h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Warp 3 zwischen allen Kernen

Entwicklung einer schnellen und sicheren Multicore-RTE

Thomas Barth(h-da)
Prof. Dr.-Ing. Peter Fromm (h-da)

Übersicht

- Herausforderung Multicore
- Anforderungen Laufzeitumgebung
- Signalobjekte und Laufzeitumgebung
- Workflow
- Anbinden von Kommunikations-Stacks
- Ausblick

Herausforderung Multicore

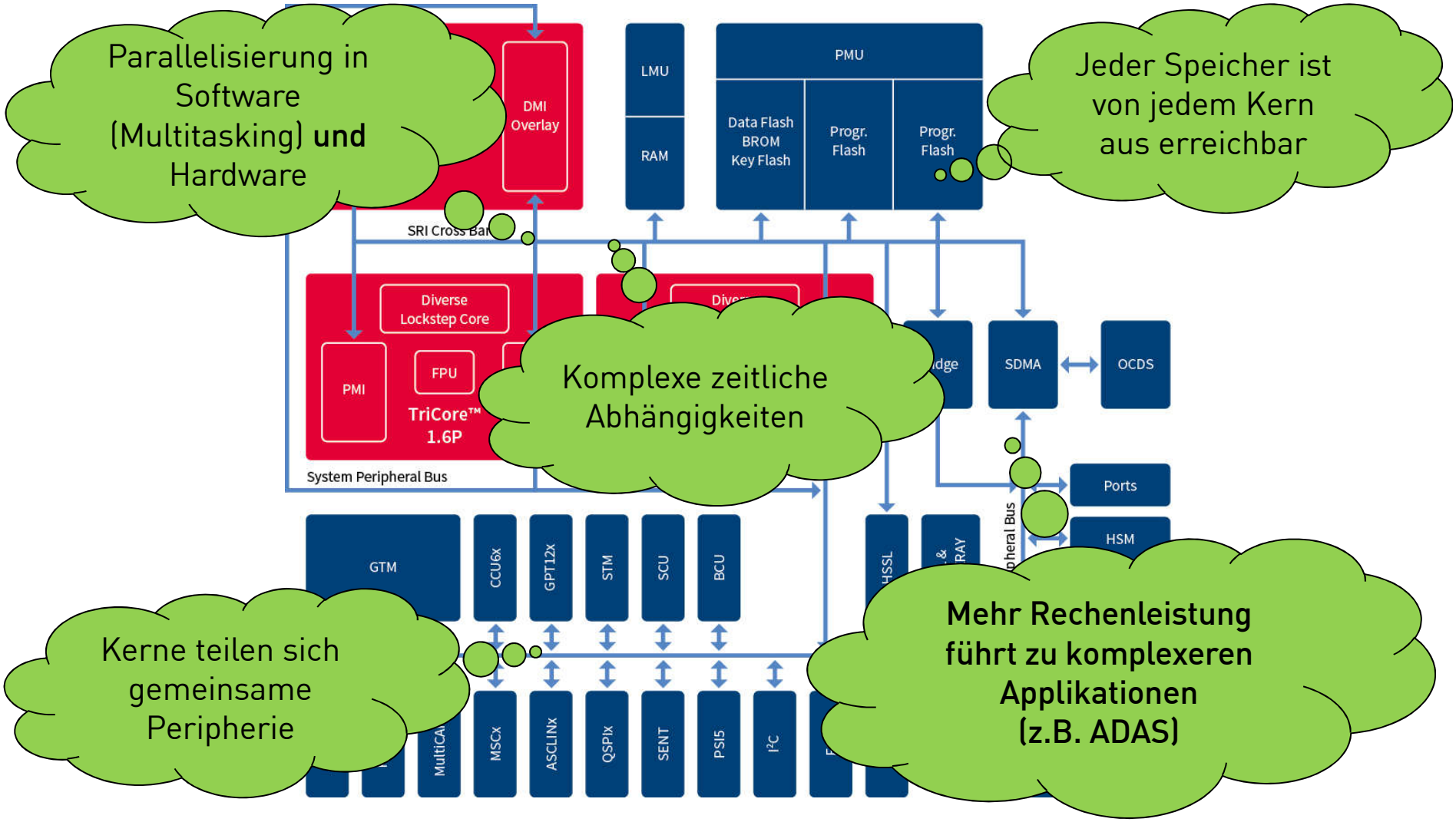
Multicore Mikrocontroller werden Singlecore Controller in vielen Bereichen aufgrund ihrer Leistungsfähigkeit verdrängen.

Herausforderung Multicore:

- Entwicklungskosten werden um den Faktor 4.5 steigen
- Personalbedarf wird um das dreifache steigen.

