HotSys Praktikum

l4*Copter*: Architektur und Aufbau

01.12.2010

Peter Ulbrich

http://www4.cs.fau.de/~ulbrich
ulbrich@cs.fau.de

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universtät Erlangen-Nürnberg

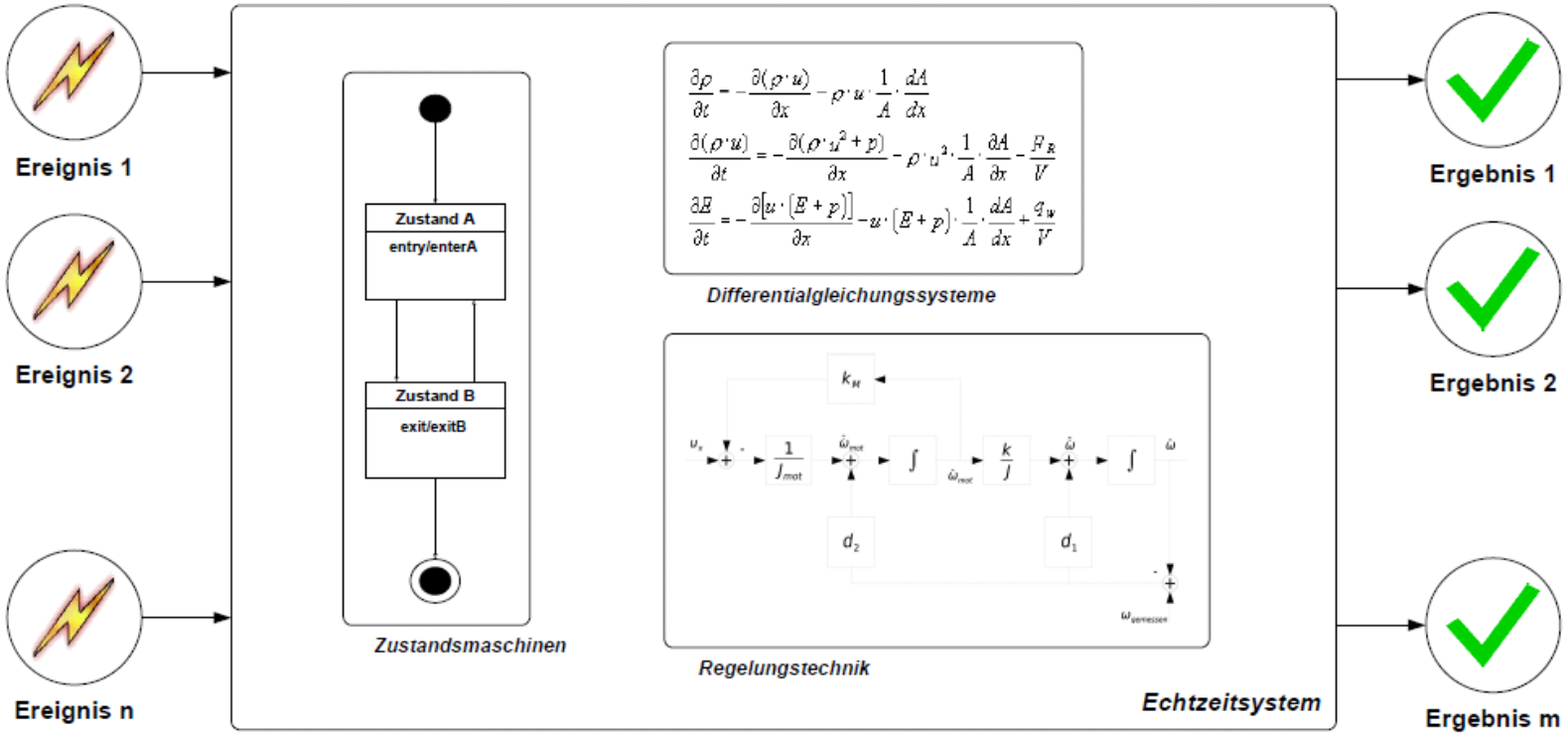
Übersicht

- **Modulorientierte Entwicklung von (Echtzeit-) Systemen**
  - Phase 1: Einordnung / Analyse
  - Phase 2: Modularisierung
  - Phase 3: Komposition
- ***I4Copter***
  - Projektübersicht
  - Hardware
  - Software

Modularisierung
Phase 1 – Analyse

Einordnung – Phase 1

- Beziehungen zwischen **Ereignis n** und **Ergebnis m**
  - **zeitlich** – wie viel Zeit darf verstreichen → Termine
  - **physikalisch** – wie ist das Ergebnis zu bestimmen?



Zielsetzung

- physikalisches Objekt
  - Welche Größen sind relevant?
  - Wie hängen diese Größen zusammen?
- Echtzeitsystem
  - Welche Ereignisse gilt es zu behandeln?
  - Welche Zeitschranken gilt es einzuhalten?
  - Welche Beziehung Zeitschranke ↔ physikalisches Objekt gibt es?
- Wie sieht das physikalische Modell aus?
  - Welche Größen des physikalischen Objekts muss man abbilden?
  - Wie bildet man diese Größen ab?

Modularisierung

Phase 2 – Modularisierung

Identifikation von Modulen – Phase 2

- **Was** ist ein Modul?
- Wie **teilt** man ein System in Module auf?
- **Welche** Module gibt es in meinem System?
- Wie sind diese Module **gekoppelt**?
- Wie **interagieren** diese Module?
- Welche Module kann man **zusammenfassen**?

Was ist eine Komponente / ein Modul?

- Softwarekomponente
  - Klasse
  - Prozedur
  - Package
  - Übersetzungseinheit
- hat eine **wohldefinierte Funktion**
- hat eine **wohldefinierte Schnittstelle**

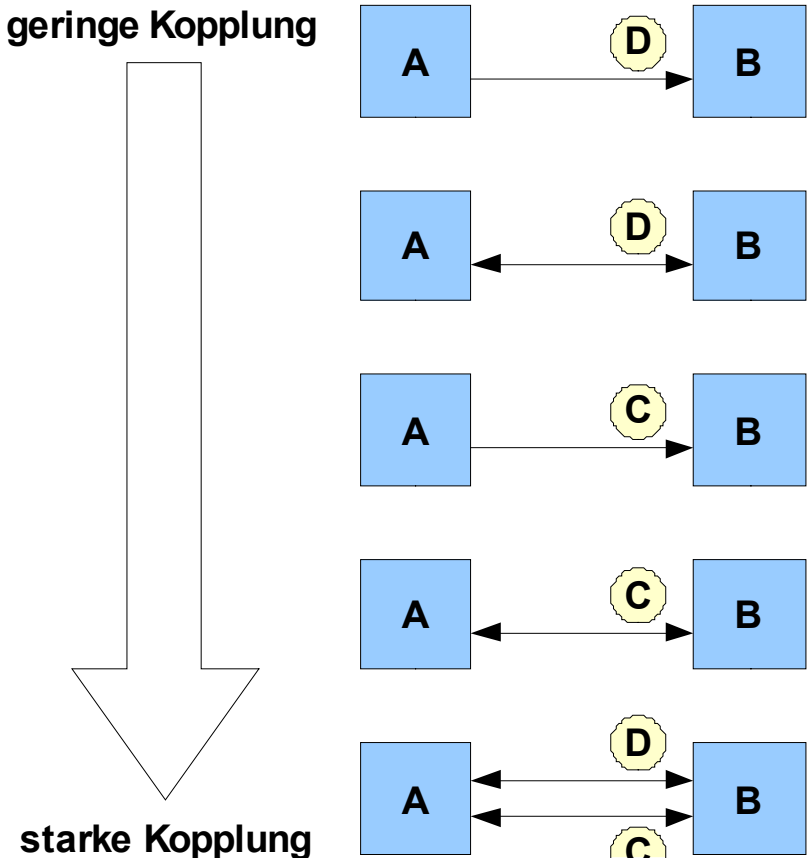
Wie gliedert man ein System in Module?

