzt

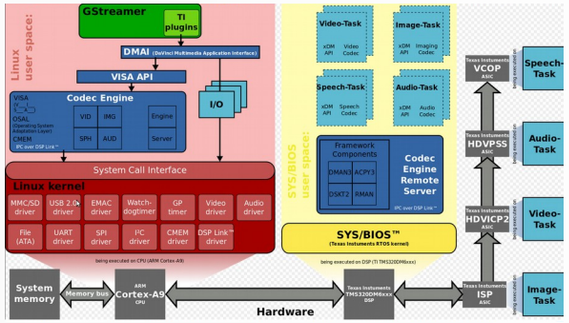
Timo Burger
JMU-Würzburg

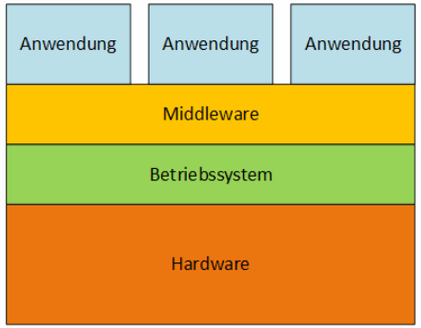
24 Juni 2020

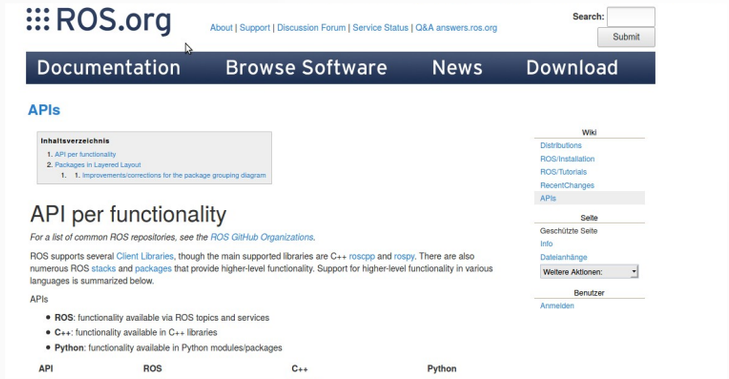
offenes Informatikcolloqium

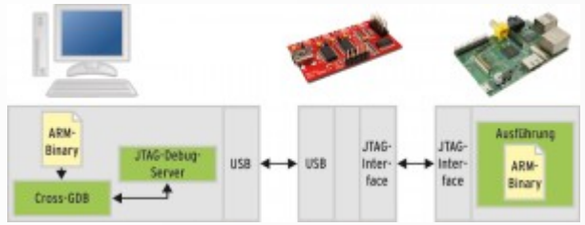

statische Scheduling
Thread/Zeitpunkt 1 2 3 4 5 6 7 8 9 10
Eins
Zwei
Drei
dynamisches Scheduling
Thread/Zeitpunkt 1 2 3 4 5 6 7 8 9 10
Eins
Zwei
Drei


Thread/Zeitpunkt 1 2 3 4 5 6 7 8
Sched.
Eins
Sys.
Zwei
Eins
Sched.
Sys.
Sched.
Zwei


GStreamer
TI plugins
DMAI
VISA API
Codec Engine
I/O
System Call Interface
Linux kernel
Linux user space:
SYS/BIOS user space:
Video-Task
Image-Task
Speech-Task
Audio-Task
Framework Components
Codec Engine Remote Server
SYS/BIOS™
System memory
ARM Cortex-A9
Hardware
Speech-Task
Audio-Task
Video-Task
Image-Task


Anwendung
Anwendung
Anwendung
Middleware
Betriebssystem
Hardware


ROS.org
About | Support | Discussion Forum | Service Status | Q&A answers.ros.org
Search:
Submit
Documentation
Browse Software
News
Download
APIs
Inhaltsverzeichnis
1. API per functionality
2. Packages In Layered Layout
1. 1. Improvements/corrections for the package grouping diagram
API per functionality
For a list of common ROS repositories, see the ROS GitHub Organizations.
ROS supports several Client Libraries, though the main supported libraries are C++ roscpp and rospy. There are also numerous ROS stacks and packages that provide higher-level functionality. Support for higher-level functionality in various languages is summarized below.
APIs
ROS: functionality available via ROS topics and services
C++: functionality available in C++ libraries
Python: functionality available in Python modules/packages
API
ROS
C++
Python
Wiki
Distributions
ROS/Installation
ROS/Tutorials
RecentChanges
APIs
Seite
Geschützte Seite
Info
Dateianhänge
Weitere Aktionen:
Benutzer
Anmelden