Quelle: Next Generation Embedded Hardware Architectures VDC Research, 2010

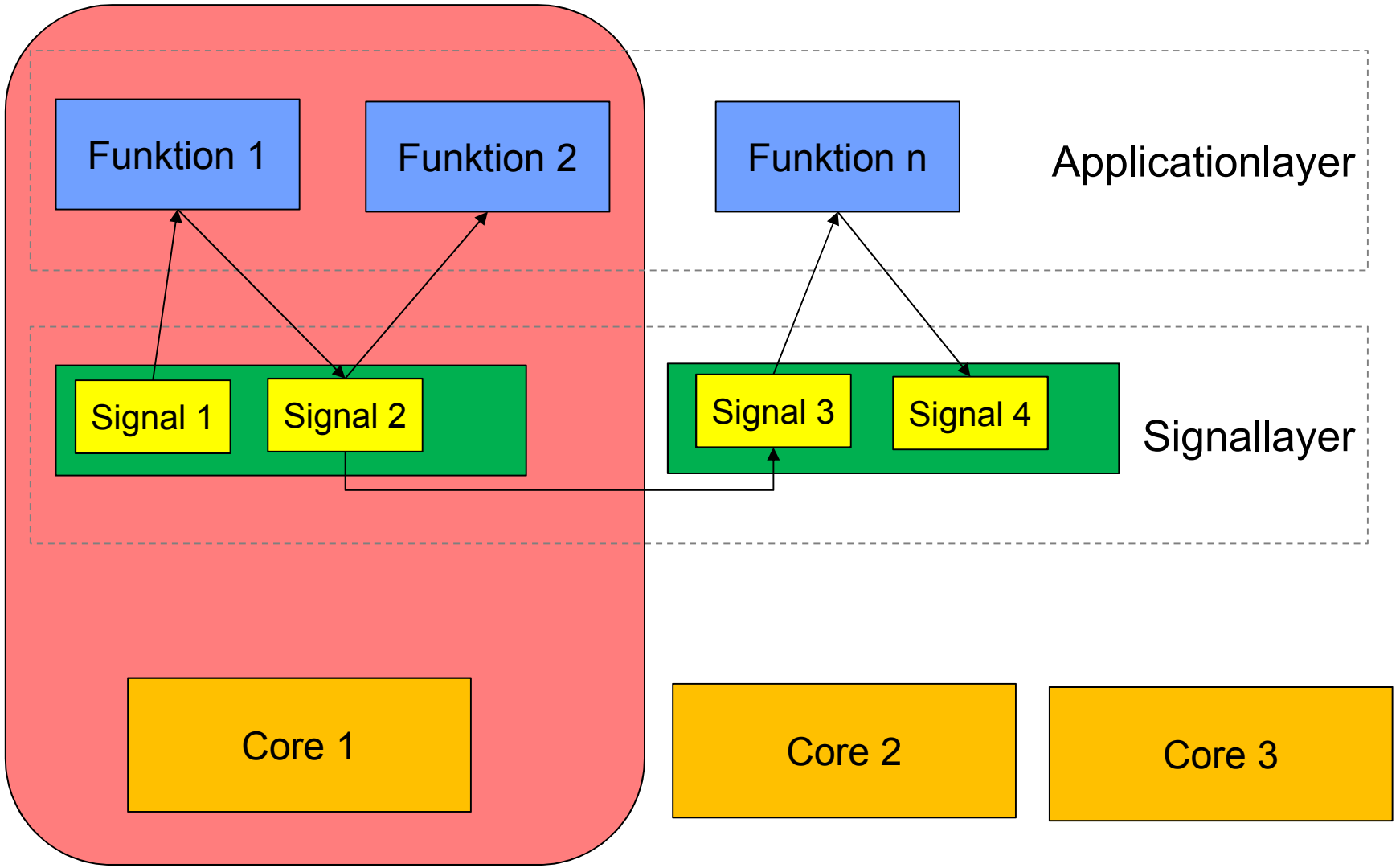
Herausforderung Multicore



Anforderungen

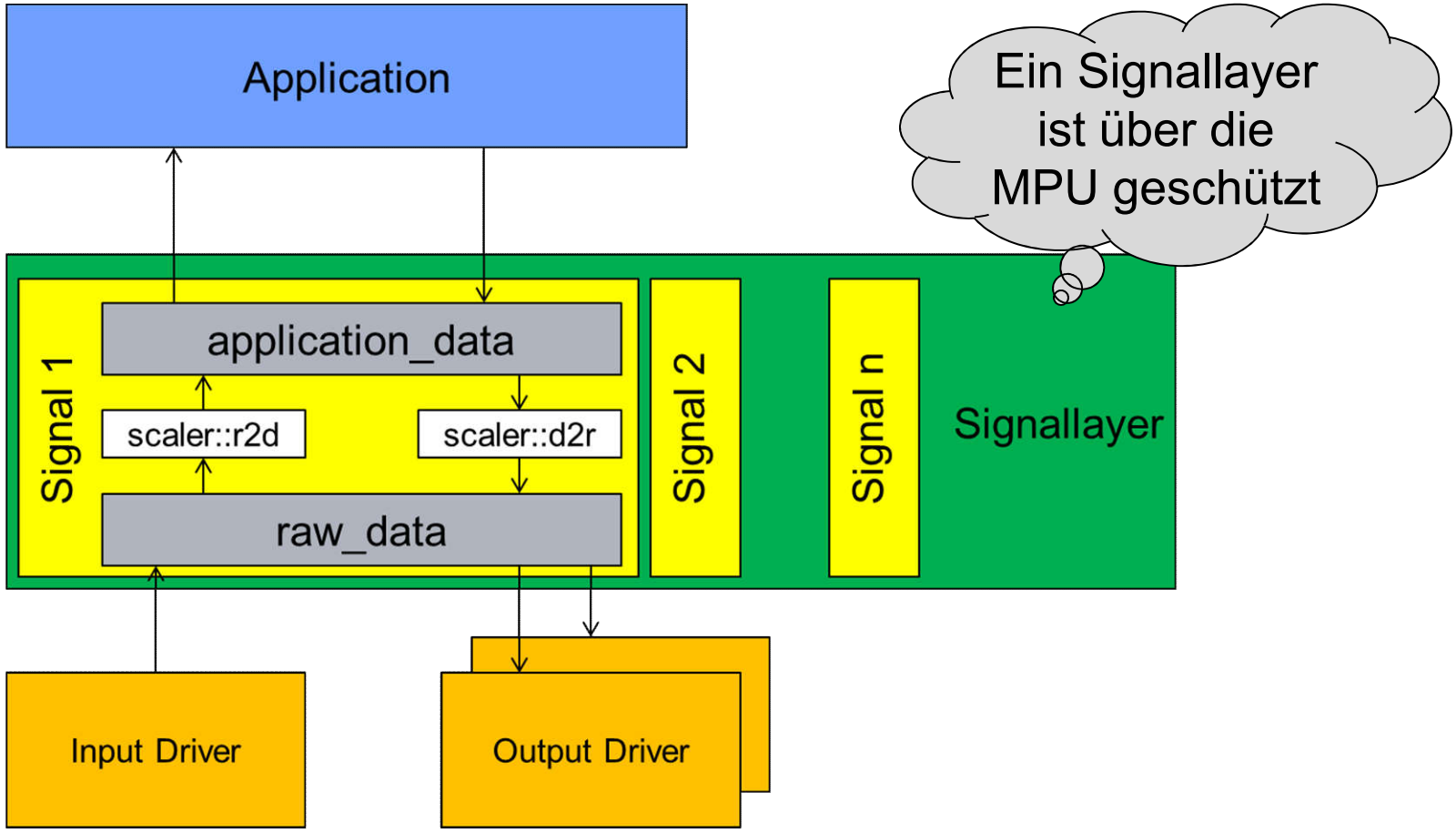
- Über eine AUTOSAR ähnliche Laufzeitumgebung soll eine **Trennung** der Applikation von der Basissoftware erfolgen. Hiermit soll ein verbesserter **Reuse** von Code zwischen verschiedenen Hardwareplattformen erreicht werden.
- Die Laufzeitumgebung soll **einfach** und ohne großen Tooloverhead realisiert werden.
- Zykluszeiten im Bereiche **<1ms** sollen unterstützt werden.
- Der Applikation sollen „physikalische“ Werte bereit gestellt werden, die Basissoftware soll über eine **normierte Treiberschnittstelle** angebunden werden
- Eine **Kapselung** unterschiedlicher Domänen zur Realisierung von Safety Architekturen soll ermöglicht werden.
- **Multicore** Controller sollen unterstützt werden.