- **Algorithmus existiert nicht!**
- ist **anwendungsbezogen** – hängt vom System ab
- ist **subjektiv** – hängt vom Designer ab
- mögliche Kriterien
  - **Partitionierung** des Systems
  - **Größe** der Module
  - **Komplexität** der Module
  - **externe Schnittstelle** der Module
  - **Beziehung** und **Kommunikation** der Module untereinander
    - sowie die daraus resultierende **Programmhierarchie**
  - Bereich der **Kontrolle** und des **Einflusses** eines Moduls
- erfordert **Erfahrung**

Kopplung von Modulen

- Maß für die **Unabhängigkeit** von Modulen
- niedrige Kopplung
  - einfach zu verstehen
  - wartbar
  - verlässlich
  - stabil
- hohe Kopplung
  - hohes Maß an ***„Collateral Evolution“***
- Hauptfaktoren
  - Komplexität von Information
  - Informationstyp
  - Kommunikationsmethode
  - Art der Kopplung

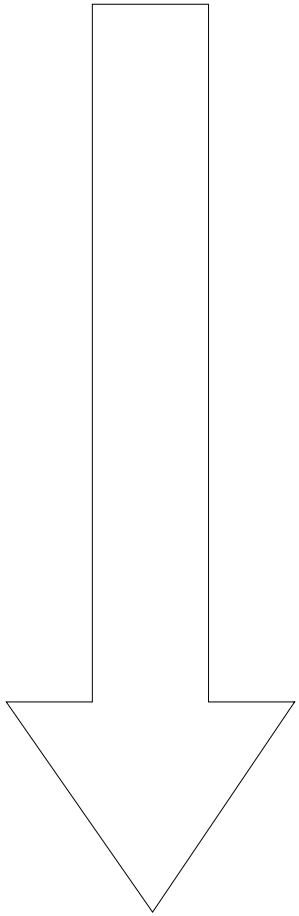
Kopplung – Informationstyp



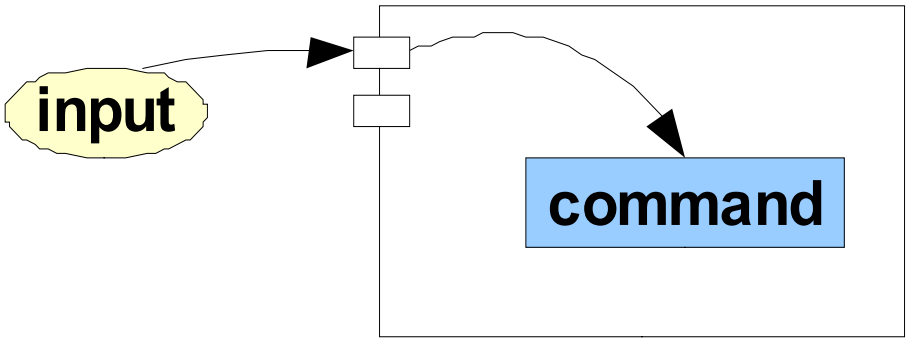
- unidirektionaler Datenfluss
- **bidirektionaler Datenfluss**
- **unidirektionaler Kontrollfluss**
- **bidirektionaler Kontrollfluss**
- **bidirektionaler Daten- und Kontrollfluss**