ARM-Binary
JTAG-Debug-Server
Cross-GDB
USB
USB
JTAG-Inter-face
JTAG-Inter-face
Ausführung
ARM-Binary

Was sind embedded Systems?
wieso – weshalb - warum

1. CPU
2. Stromversorgung
3. Peripherie

<u>Im Regelfall:</u>

- Minimale Kosten
- Wenig Platz
- Geringer Energie- und Speicherverbrauch

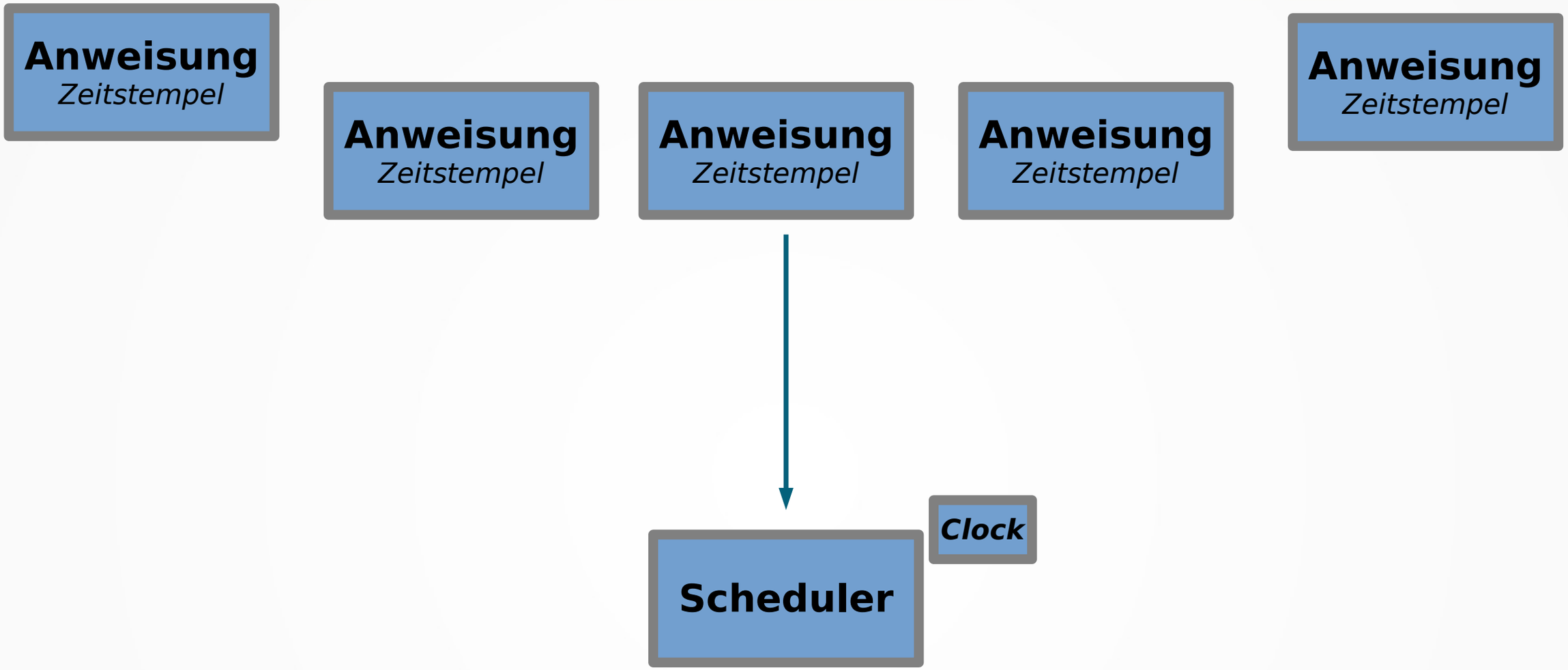
Das Betriebssystem

ROS	OS
• führt Anweisung dann aus, wenn gewollt • RODOS, TIRTOS(Sysbios), FreeRTOS	• so viele Anweisungen wie möglich • WIN, MacOS, Android

Auf Timing kommt es an!

