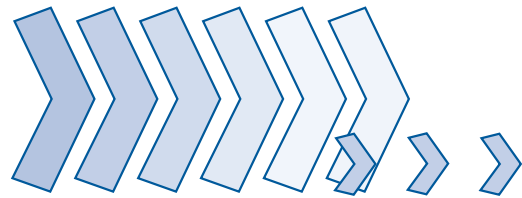


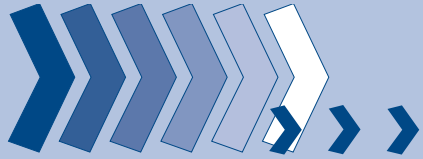
HUMBOLDT-UNIVERSITÄT ZU BERLIN



# Echtzeit-Anwendungen

Dipl.-Inf. J. Richling  
Wintersemester 2003/2004

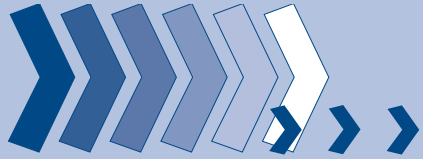
**EMES**



## *Anwendungsgebiete von Echtzeitsystemen*

- Fahr- und Flugzeuge
- Industrie-Automatisierung
- Haus-Automatisierung
- Telekommunikation
- Medizin
- Robotik
- Multimedia
- Geschäftssysteme
- ...

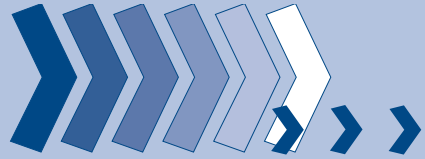
⇒ Echtzeitsysteme sind überall anzutreffen



## *Demonstration: Echtzeit und Multimedia*

... oder:

“Was passiert, wenn wir nicht über nichtfunktionale Eigenschaften nachdenken?”

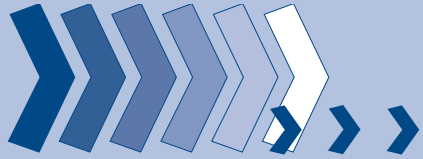


Im Beispiel:

- Inakzeptable Videoqualität
- Einfache Ad-hoc-Lösung
- Keine Garantien
- (weitgehend) ausreichend für Beispiel

In der Realität:

- Harte Deadlines
- Garantien sind erforderlich
- Lösungen müssen bewiesen werden



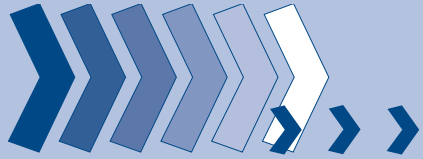
## *Beispiel: Flugzeug-Steuerung*

Hauptziel: Menschen dürfen nicht zu Schaden kommen!

Untergeordnete Ziele (ohne Wertung):

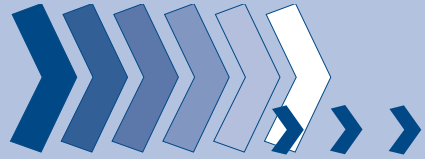
- Material soll nicht zu Schaden kommen
- Flugziel soll pünktlich erreicht werden
- Flugziel soll kostenoptimal erreicht werden
- Komfort für Menschen soll maximal sein
- Umwelt soll wenig belastet werden durch Abgase und Lärm

Problem: Abbildung auf technische Anforderungen (Tasks, Deadlines)



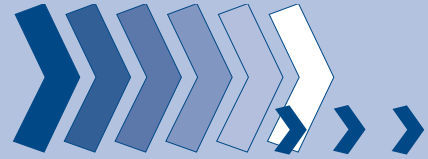
Welche Tasks werden gebraucht? (Beispiele)

- Aus Anforderungen der Flugzeugsteuerung:
  - Fluglage
  - Triebwerkssteuerung
- Aus Anforderungen der Mission:
  - Navigation
  - Kommunikation
  - Ortungssysteme
- Aus weiteren Anforderungen:
  - Komfortsteuerungen (Klima, Licht, Unterhaltungssysteme)
  - Service- und Diagnosesysteme