Kopplung – Kommunikationsmethode

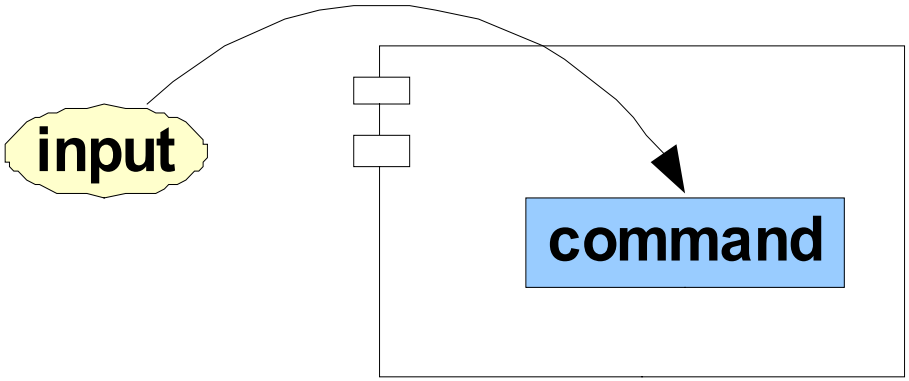
geringe Kopplung



starke Kopplung

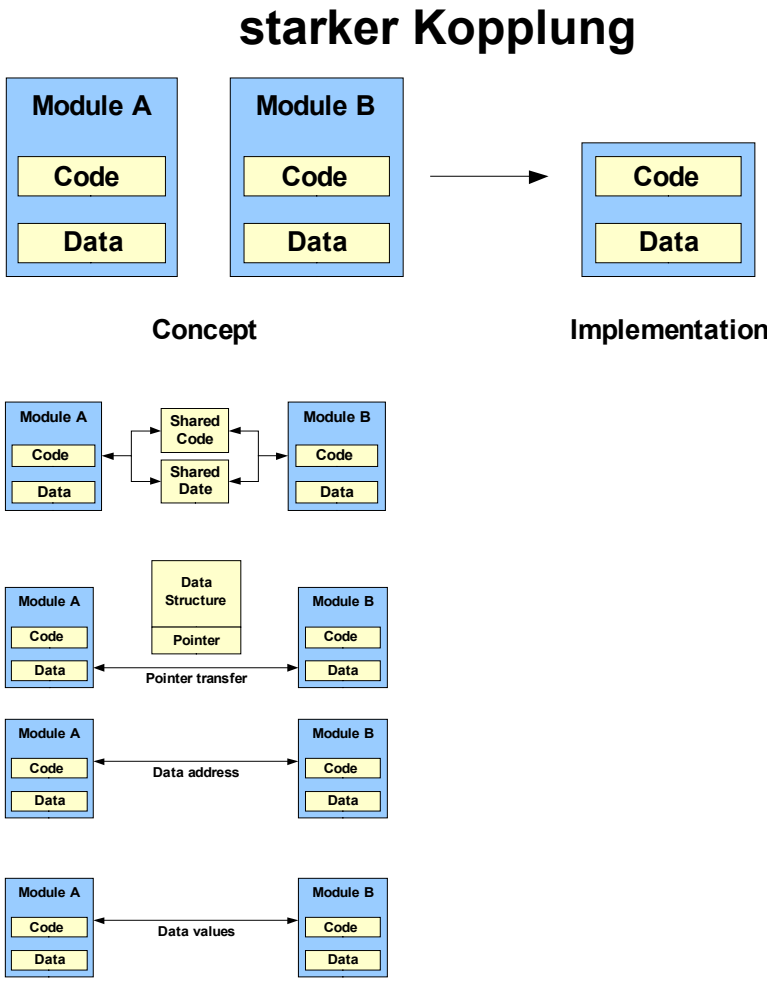


normal module connection



pathological module connection

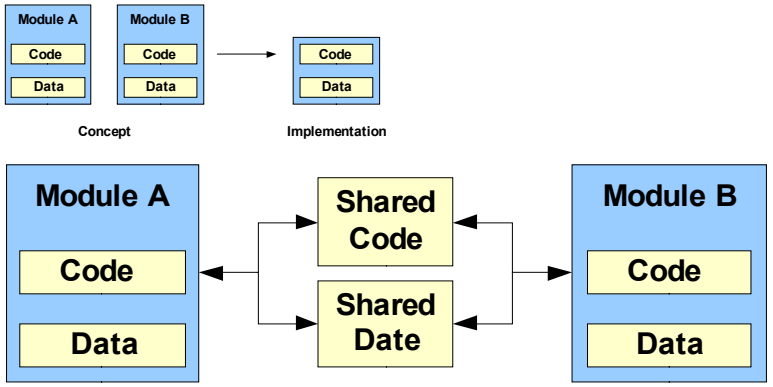
Kopplung – Art

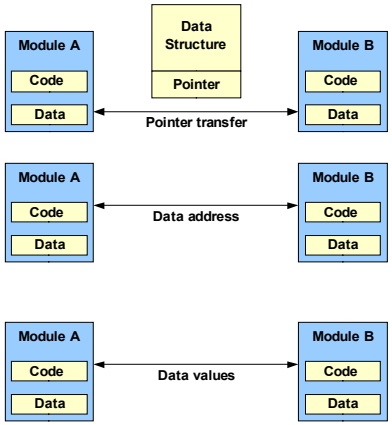


- **Content Coupling**
  - Module existieren im Konzept
  - keine Trennung in der Implementierung
- Common Coupling
- Stamp Coupling
- Data Coupling – by reference
- Data Coupling – by value

Kopplung - Art

starker Kopplung



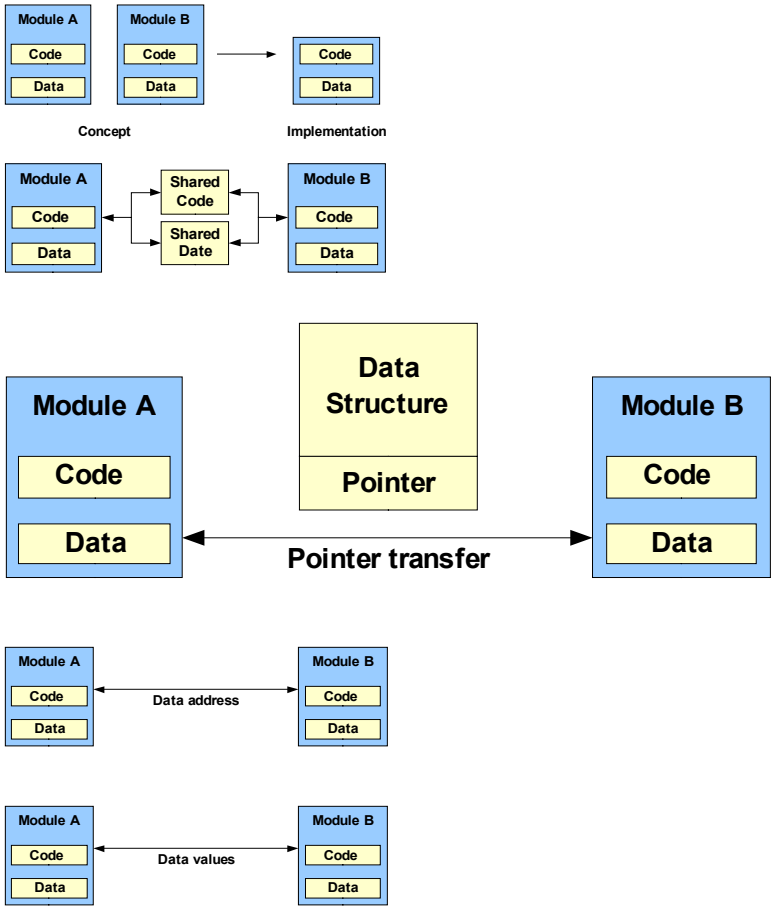


- Content Coupling
- **Common Coupling**
  - Unterscheidung: globale vs. private Daten- und Programmbereiche
  - globale Bereiche von allen Modulen zugreifbar
- Stamp Coupling
- Data Coupling – by reference
- Data Coupling – by value

geringe Kopplung

Kopplung - Art

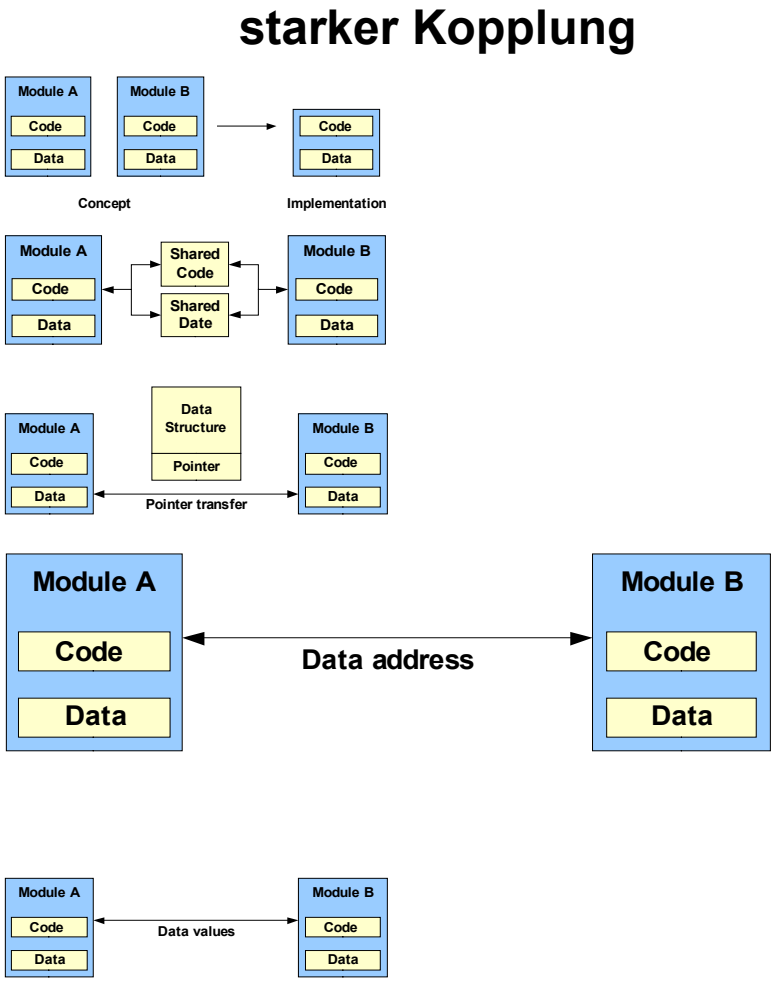
starker Kopplung



- Content Coupling
- Common Coupling
- **Stamp Coupling**
  - spezifische Datenstruktur statt globaler Bereiche
  - nur bestimmte Bereiche werden geteilt
  - teilende Module müssen konsistente Sicht auf die Datenstruktur haben
- Data Coupling – by reference
- Data Coupling – by value