Anweisung
Zeitstempel

Wie sieht dieser Stempel aus?



statische - dynamisch

Statisches Scheduling

- Feste Zeiten
- Eindeutige Reihenfolge

Zeitpunkt / Thread	1	2	3	4	5	6	7	8	9
Eins									
Zwei									
Drei									

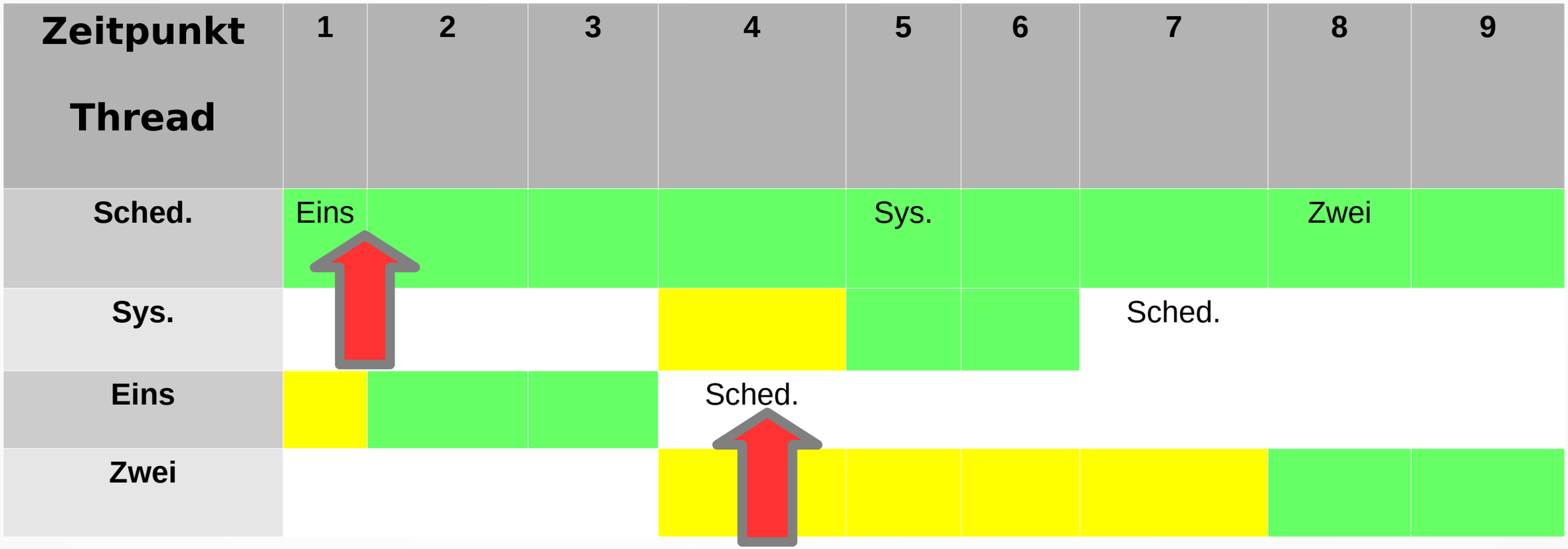
Dynamische Scheduling

- feste Prioritäten
- Rate-Monotonic(Periodendauer)