Grundidee: Reduzierung der Komplexität für den Applikationsentwickler



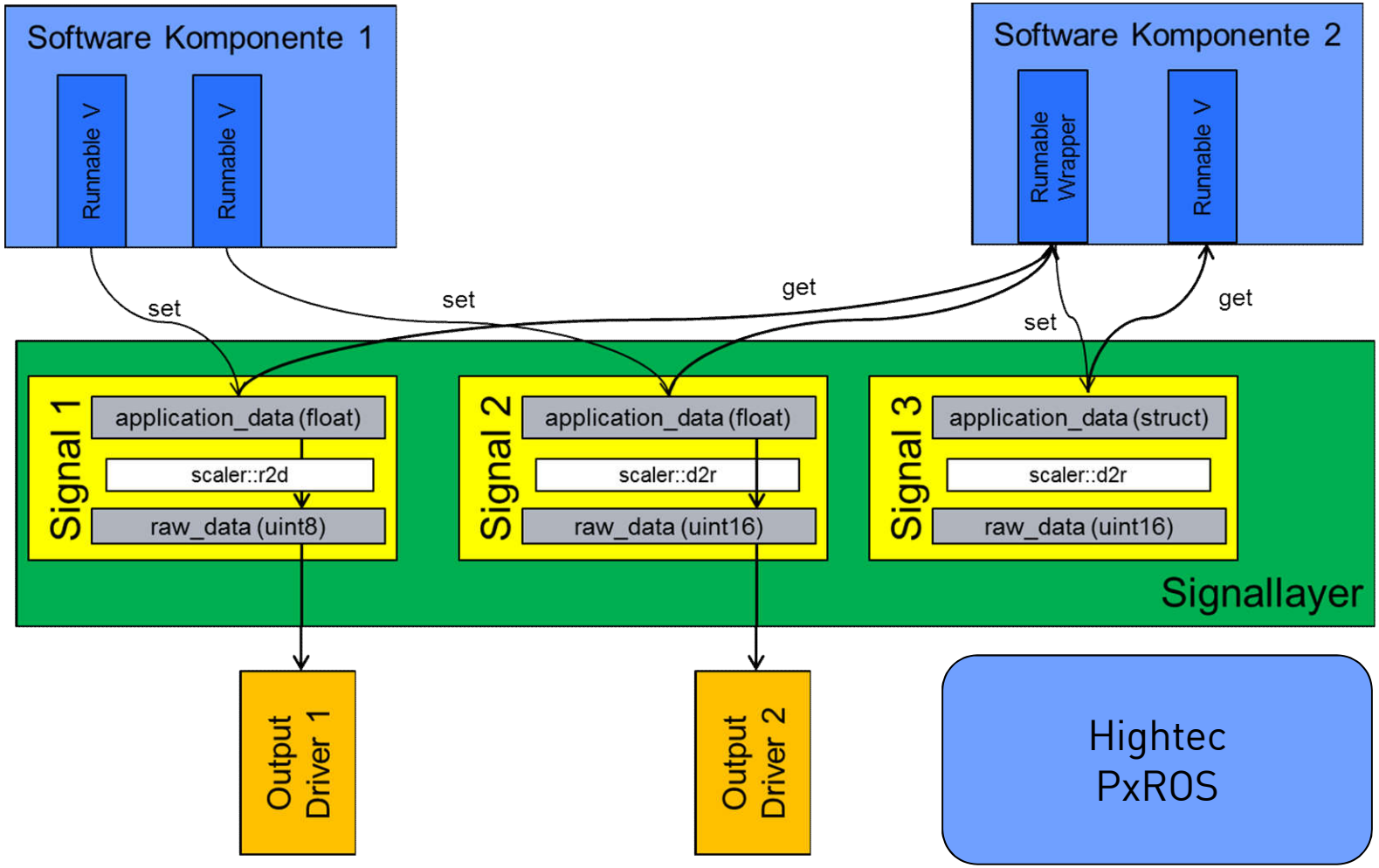
Signal

Kernelement der Signalschicht sind Datenklassen, welche den Zugriff auf die Signalobjekte über eine einheitliche API definieren.



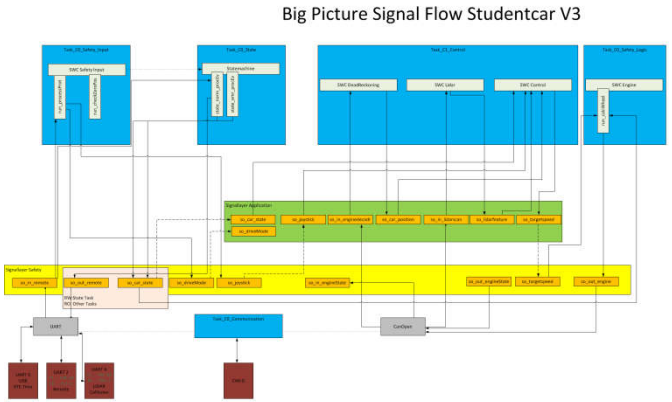
Laufzeitumgebung

Die Laufzeitumgebung definiert das zeitliche Verhalten



Workflow

Definition Datenfluss



Beschreibung der Signale als XML

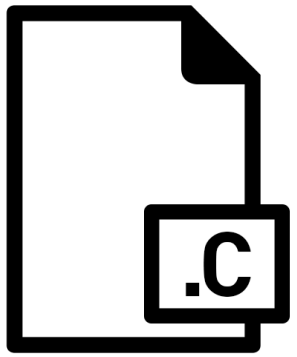
```
<signalclass name=„ADC„ rawdata=„float_t"
          applicationdata=„uint16_t„ (…)/>

<signallayer name = "sl_sensor" description = „Sensor
Data. " memRAM = ".sl_sensor" (…)>

<object name="so_in_v_bat" description = „Battery
voltage" signal="ADC„ (…) />
```

Einhängen von runnables

```
//cyclic runnables
//runnable ptr, cyclic Time, Offset
{(RTp_t)(read_battery), 100, 0},

//event runnables
{(RTp_t)(signal_lowVoltage),
ev_lowVoltage},
```

```
ADC_set(&so_in_v_bat)
ADC_get(&so_in_v_bat)
ADC_get_status(&so_in_v_bat)
{…}
```