geringe Kopplung

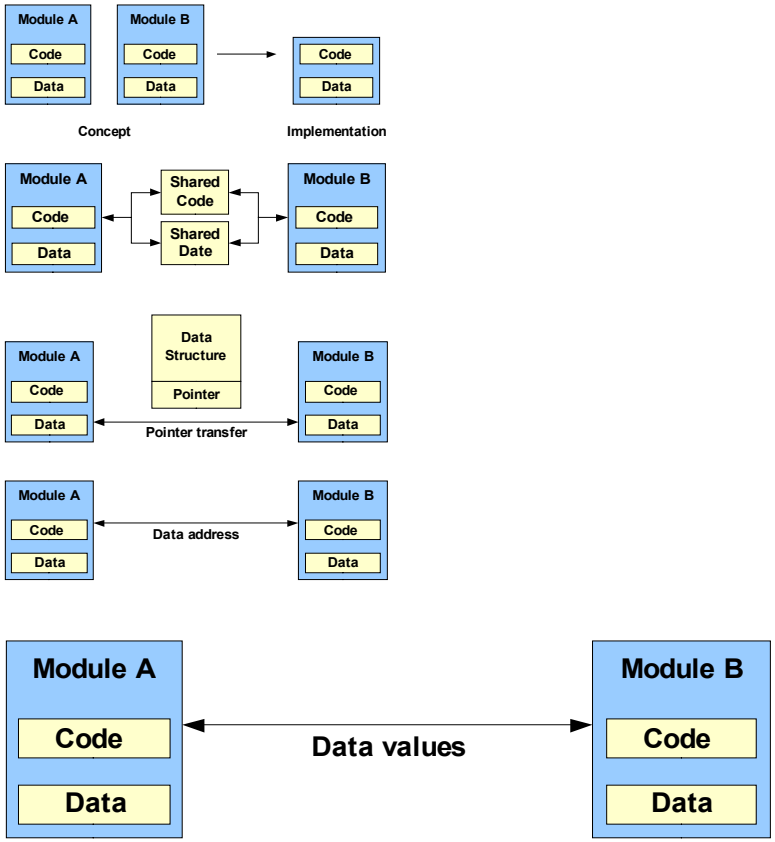
Kopplung - Art



- Content Coupling
- Common Coupling
- Stamp Coupling
- **Data Coupling – by reference**
  - Zugriff auf Daten per Referenzen
  - Konsistenz der Daten durch Programmiersprache sicher gestellt
- Data Coupling – by value

Kopplung - Art

starker Kopplung



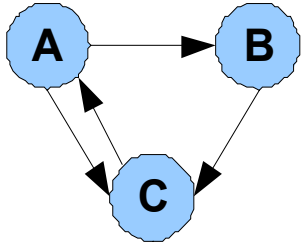
- Content Coupling
- Common Coupling
- Stamp Coupling
- Data Coupling – by reference
- **Data Coupling – by value**
  - nur Werte werden weiter gegeben
  - Module können Zustände anderer Module nicht ändern
  - sehr sicher
  - hoher Speicherbedarf

geringe Kopplung

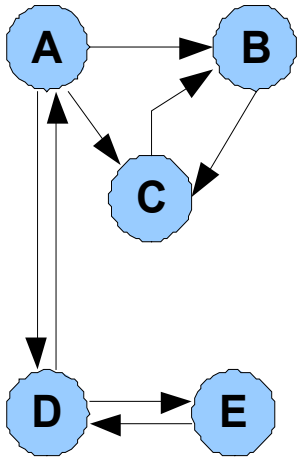
Kohäsion

- ideales Module
  - sehr einfach
  - 1 Eingang, 1 Ausgang
- einfache Systeme
  - übersichtlich
- komplexere Systeme
  - Systemkomplexität steigt schnell an
  - unübersichtlich
- Kompromiss
  - einfache Module vs. einfaches System

ideales Modul



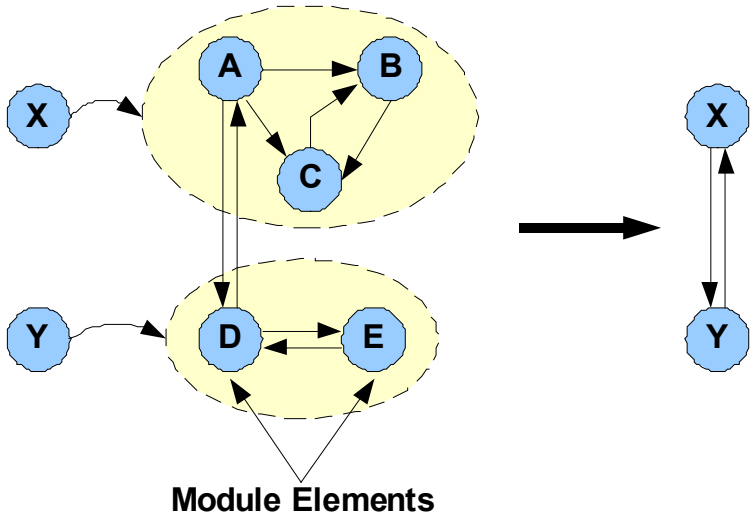
einfaches System



komplexeres System

Kohäsion

- fasse Module zu Super-Modulen zusammen
  - Module werden Elemente des Super-Moduls

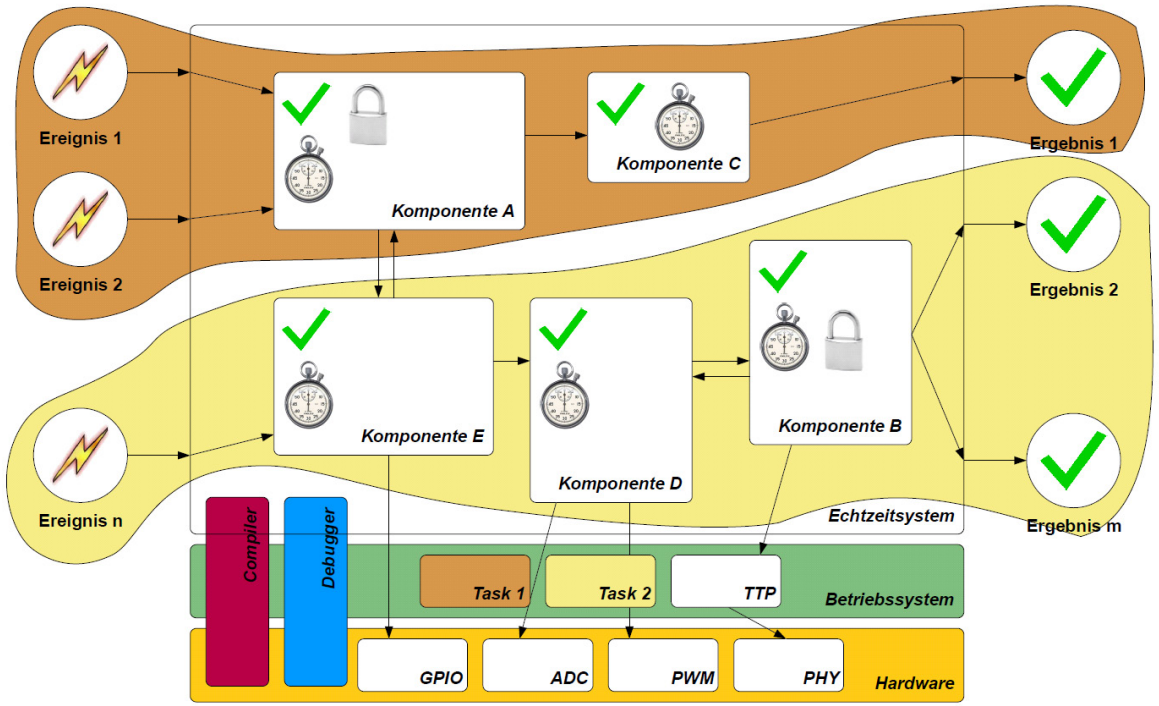


- Kohäsion: ein Maß für den **Zusammenhalt** der Elemente
  - Anzahl der Verbindungen **untereinander**
  - Anzahl der Verbindungen **nach außen**

Modularisierung
Phase 3 – Komposition

Phase 3 – Komposition

- **Abbildung:**
  - Komponenten → **Ereignisbehandlungen**
  - Ereignisbehandlungen → **Betriebsystem**
- **Planung:**
  - Prioritätenvergabe oder statische Ablaufplanung
  - Zulässigkeitsprüfung



Grundlagen

- Teil 1: **Abbildung**
  Ereignisbehandlungen → BS-Dienste
  - Verknüpfung von Ereignissen mit **Ereignisbehandlungen**
  - **Abhängigkeiten** verschiedener Ereignisbehandlungen
  - **Synchronisation** kritischer Abschnitte
  - Modellierung von **Datenabhängigkeiten**
  - ...
- Teil 2: **Ablaufplanung**
  - Berechnung **statischer Ablaufpläne**
  - Auswahl eines geeigneten **Algorithmus** (RMA, DMA, EDF, ...)
  - **Planbarkeitsanalyse**: Antwortzeitanalyse, CPU-Auslastung, ...
  - ...