- Earliest Deadline First

→ Multitasking möglich

Kooperatives Multitasking

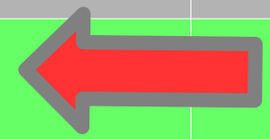
Keine gute Strategie



Präemptives Multitasking

Basis aller Scheduling-Methoden

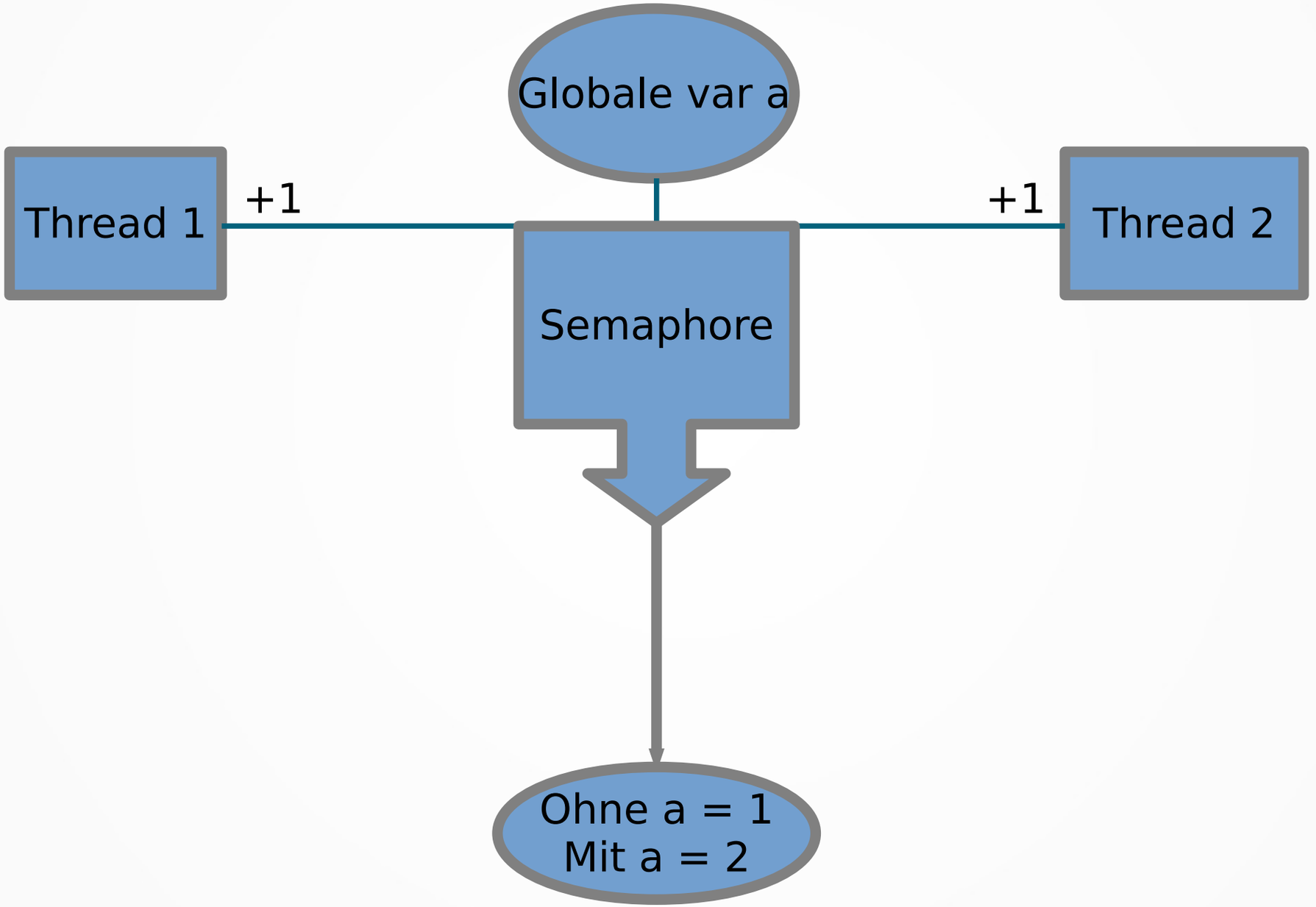
Zeitpunkt / Thread	1	2	3	4	5	6	7	8	9
Sched.	Eins			Eins	Sys.		Sys.		
Sys.									
Eins									
Zwei									