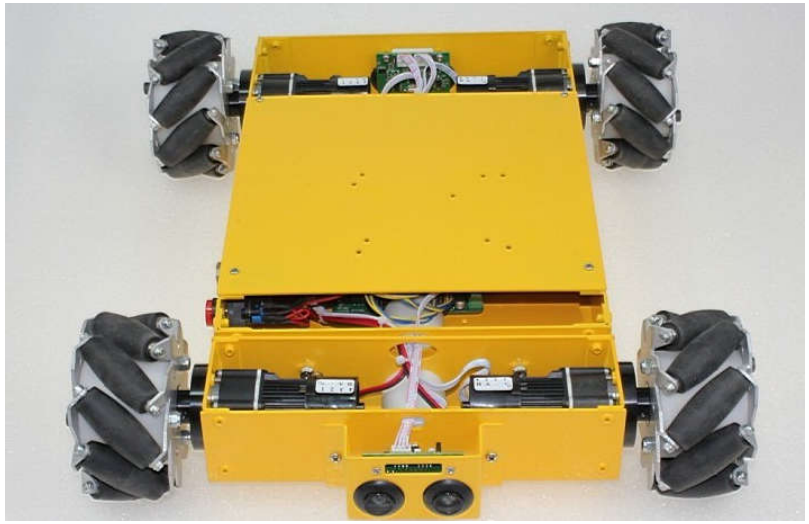
Generieren von C-Code

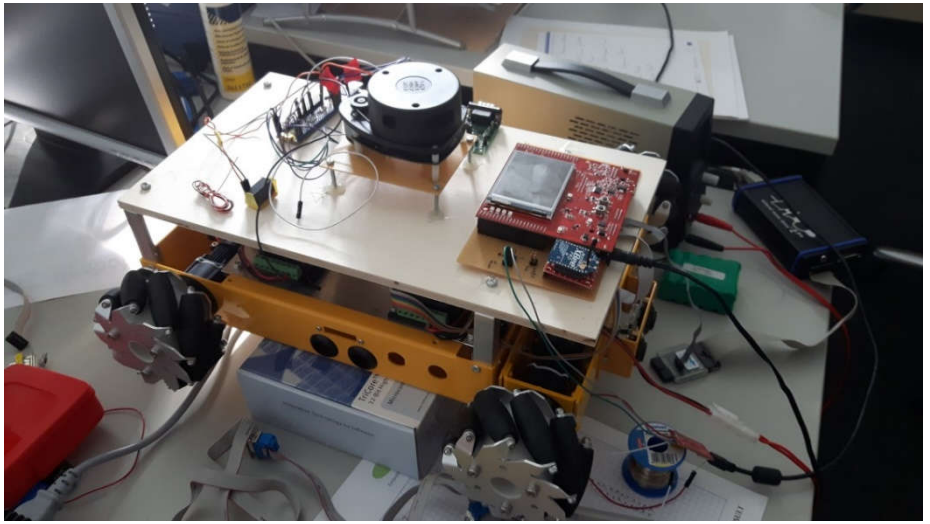


Aufbau eines Demonstrators

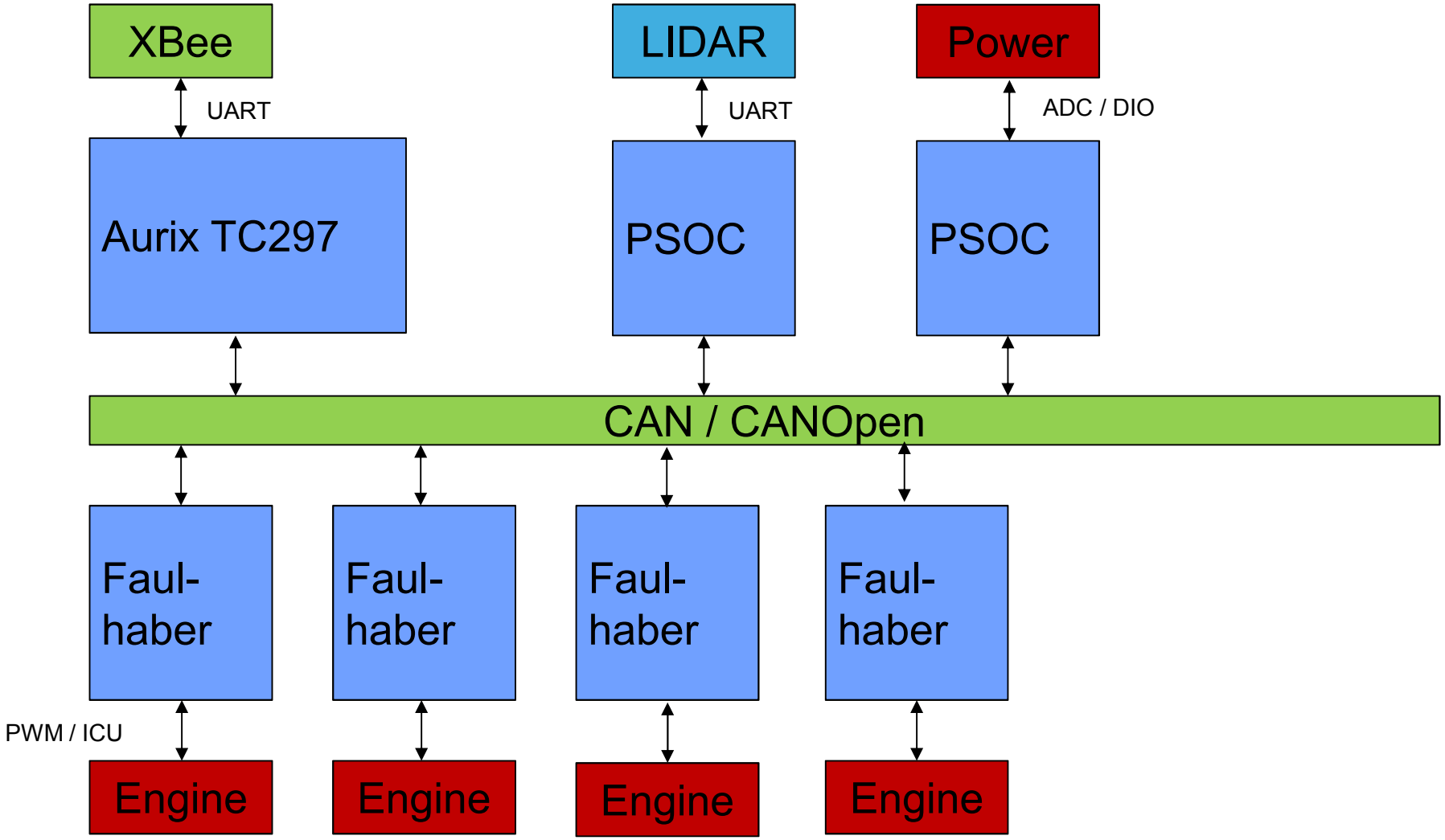
5 Projekte mit 30 Studierenden in 5 Teams 3 Monate in ihrem ersten Multicore Projekt....

- Remote Control via PC
- Integration Laserscanner (Pfadberechnung, Kollisionsvermeidung)
- Koppelnavigation über Drehgeber
- Echtzeit Timinganalyse der Laufzeitumgebung
- Treiberentwicklung