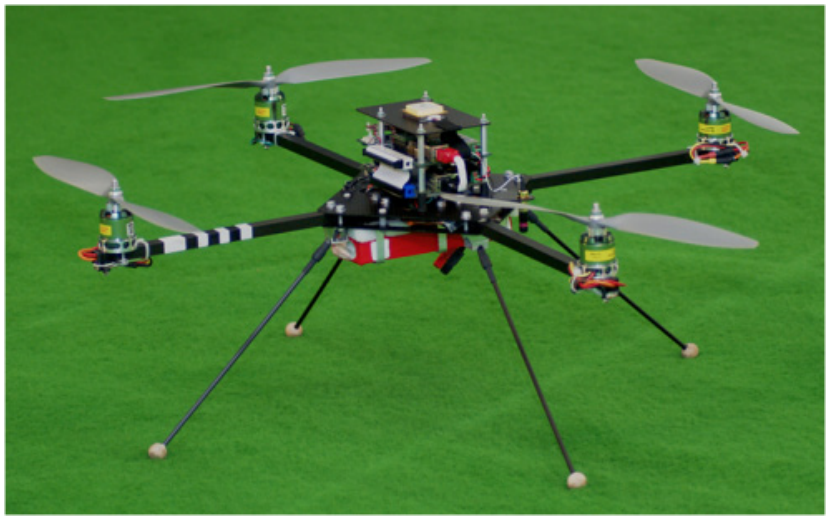
Komposition: Eingaben

- **Ereignisbehandlungen**
  - welche **Ereignisse**:
    - zeitliche Parameter: periodisch, aperiodisch, sporadisch
    - Herkunft: physikalische bzw. logische Ereignisse
  - welche **Termine**: hart, fest, weich
  - **Abhängigkeiten**
    - gerichtete Abhängigkeiten
    - gegenseitiger Ausschluss ↔ kritische Abschnitte
- **Komponenten**
  - funktionale und temporale Spezifikation
- **Laufzeitsystem**
  - funktionale und temporale Spezifikation

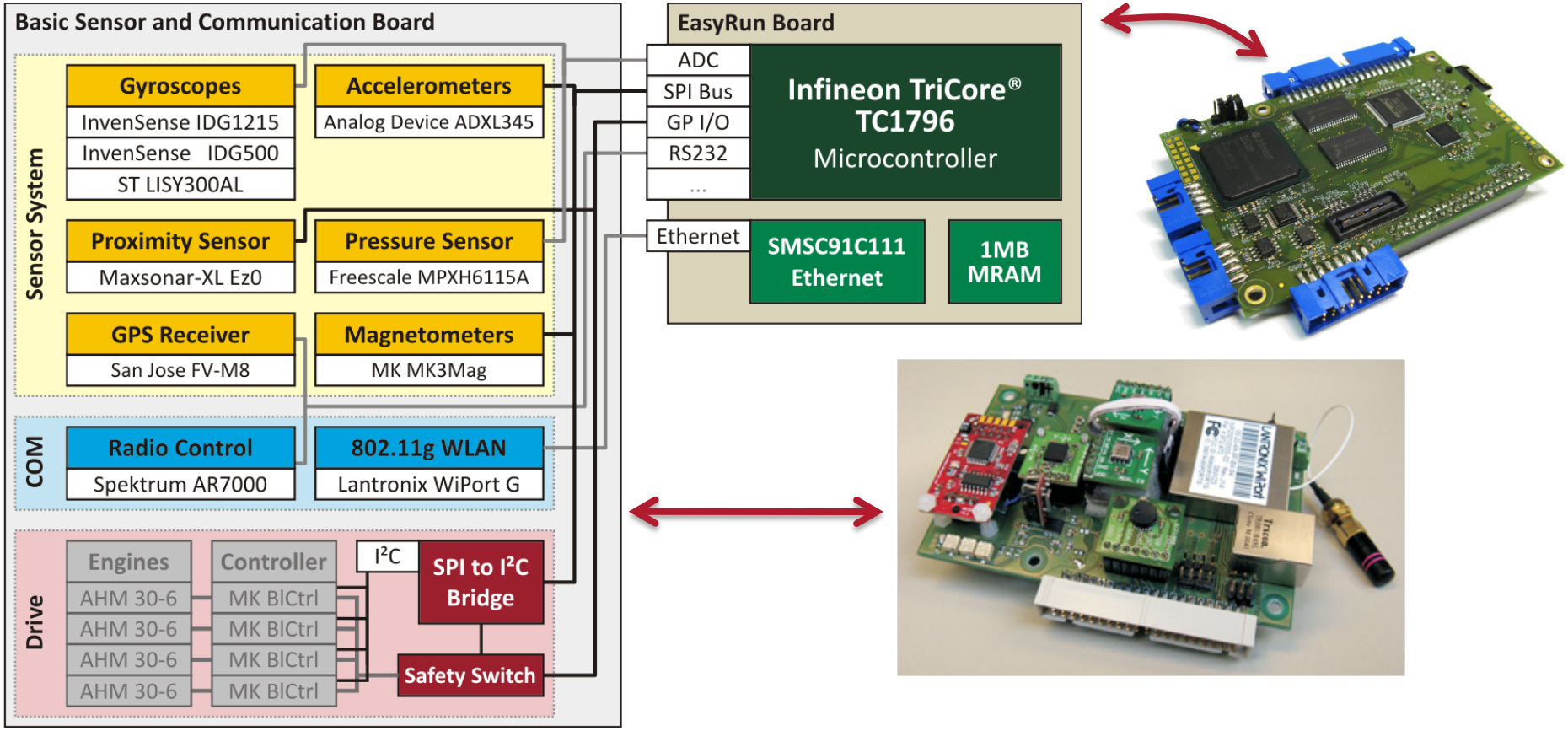
I4Copter
Architektur & Aufbau

I4Copter – Übersicht

- **Component Architecture for Safety-critical Embedded Systems**
  - Term: 08/2007 → now
- ***CoSa*** Component Framework
  - Basic Framework: Reuse & Abstraction
  - Component isolation and interaction
  - Sensor system abstraction & sensor fusion components
- ***CoRed*** Software Redundancy
  - Component-level software redundancy
  - Input-to-output protection
- ***I4Copter*** Evaluation System
  - Quadrotor helicopter
  - "Source of inspiration"