Synchronisation von Threads

It just stays there

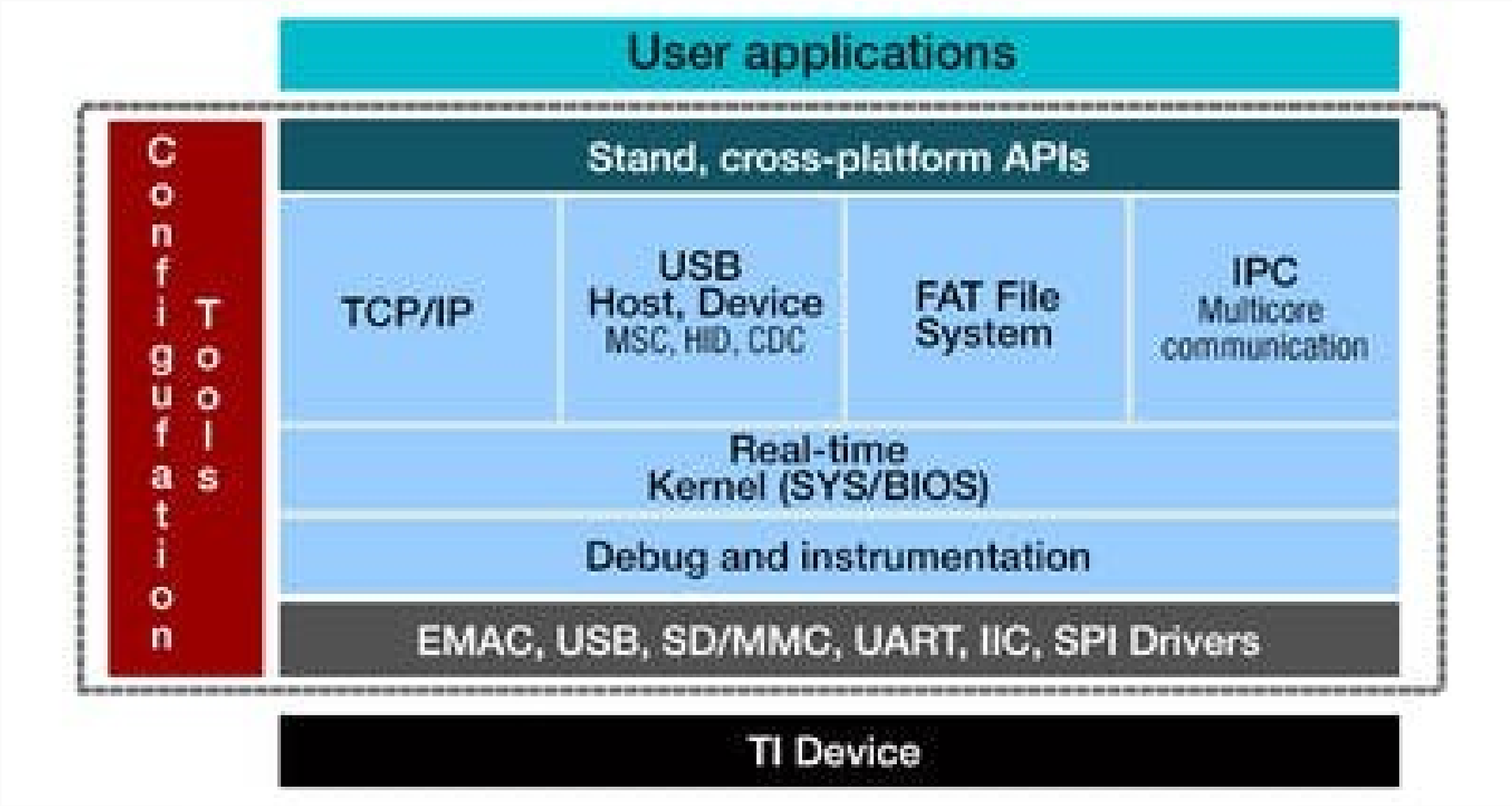
C/C++ - der größte Feind