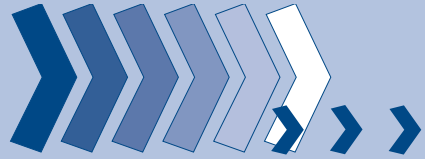
## Woher kommen die Deadlines der Tasks? (Beispiele)

- Abgeleitet aus physikalischen Notwendigkeiten
  - regelungstechnische Gesetzmäßigkeiten
  - Aerodynamik des Flugzeuges
- Abgeleitet aus Spezifikationen
  - Verlangtes Verhalten auf Stimuli (z.B. Ortung eines anderen Flugzeuges erfordert Reaktion in einem Zeitrahmen)
  - “Überdefinition” physikalischer Notwendigkeiten (Komfortfunktionen)



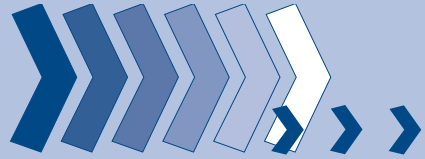
Tasks und Deadlines sind nicht in jedem Fall statisch!

- Mission eines Echtzeit-Systems besteht oft aus verschiedenen Stadien
- Jedes Stadium hat andere Anforderungen
  - Flugzeug beim Rollen auf dem Boden
  - Flugzeug beim Starten
  - Flugzeug beim Reiseflug
  - Flugzeug beim Landeanflug
  - Flugzeug beim Landen
  - Flugzeug bei der Wartung
- Betriebszustand in einem Stadium nennt man “Modus” (*mode*)
- Deadlines und Tasks sind mode-spezifisch
- Wechsel zwischen Modi (*mode changes*) geben statischen Systemen dynamisches Verhalten
- Mode Changes sind ressourcenschonend



### Problem: Verhalten beim Mode Change

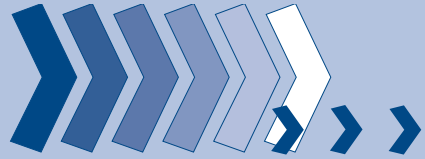
- Es gibt Tasks, die nur in einem Mode vorhanden sind: kein Problem, Starten, bzw. Beenden
- Es gibt Tasks, die übernommen werden müssen
  - Jitter oder verpaßte Deadlines beim Wechsel stören nicht  
→ Einfache Behandlung (einfach umschalten)
  - Tasks, deren Deadlines auch beim Umschalten eingehalten werden müssen und weitgehend jitterfrei laufen müssen, erfordern den größten Aufwand



- Mode Change ist ein eigener Schedule zur Überführung (Überführungs-Schedule)
- Deadline für Mode Change (vom Request bis zum Beginn des neuen Modes)
- Da das System statisch ist, kann der Mode Change Schedule vor Laufzeit erzeugt werden

Problem: Wie wird ein solcher Schedule berechnet?

Laufzeit: Abarbeitung des vorberechneten Umschalt-Schedules



(nach: Gerhard Fohler)

Konstruktion eines statischen Schedules nach Abhängigkeiten (Idee)

Step 1:



Step 2:

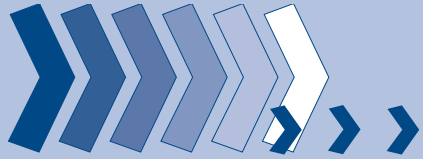


Step 3:



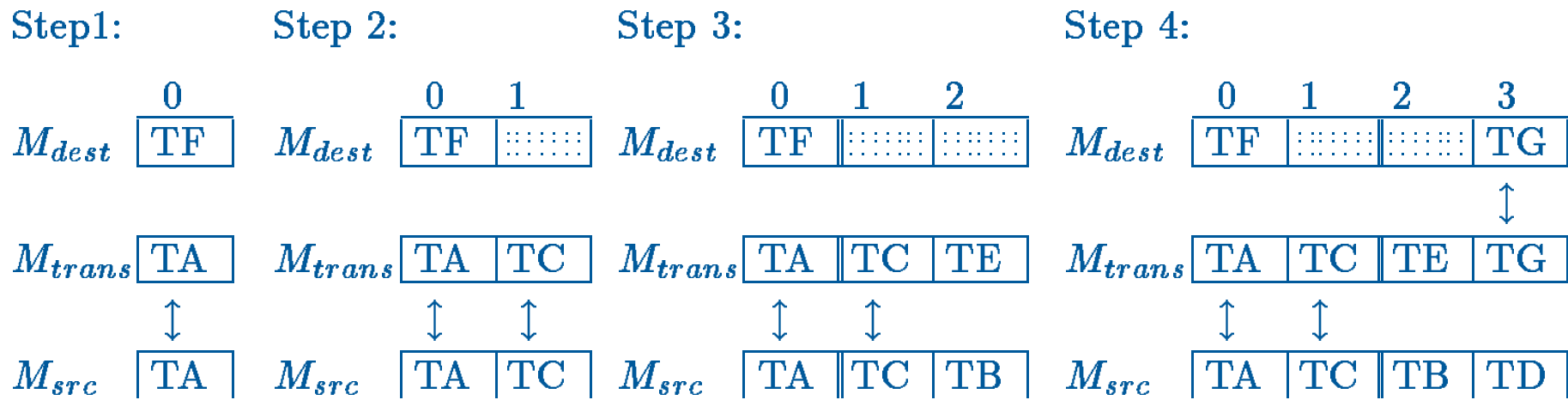
Step 4:

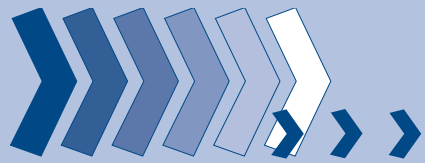




(nach: Gerhard Fohler)

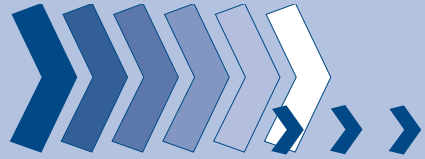
Konstruktion eines Umschalt-Schedules (Idee)





## *Echtzeit-Anwendungen: Betrachtete Beispiele*

- GAP — Generic Avionics Platform: Systementwurf für ein Echtzeitsystem zur Steuerung eines Militärflugzeuges
- Steuer- und Diagnosesysteme in Fahrzeugen



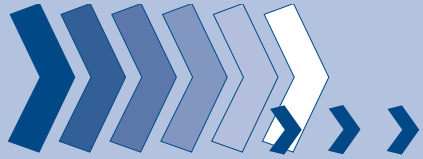
## *Beispiel: GAP — Generic Avionics Platform*

aus: C. Douglas Locke, David R. Vogel und Thomas J. Mesler: Building a Predictable Avionics Platform in Ada: A Case Study

IEEE RTSS 1991

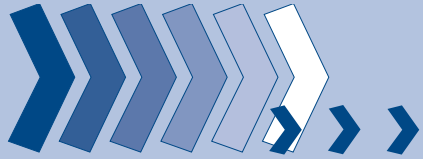
Ziele:

- Anwendung von RMS auf ein existierendes RT-Problem
- Benutzung moderner Software-Engineering-Methoden
  - Information Hiding
  - Separation of Concerns
- Benutzung von ADA-Tasking
- Erzielen einer hohen Auslastung