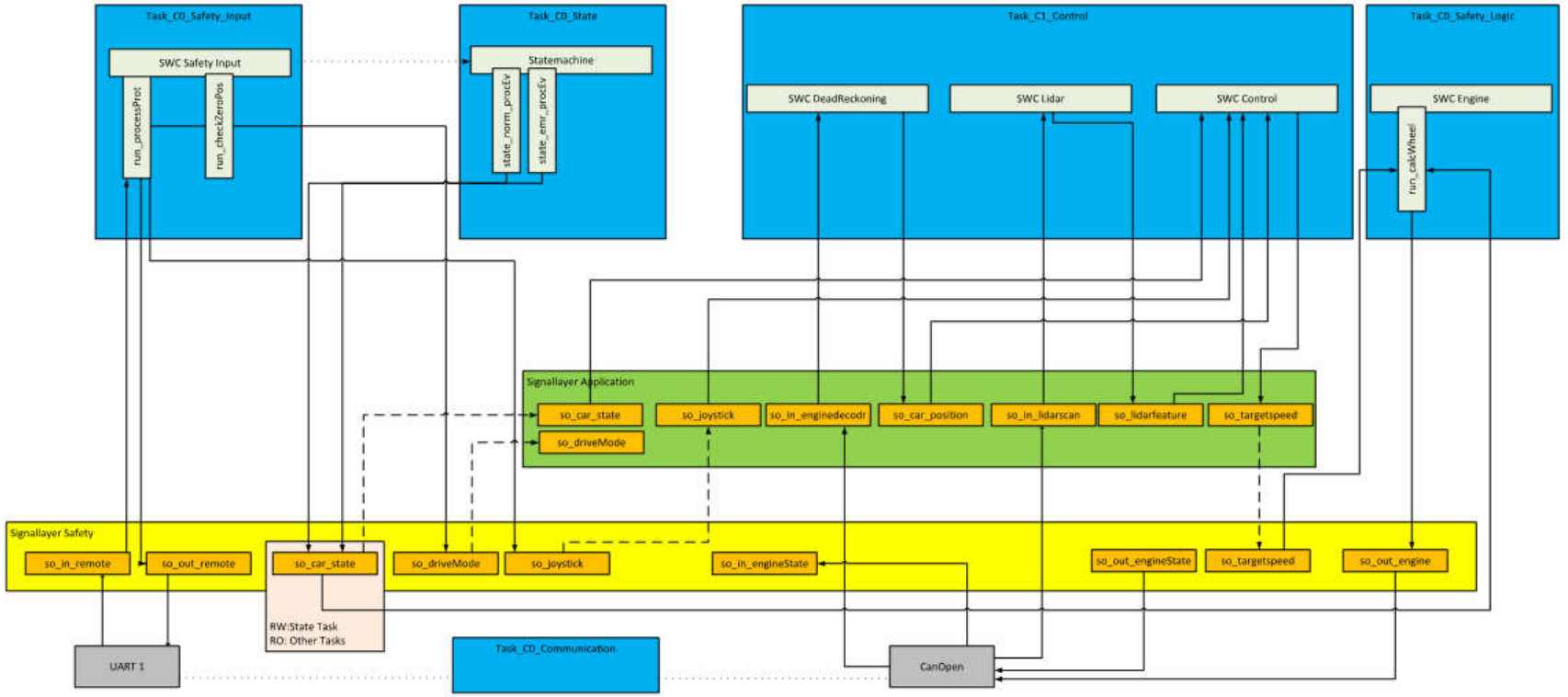




Topologie des Fahrzeugs



Signalfluss



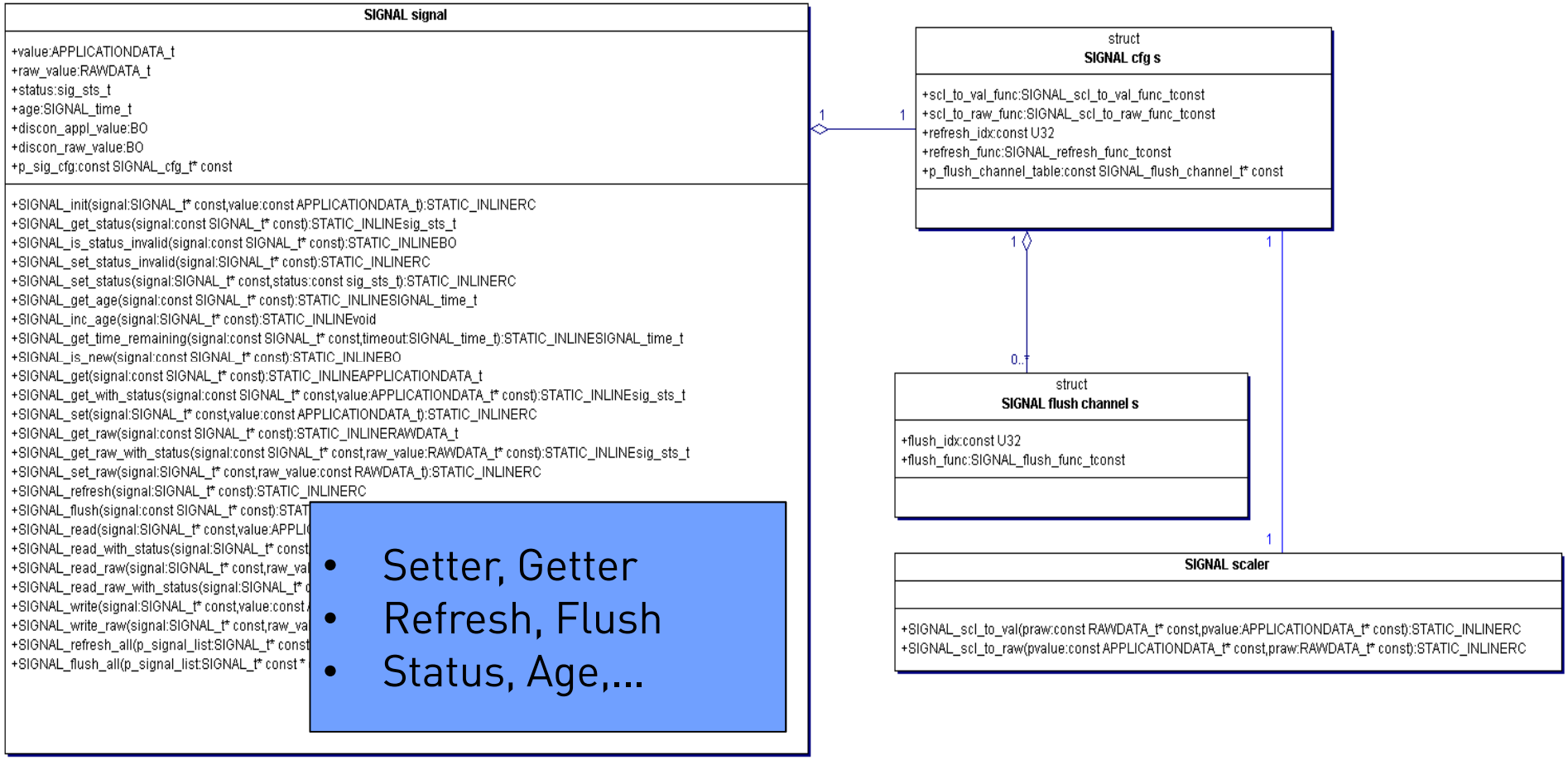
- 10 Runnables
- 20 Signale
- 3 Signaldomänen

Überführung des Designs in XML

```
</signalclasses>

<!-- Memory values can be set explicitly in objects or inherited from sign
<objectpool>

    <signallayer name = "sl_safety" description = "Contains all safety cri
    memRAM = ".sl_safety" memROM = ".rodata">

        <object name="so_in_remote" description = "Protocol package receiv
        "REMOTE" initvalue="REMOTE_INIT_DATA" initrawvalue="REMOTE_INIT_RA
            r2d="REMOTE_scaleR2D" d2r="REMOTE_scaleD2R" inportindex="0" in
            memRAM = "0" memROM = "0">
        </object>




        <object name="so_out_engine" description = "Target value for the e
        initvalue="ENGINE_INIT_DATA" initrawvalue="ENGINE_INIT_RAW"
            r2d="ENGINE_scaleR2D" d2r="ENGINE_scaleD2R" inportindex="0" in
            "opl_engine" memRAM = "0" memROM = "0">
            <outportlist name="opl engine">
```