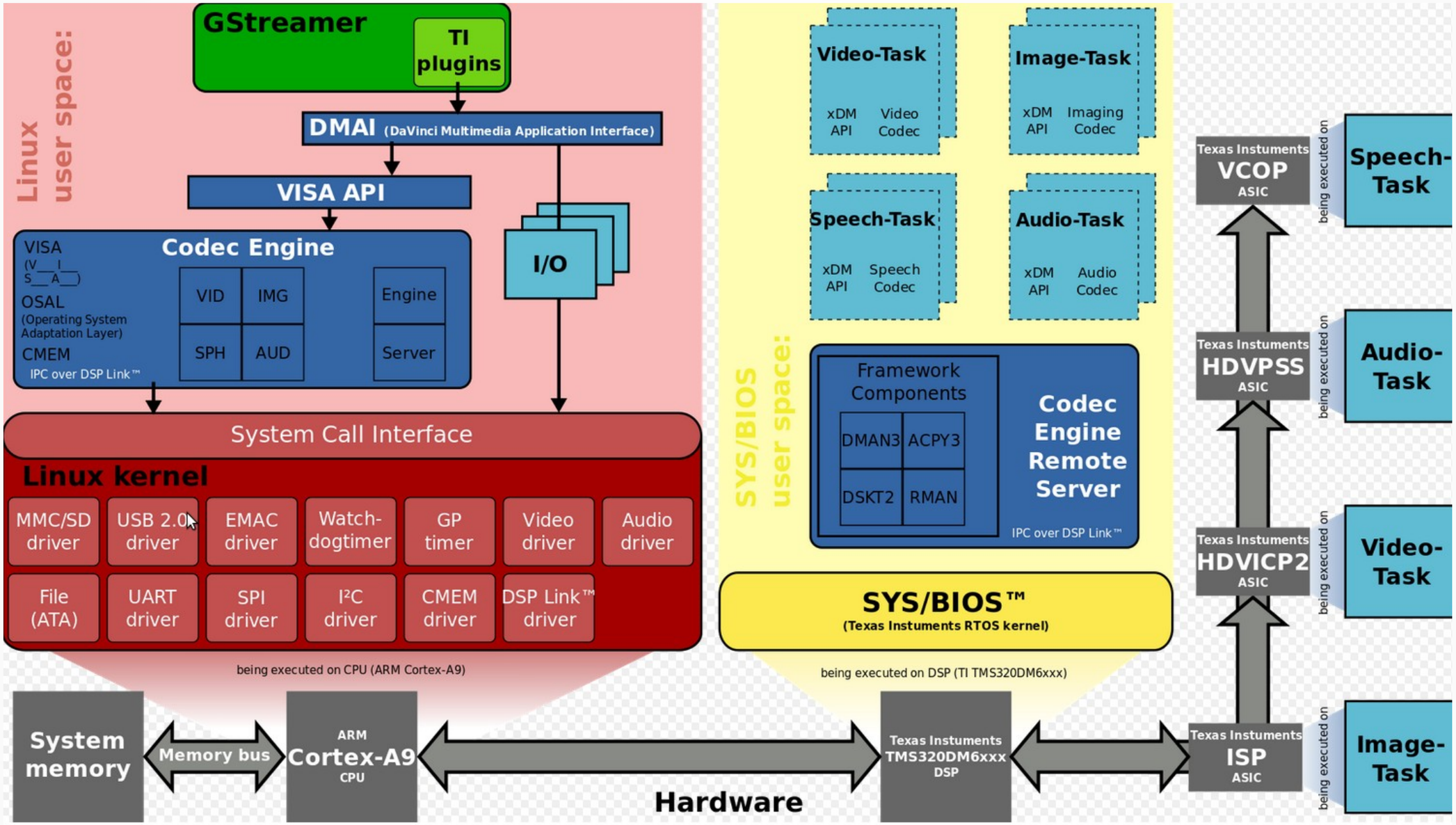
OS-Features

- Betriebsystem - standalone
- Middleware - Kommunikation(intern/extern)
- Stromversorgung
- Watchdog