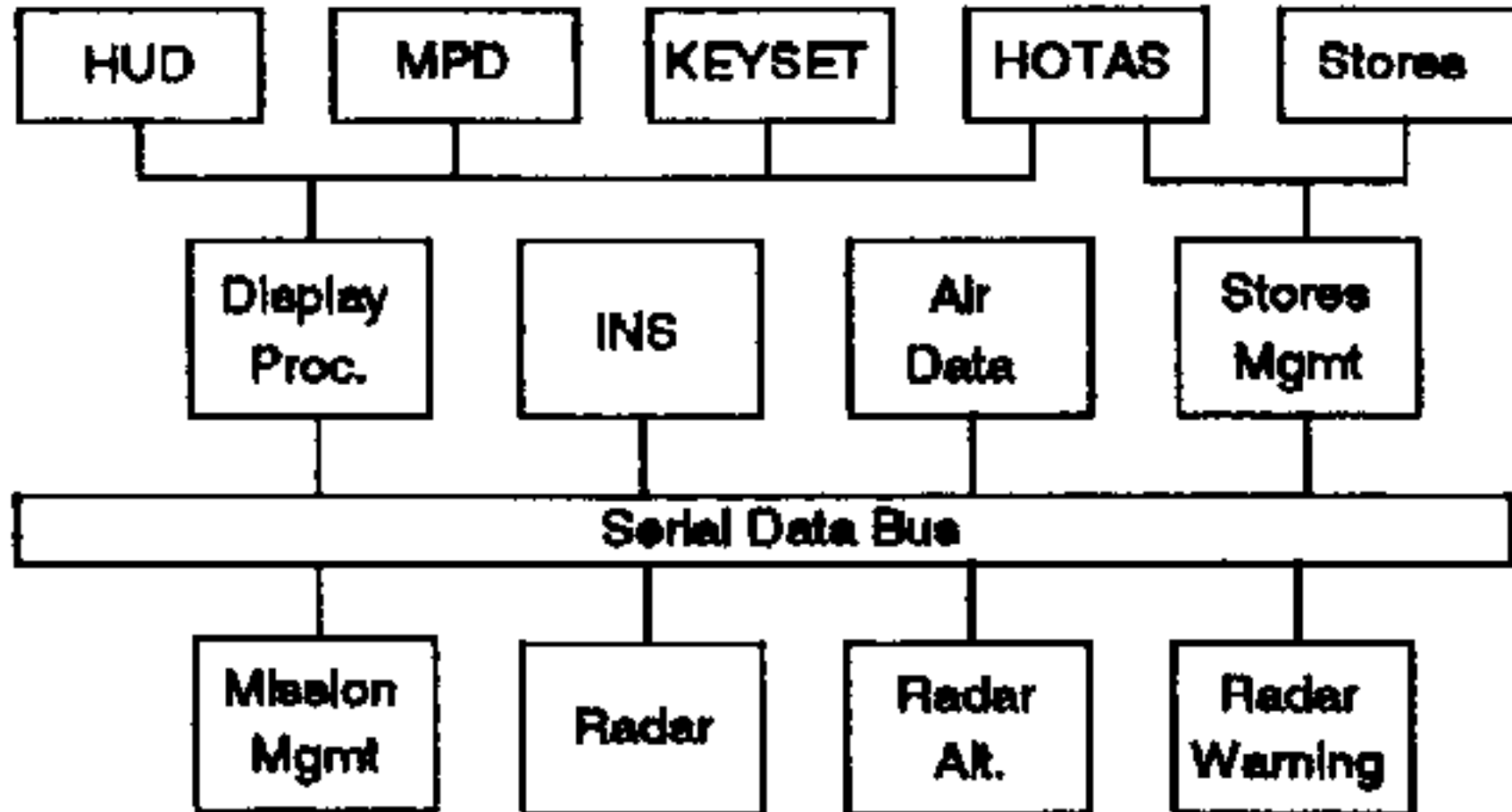


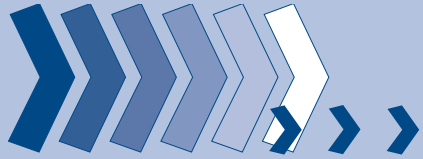
## GAP — *Generic Avionics Platform*

- GAP ist ein Modell eines Flugzeug-Missions-Computer-Systems
- Identisch zu existierenden US-Navy Flugzeugen nach der *Generic Avionics Software Specification*
- Modelliert das Missionsprogramm in
  - Timing
  - Komplexität
  - Datenabhängigkeiten
  - Funktionalität, soweit erforderlich für
    - \* Einhaltung von zeitlichen Anforderungen
    - \* robustes Softwaredesign