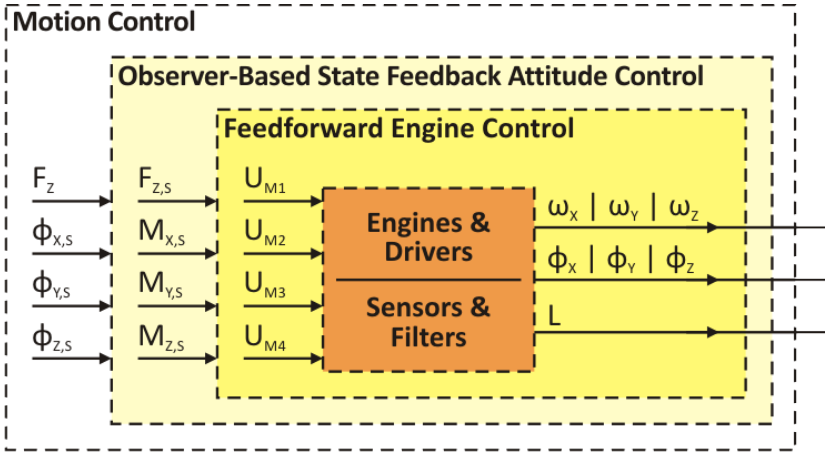
I4Copter Abstract – Hardware



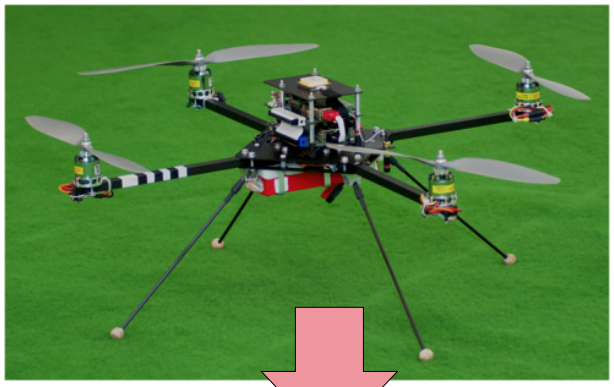
- **Resembles automotive and industry applications**
- **Custom made sensor periphery board**
  - 8 sensors, heterogeneous interfaces
- **Automotive microcontroller**
  - Infineon TriCore TC1796
  - 150MHz / 64KB+1MB RAM
- **3 units build: Ikarus, Apollo, Deadalus**

I4Copter Abstract – Software & Control

- **I4Copter flight-control application**
  - Based on CoSa component framework
  - Written in **C++**, comprises **~26.000LOC**
  - **Core components:** signal processing, flight control, behaviour control, navigation
- **Feedback control system**
  - **Model-based** controller and plant → Matlab Simulink
  - Main attitude controller: Three layers cascaded structure
  - Altitude control (beta)
  - Trajectory tracking control (under devel.)
- **Sensor fusion & state observation**
  - Orientation: Tilt angle, direction (heading)
  - Altitude, vertical acceleration and speed