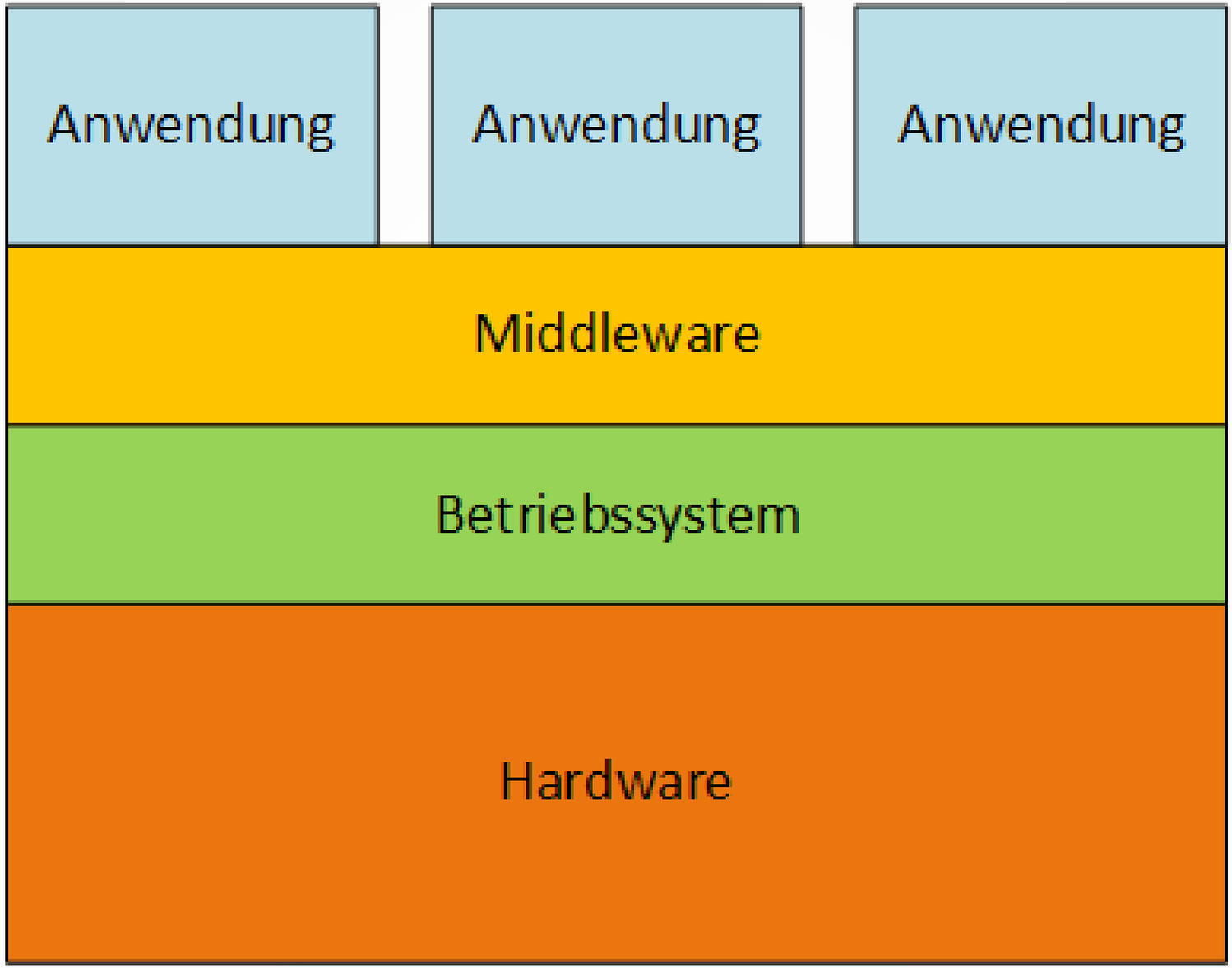
Betriebsystem - standalone



Betriebssystem - standalone



Middleware – Kommunikation(intern/extern)



LIVE DEMO

Middleware

Energieverbrauch/CPU-Taktrate

- Booten → Systemzustand: IDLE(vor Scheduling)
- Energiesparmaßnahmen
- Herunterfahren der Taktrate mittels Treiber,
  ansprechbar, z.B. I²C