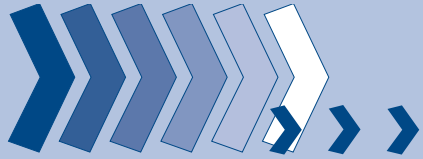


# GAP — Block Diagram





- Navigation  
berechnet Position, Flughöhe und Geschwindigkeiten
- Radar Kontrolle  
liefert Zielpositionen
- Radar Warnsystem
- Waffen-Steuerung
- Display  
aktualisiert die Informationen für den Piloten
- Tracking  
aktualisiert Informationen über das Ziel
- Test
- Datenbus  
ermöglicht Kommunikation zwischen dem Mission Control Computer (MCC) und externen Systemen

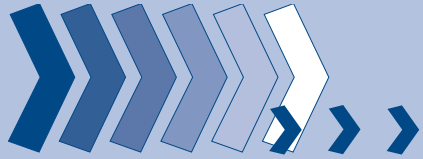


## *GAP — Zeitanforderungen I*

- Navigation  
20 hz Frequenz auf Basis der erforderlichen Genauigkeit (bei Mach 2 30 m Flugstrecke in 50 ms)
- Display  
100 ms auf Basis der menschlichen Wahrnehmung, 65 ms, wenn kontinuierlich erscheinen soll
- Ballistische Berechnungen  
5 ms für Genauigkeit auf Basis von Geschwindigkeiten der verschiedenen Flugkörper
- Sensorsteuerung  
angepaßt an Hardware: 10 Hz für Radar, mehr als 1 KHz für elektromagnetische Beobachtung

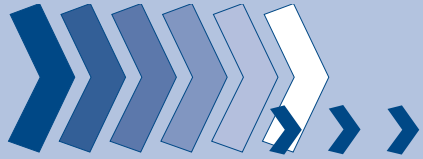


<b>Table 1. GAP Timing Requirements</b>				
<i>System</i>	<i>Subsystem</i>	<i>Periodicity or Response Time (ms.)</i>	<i>F</i>	<i>U</i>
Display	Status Update	200	3	1.50
	Keypad	200	1	0.50
	Hook Update	80	2	2.50
	Graphic Display	80	9	11.25
	Stores Update	200	1	0.50
RWR	Contact Mgmt.	25	5	20.00
Radar	Target Update	50	5	10.00
	Tracking Filter	25	2	8.00
NAV	Nav Update	59	8	13.56
	Steering Cmds	200	3	1.50
	Nav Status	1000	1	0.10
Tracking	Target Update	100	5	5.00
Weapon	Weapon Protocol	A 200	1	0.50
	Weapon Release	A 200*	3	1.50
	Weapon Aim	A 50	3	6.00
BIT	Equ. Status Update	1000	1	0.10
Data Bus	Poll Bus Devices	40	1	2.50



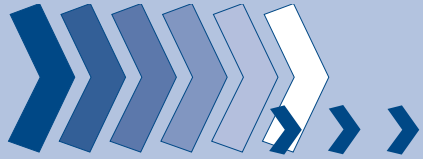
# GAP — Datenabhängigkeiten I

Table 2. GAP Data Flow					
System	Subsystem	Input	Output		
Display	Status Update	all	DB		
	Keyset	DB	all		
	Hook Update	DB			
	Graphic Display	all	DB		
	Stores Update	W	DB		
RWR	Contact Mgmt.	DB,N,K, W	D,DB,T		
Radar	Target Update	DB,N,K	D,T,DB		
	Tracking Filter	DB,N	D,T,DB		
NAV	Nav Update	DB,K,R	R,T,DB, W,D,RW		
	Steering Cmds	D	D		
	Nav Status	DB	D		
Tracking	Target Update	N,DB,K, R,RW	D,W		
Weapon	Weapon Protocol	K	DB		
	Weapon Aim	N,T	DB,D		
	Weapon Release	n/a	DB		
BIT	Equ. Status Update	n/a	D		
Data Bus	Poll Bus Devices	all	n/a		
<b>LEGEND</b>					
B	- BIT Status	D	- Display	DB	- Data Bus
K	= Keyset	N	= Nav	R	= Radar
RW	= RWR	T	= Tracking	W	= Weapon