- carposition_signal.h
- carposition_type.h
- carstate_signal.h
- carstate_type.h
- engine_signal.h
- engine_type.cpp
- engine_type.h
- enginedecoder_signal.h
- enginedecoder_type.cpp
- enginedecoder_type.h
- enginestate_signal.h
- enginestate_type.cpp
- enginestate_type.h
- joystick_signal.h
- joystick_type.h
- lidarfeature_signal.h
- lidarfeature_type.h
- lidarscan_signal.h
- lidarscan_type.cpp
- lidarscan_type.h
- remote_signal.h
- remote_type.cpp
- remote_type.h
- signal_bb.cpp
- signal_bb.h
- sint32_signal.h
- targetspeed_signal.h
- targetspeed_type.cpp
- targetspeed_type.h
- uint8_signal.h

Signalstruktur



Beispiel einer Applikations-Runnable

```
RT_safety_t SAFETYLOGIC_run_checkEngineTargetSpeed()
{

    TARGETSPEED_data_t targetSpeed = TARGETSPEED_get(&so_targetspeed);
    CARSTATE_data_t carState = CARSTATE_get(&so_carstate);

    //Calculate slow down factor for slow speed
    //Take the highest value, limit that to the speed limit set by the
    sint8_t maxSpeedParamater = _abs(targetSpeed.vx);
    if (_abs(targetSpeed.vy) > maxSpeedParamater) maxSpeedParamater =
    if (_abs(targetSpeed.vphi) > maxSpeedParamater) maxSpeedParamater

    //Calculate the slow down factor, based on the max speed allowed
    sint8_t slowSpeedFactor = 1;
    if (maxSpeedParamater > carState.speedLimit && maxSpeedParamater !
    {
        slowSpeedFactor = carState.speedLimit / maxSpeedParamater ;
    }

    targetSpeed.vx *= slowSpeedFactor;
    targetSpeed.vy *= slowSpeedFactor;
    targetSpeed.vphi *= slowSpeedFactor;
    TARGETSPEED_set(&so_targetspeed, targetSpeed);

    return SAFEOK;
}
```

Daten einlesen (E)

Daten verarbeiten (V)

Daten ausgeben (A)

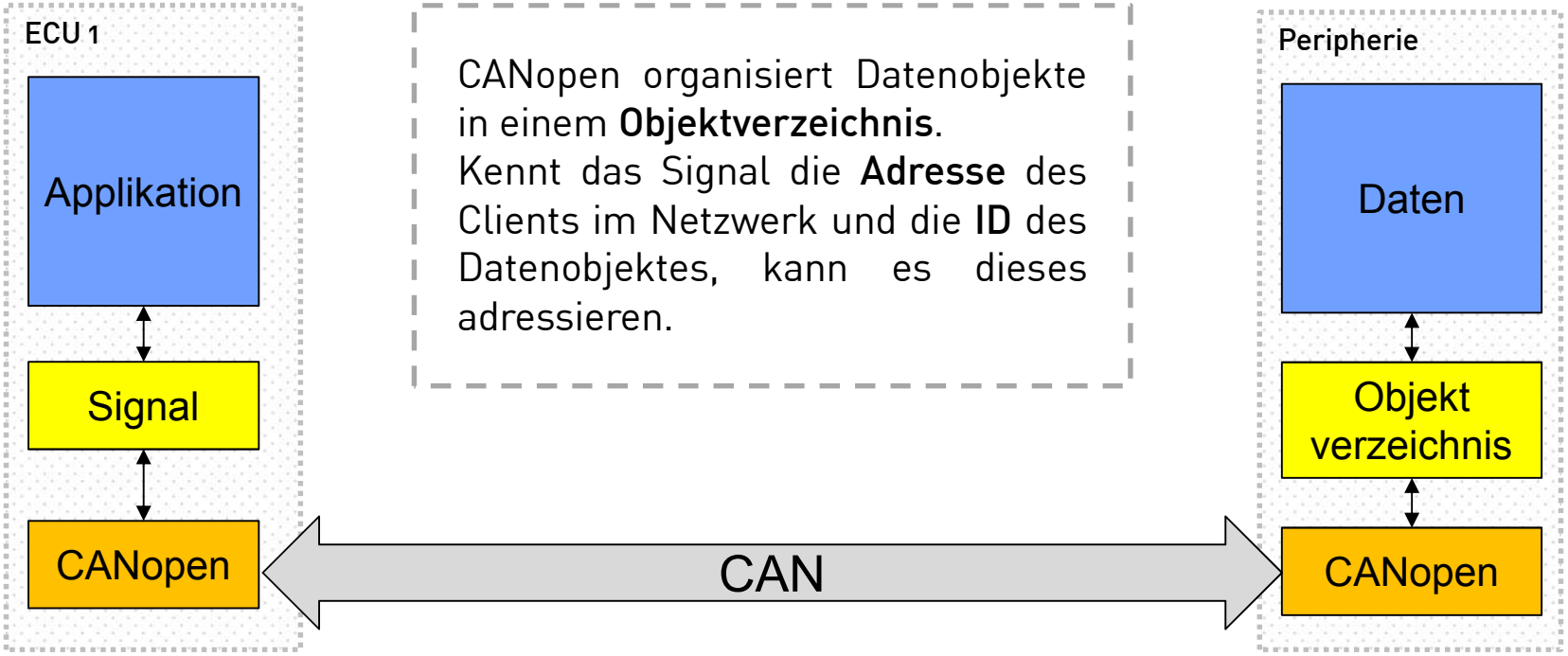
Einhängen der Runnables in die Laufzeitumgebung

```
/*
 * Cyclic Runnables Table
 * Entry to the table shall be made as
 * runnable ptr,        cyclic Time,        relative Offset Time
 */
const RT_eventSafetyTable_t RT_C0_safetyLogic_eventRunTable[] = {

      //runnable ptr,                                                    eventMask
      {(RT_runnableSafetyPtr_t)(SAFETYLOGIC_run_checkTargetSpeedUpdate),        ev_Task_C0_Safety_Logic_0_100ms},
      {(RT_runnableSafetyPtr_t)(SAFETYLOGIC_run_checkEngineTargetSpeed),        ev_Task_C0_Safety_Logic_0_100ms},
      {(RT_runnableSafetyPtr_t)(SAFETYLOGIC_run_setEngineSpeedCarState),        ev_Task_C0_Safety_Logic_0_100ms},
      {(RT_runnableSafetyPtr_t)(DR_run_getDecoder),                             ev_Task_C0_Safety_Logic_0_100ms},
      {(RT_runnableSafetyPtr_t)(ENGINE_run_emergencyStop),                      ev_Task_C0_Safety_Logic_2_requestEmergencyStop},
      {(RT_runnableSafetyPtr_t)(ENGINE_run_stopCar),                            ev_Task_C0_Safety_Logic_3_requestStop},
      {(RT_runnableSafetyPtr_t)(ENGINE_run_setSpeed),                           ev_Task_C0_Safety_Logic_1_newTargetSpeed}
};  /*
     * Cyclic Runnables Table
     * Entry to the table shall be made as
     * runnable ptr,        cyclic Time,        relative Offset Time
     */
    const RT_cyclicSafetyTable_t RT_C0_safetyInput_cyclicRunTable[] = {

          //runnable ptr,                                                  cyclic Time,    Offset
          {(RT_runnableSafetyPtr_t)(REMOTE_run_readAndCheckProtocol),          10,            0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkJoystickData),        10,            0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkJoystickZeroLeavingSafe2State), 10,  0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkRemoteTransmissionProtocol), 10,     0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkZigbeeSignalStrength),  100,         0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkEngineControlHeartBeat), 10,         0},
          {(RT_runnableSafetyPtr_t)(SAFETYINPUT_run_checkCollissionSensor),    200,           0},

    };
```