I4Copter – Abstract View



Command

Monitoring

Control

Sensors

Flight Control

Copter Control

Signal Processing

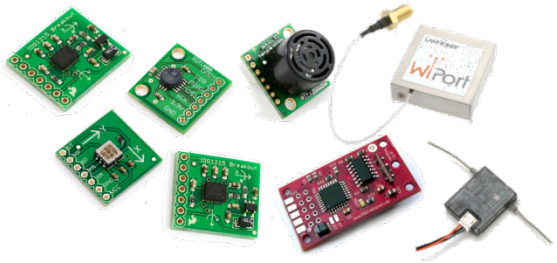
Safety Switch

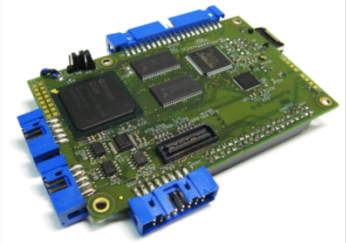
Engine Voltage

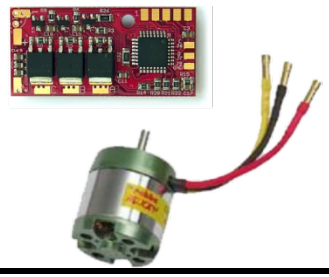
Sensor Data Acquisition

Processing & Controlling

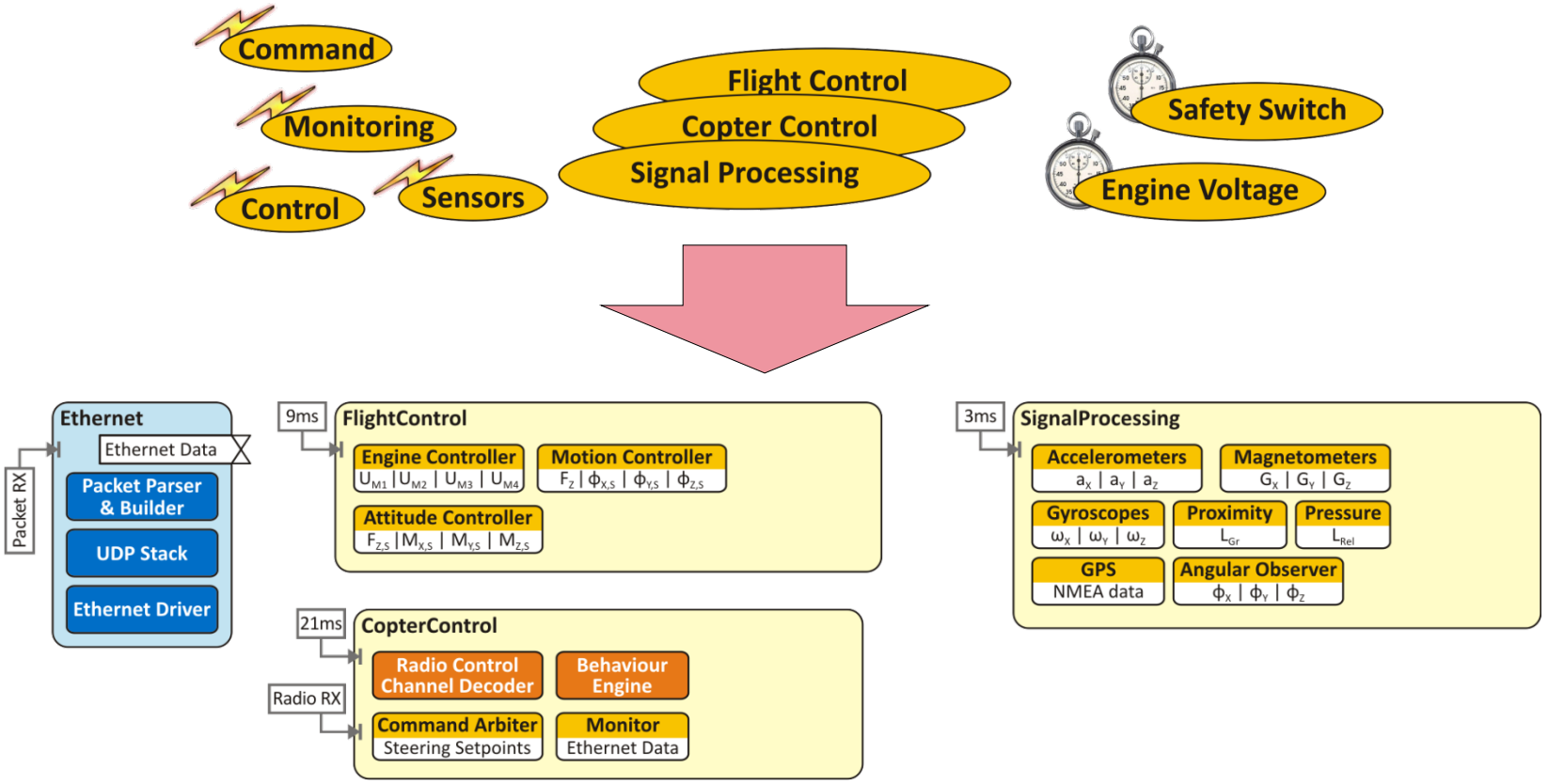
Actuator Output





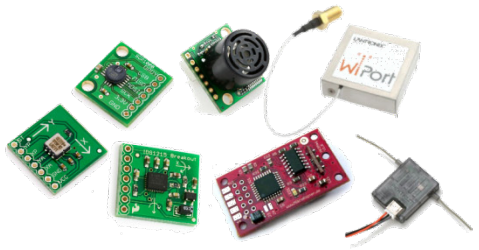


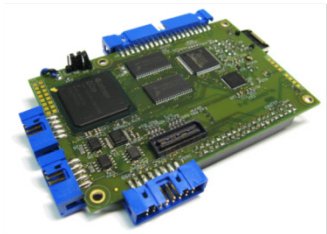
Framework basics – Components

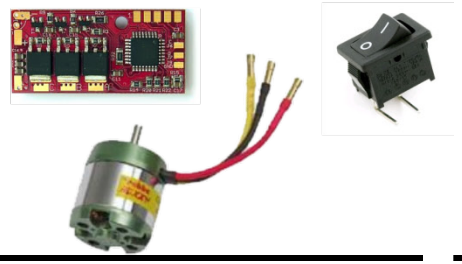


Framework Basics - Abstraction

- **Component reuse**
  - Hardware abstraction
  - OS abstraction
  - Basic device drivers
- **TriCore HAL**
  - Analog, digital or bus channel → Application specifies output port
  - Implements hardware internal routing and configuration
  - Prevents double assignments



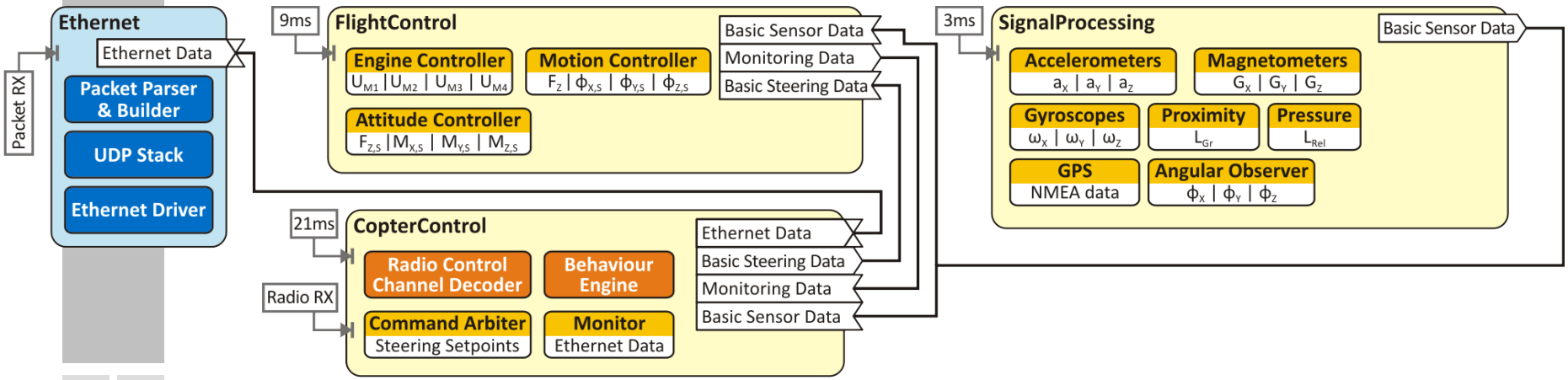




Isolation – Component Connectors

➔ **Distributing state messages between protection domains**

- **Component isolation**
  - **Spatial:** Utilising TriCore memory protection (OS support)
  - **Temporal:** Utilising real-time operating system
- **CoSa connector pattern**
  - Overcome memory protection boundaries
  - Automatic mapping to appropriate mechanism (message, shared memory)

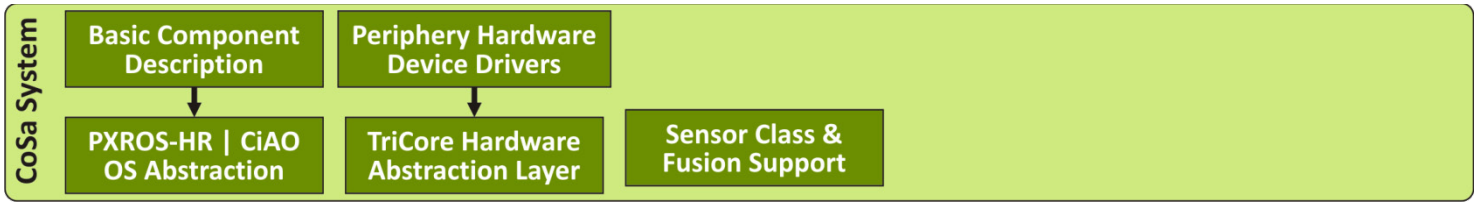


CoSa Sensor Abstraction

➔ **Unhinge sensor processing and control engineering**

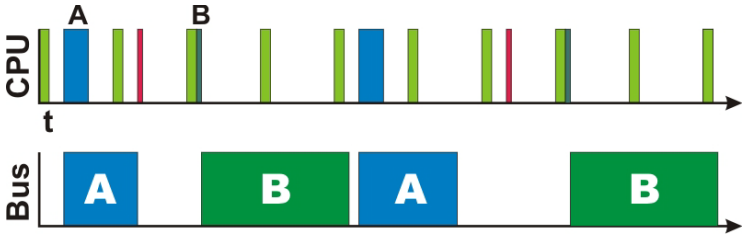
- **CoSa sensor class components**
  - Persistent physical measurement → Interface for controller
  - Sensor device driver(s) abstraction
  - Real-time and control theory requirements

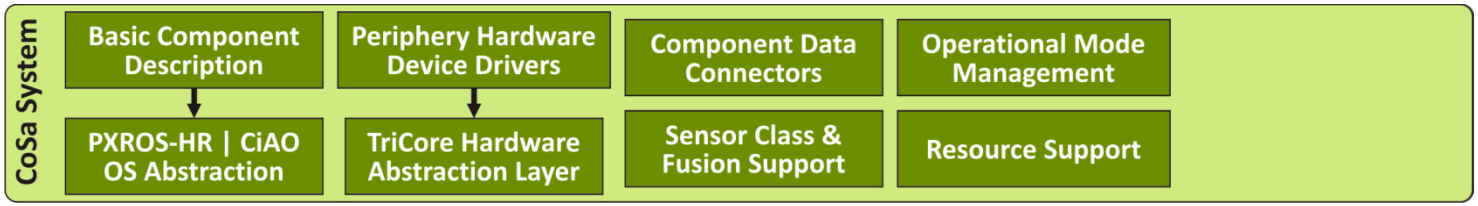
- **CoSa sensor fusion components**
  - Extending sensor class components
  - Deviation criterion