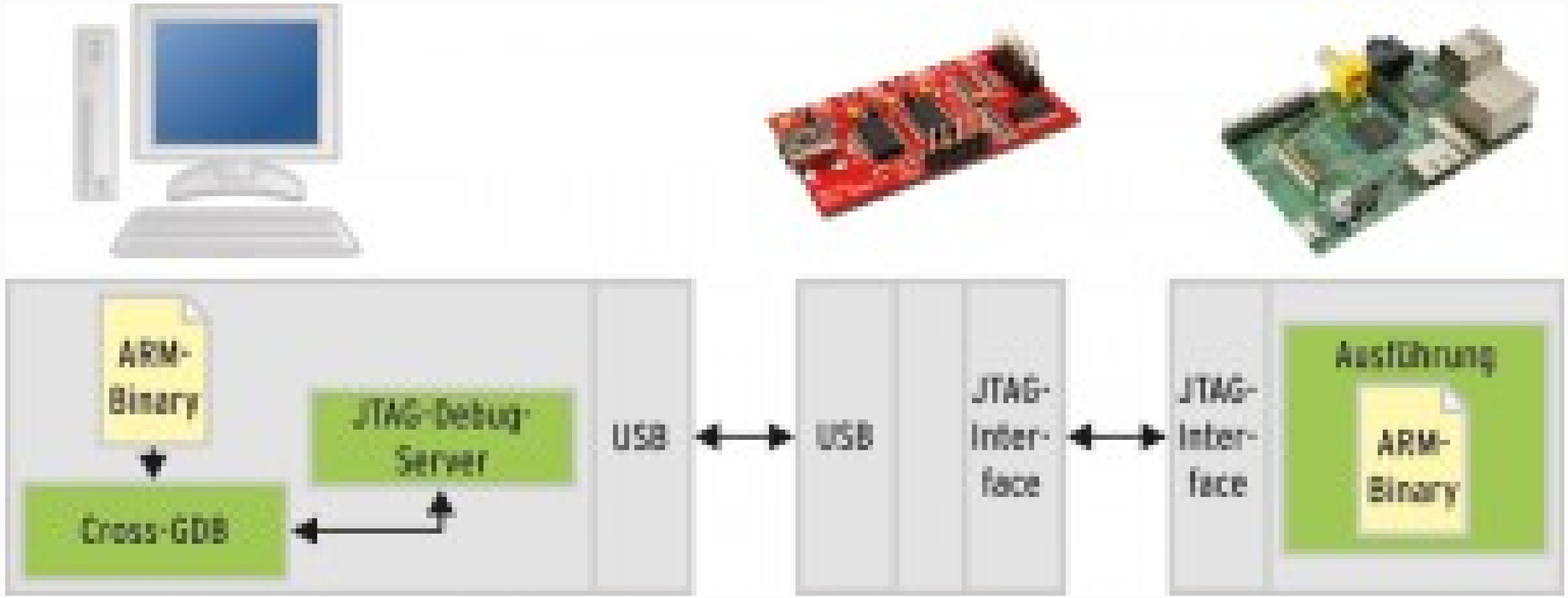
Watchdog

- Überwacht das System von Threads und anderen Systemkomponenten
- Reset oder beenden der ausgefallenen Software
- Hardware/Software Watchdog

Programmierung

- Debugging
- Analyse-Features

Betriebsystem - standalone



LIVE DEMO

Programmierung
+
Debugging

Bootloader und Sequenz

- Mehrstufiges booten
- Firststage Bootloader
- Second Stage Bootloader

  Opensource:
  https://github.com/u-boot/u-boot

```
U-Boot 2010.12-xes_r3 (Aug 25 2011 - 11:04:04)

CPU0:  P2020E, Version: 1.0, (0x80ea0010)
Core:  E500, Version: 4.0, (0x80211040)
Clock Configuration:
       CPU0:1066.667 MHz, CPU1:1066.667 MHz,
       CCB:533.333 MHz,
       DDR:400  MHz (800 MT/s data rate) (Asynchronous), LBC:133.333 MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: X-ES XPedite5501 PMC/XMC SBC
       Rev SA, Serial# 36093001, Cfg 90015130-1
I2C:   ready
DRAM:  2 GiB (DDR3, 64-bit, CL=6, ECC on)
FLASH: Executed from FLASH1
POST memory PASSED
FLASH: 256 MiB
L2:    512 KB enabled
NAND:  4096 MiB
PCIE1: connected as Root Complex (no link)
PCIE1: Bus 00 - 00
PCIE2: disabled
PCIE3: disabled
In:    serial
Out:   serial
Err:   serial
DTT:   37 C local / 59 C remote (adt7461@4c)
DTT:   37 C local (lm75@48)
Net:   eTSEC1 connected to Broadcom BCM5482S
eTSEC2 connected to Broadcom BCM5482S
eTSEC1, eTSEC2
POST i2c PASSED
Hit any key to stop autoboot:  0
=>
```

Bootsequenz

Festegelegte Bootreihenfolge(second stage Bootloader):

- EMMC/SD-Karte (wenn vorhanden)
- UART
- USB
- Bluetooth
- Netzwerkverbindungen

Muss vorher extern über entsprechende Verbindung geladen werden!

Disclaimer – eher grundlegende Devisen

- Hardware ist begrenzt, ihr Speicher auch
- Verstehen und Wissen, sind zwei unterschiedliche paar Schuhe → behalte den Überblick
- Dokus sind Freunde, kein Futter

QUELLCODE UND HARDWARE SIND GEIL!!!

Abspann

- TIRTOS(Sysbios) – http://processors.wiki.ti.com/index.php/TI-RTOS
- RODOS http://www8.informatik.uni-wuerzburg.de/wissenschaftforschung/rodos/
- ROS http://wiki.ros.org/de
- Debugging – https://www.linux-magazin.de/ausgaben/2013/09/embedded-debugging/
- Bootloader – https://www.embedded.com/Bootloader/
- Bootorder – https://www.embedded.com/fundamentals-of-booting-for-embedded-processors/
- Middleware – ROS + Graphik von wikipedia.de Eintrag „Middleware“