Erweiterung: Anbinden von Kommunikations-Stacks

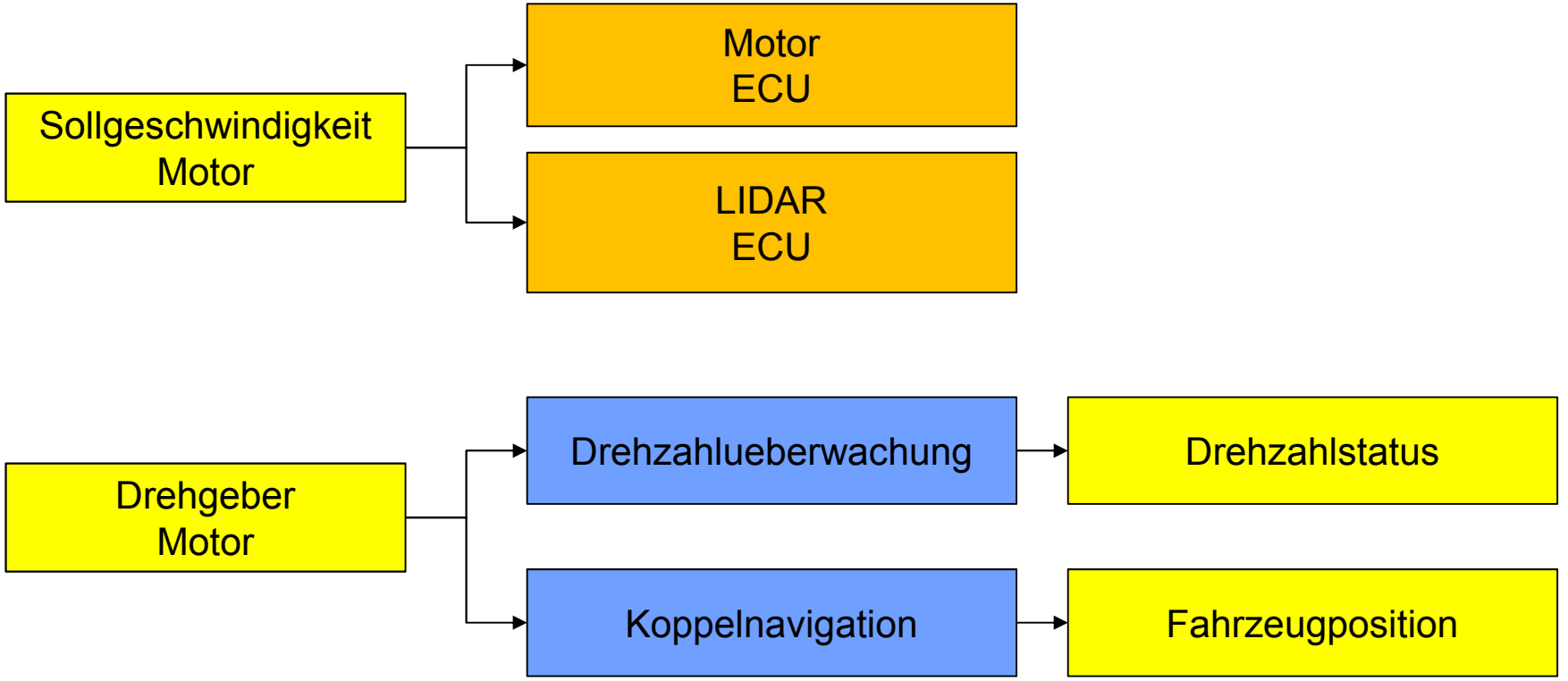
Unter Berücksichtigung von Latenz und Jitter ist es irrelevant, ob sich Signalquelle und –senke lokal oder im Netzwerk befinden. Auch eine Restbus- oder Peripheriesimulation wird möglich.

Beispiel CANopen:



Unterstützung von Broadcasting

- Signale sollen konsistent an mehrere Empfänger gesendet werden
- Unterstützung von parallelen Signalpfaden



Zusammenfassung und Ausblick

- Nach 3 Monaten lief ein erster Software Stand bestehend aus
  - Lines 34.793
  - Files 162
  - Functions 588

- Timing
  - Zykluszeiten der Tasks im Bereich 1-100ms
  - Latenzzeit Signalbenachrichtigung: ca. 4-8 µs
  - Data access: ca. 0.3 – 2µs for one int depending on memory location

- Die Laufzeitumgebung befindet sich bei ersten Kunden im Einsatz

- Der Workflow muss weiter automatisiert und vereinfacht werden.