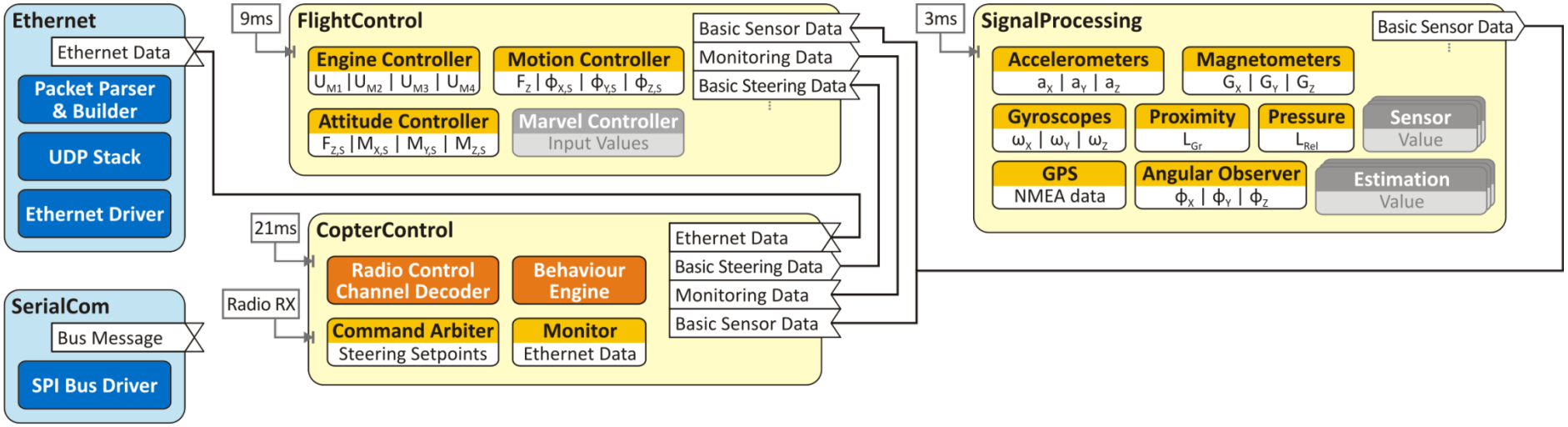
CoSa Real-Time Analysis Support

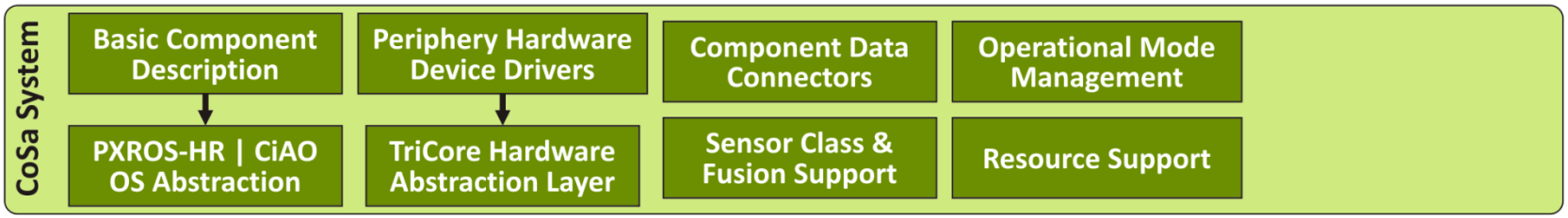
- **Dealing with complex jobs**
  - Common resource analysis
  - Implicit vs. explicit synchronisation
- **Operational Mode Management**
  - Round-based arbiter for component-mode
- **Exception Handling**
  - Connector like
  - Coupled with debugging support





CoSa based I4*Copter* Application

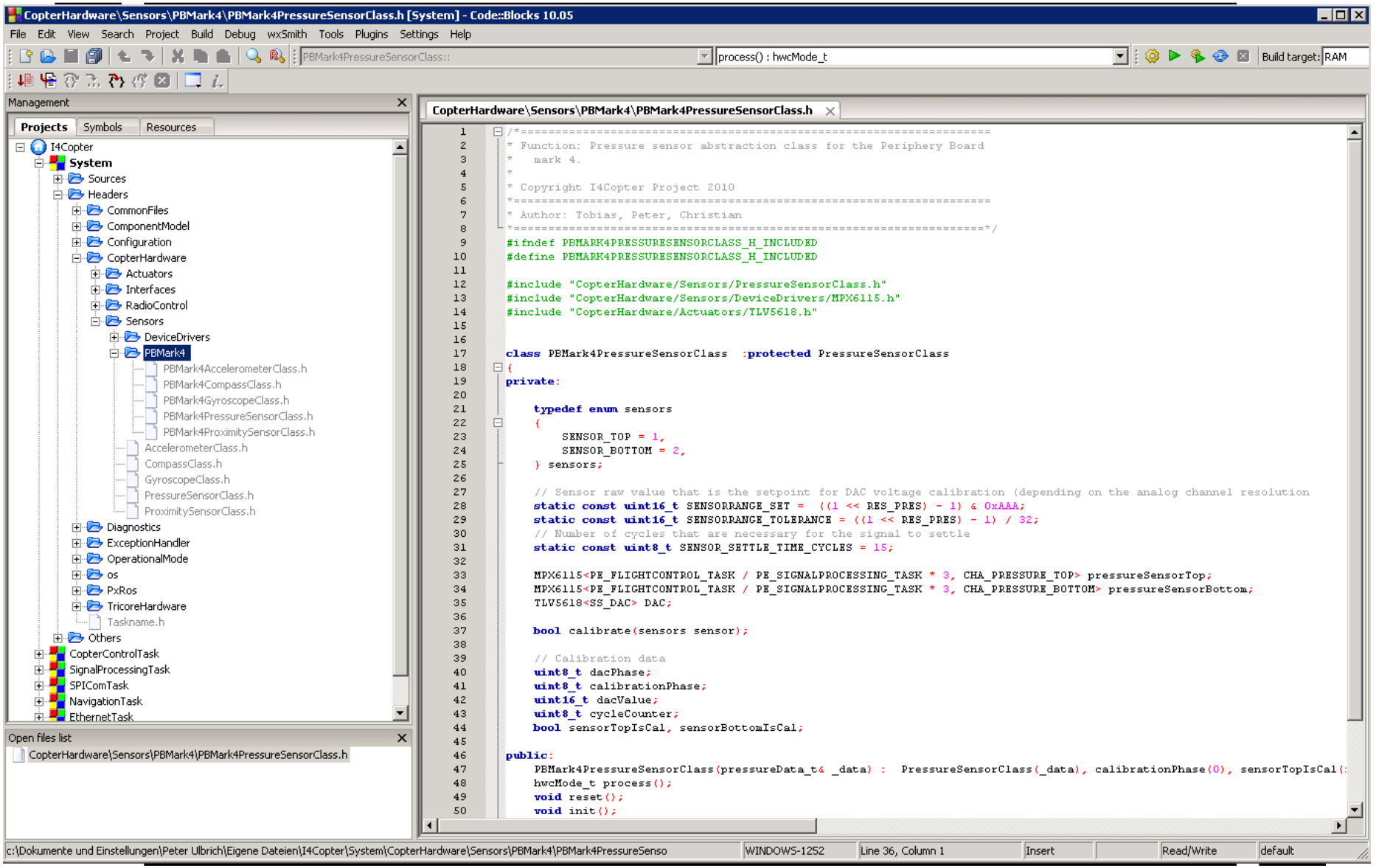




Getting started (1)

- I4*Copter* SVN Repository
  - `http://www4.informatik.uni-erlangen.de/i4cosa/I4Copter/I4Copter/trunk/`
  - **Login:** Peter
  - Codebasis enthält auch Code von Kooperationspartnern
    → **Weitergabe derzeit nicht erlaubt!**
- CodeBlocks Projekt (auch)
  - `I4Copter.workspace`
- Flashen mit dem Trace32
  - `System/ebu.cmm`

Getting started (2)



Ende

ENDE

Fragen?