Kontakt

M.Sc. Thomas Barth
Prof. Dr.-Ing. Peter Fromm

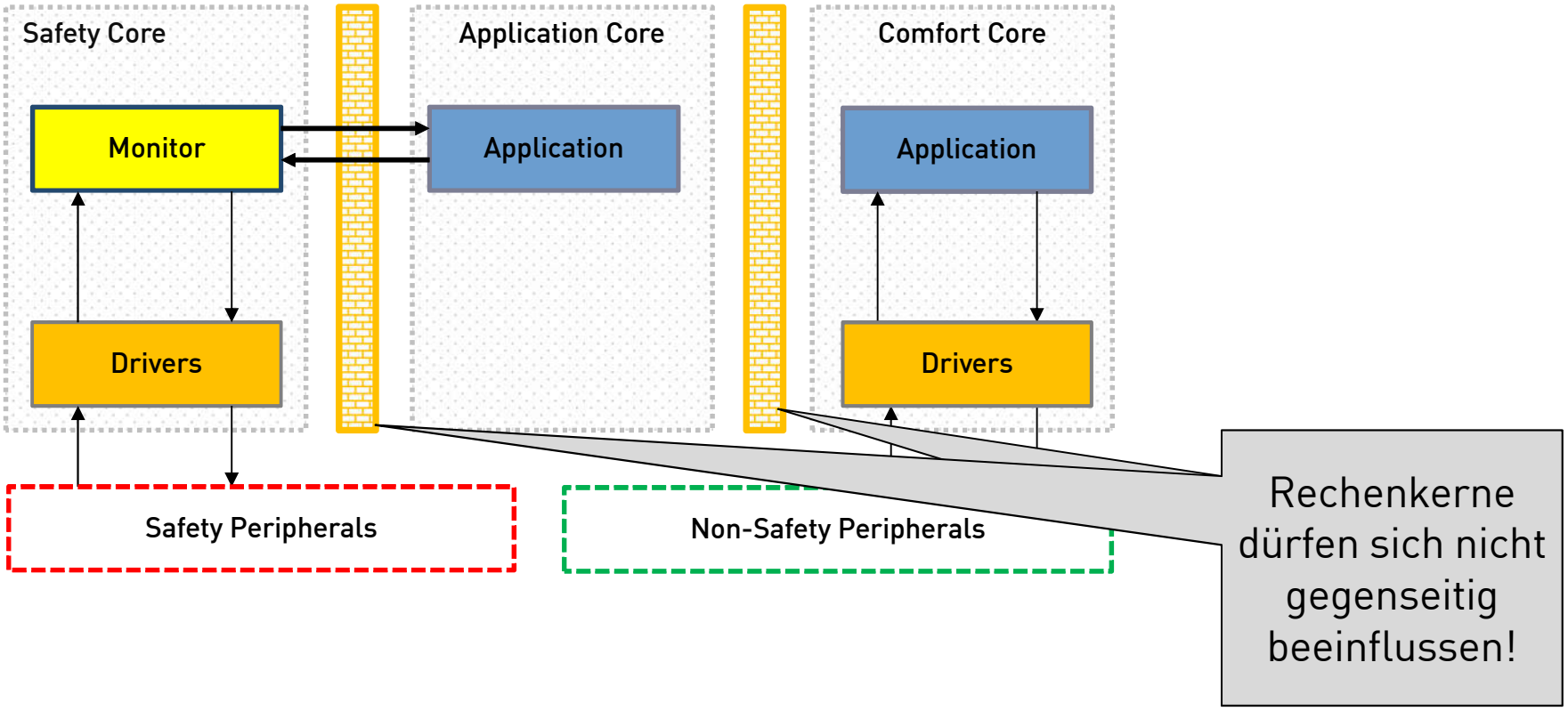
Hochschule Darmstadt - University of Applied Sciences
FB Elektrotechnik und Informationstechnik (EIT)
Birkenweg 8
64295 Darmstadt

Email: thomas.barth@h-da.de; peter.fromm@h-da.de
Web: www.eit.h-da.de

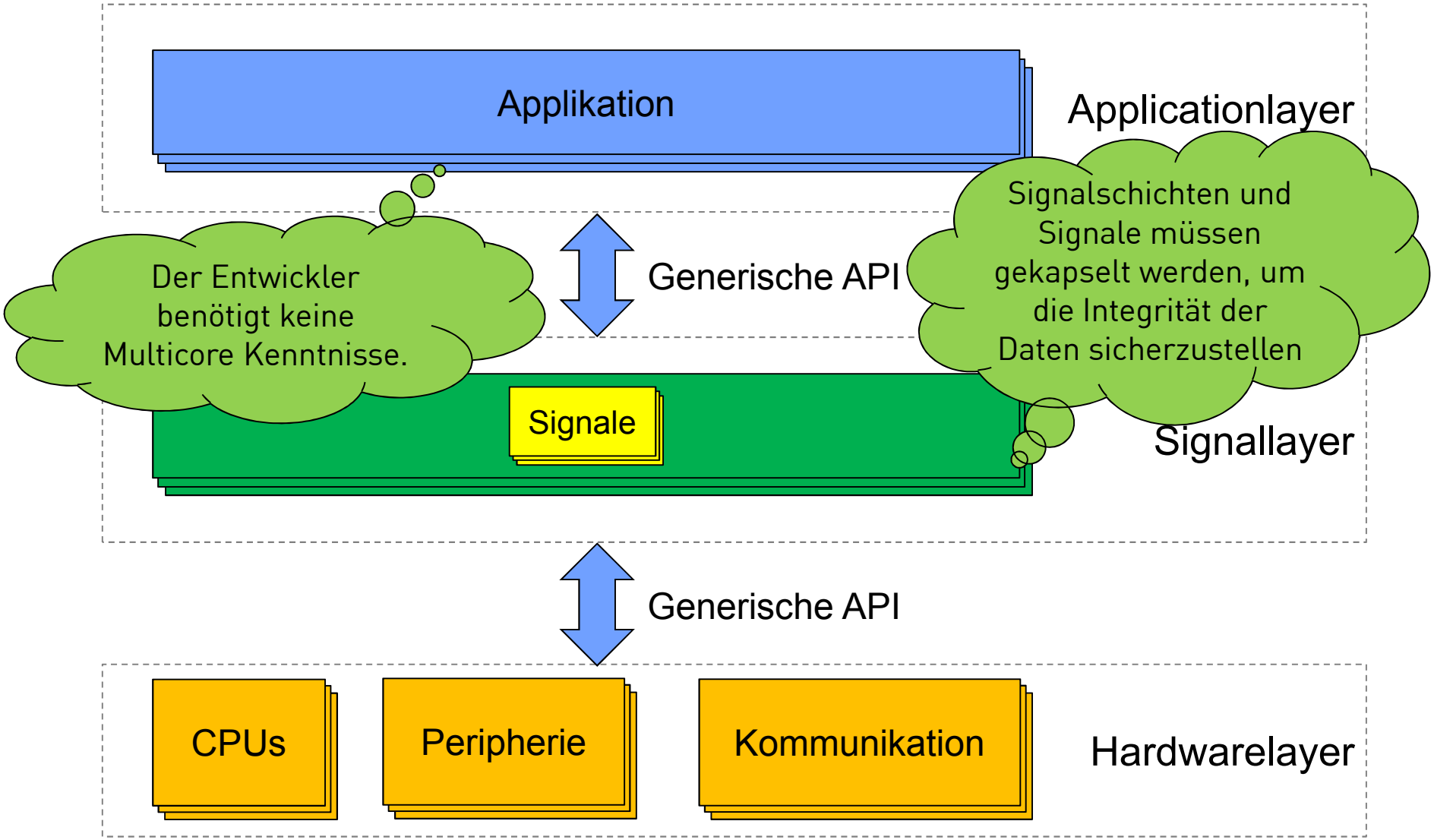
BACKUP

Applikative Kapselung auf Multicore

Entwickeltes Architekturmuster zur applikativen Kapselung.
Applikation hat keinen direkten Zugriff auf Peripherie.



Architektur Laufzeitumgebung



Hardware-Support für Kapselung

CPU-MPUs

- Überwachung ausgehenden Traffics (lesen, schreiben, ausführen)
- Dynamische Rekonfiguration durch RTOS PxROS der Firma HighTec

Bus-MPUs

- Zuweisen von Schreibrechten für definierbare RAM-Speicherbereiche an frei definierbare Bus-Teilnehmer (z.B. CPUs)

RAP (Register Access Protection)

- Zuweisen von Schreibrechten auf Peripheriemodule (z.B. GPIO Ports) an frei definierbare Bus-Teilnehmer (z.B. CPUs)