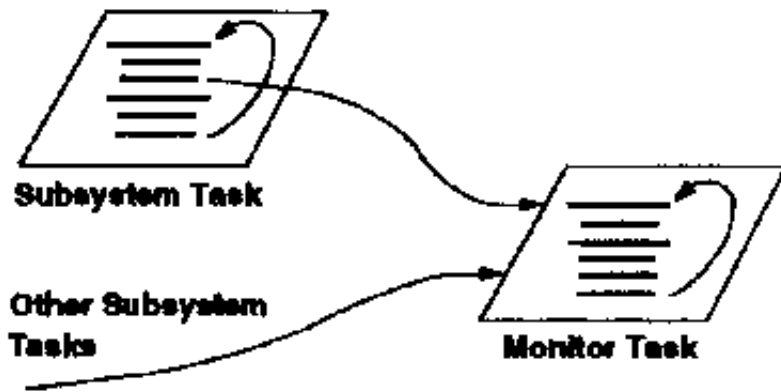


## *GAP — Datenabhängigkeiten II*

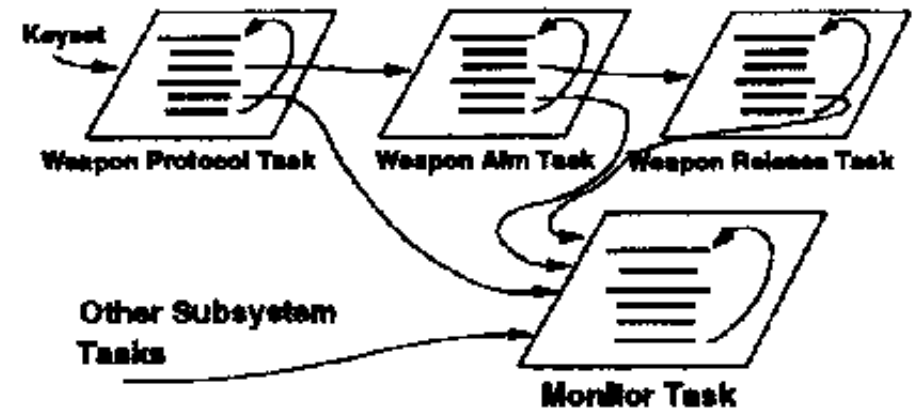
- Keine gemeinsamen Daten zwischen Subsystemen (“separation of concerns”)
- Benutzung eines separaten Daten-Managers pro Subsystem
- Daten-Manager speichert Daten des Subsystems
- Daten-Manager behandelt Transfers zu anderen Subsystemen
- Benutzung von Priority Ceiling, um Prioritäts-Invertierungen zu verhindern
- Ausnahme: Waffen-System als aperiodischer Vorgang: Tasks der Waffensteuerung sind nur aktiv, wenn Waffensteuerung aktiviert wurde



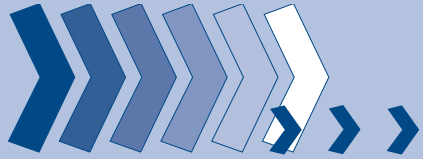
# GAP — Datenabhängigkeiten III



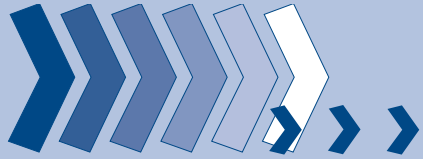
Typical GAP System



Weapon System

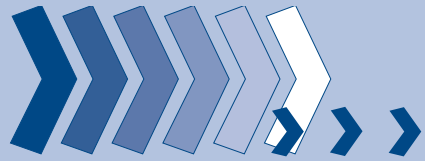


<b>Table 3. GAP Task Set Characteristics</b>				
<i>Task</i>	<i>Priority (PCP)</i>	<i>Period (ms)</i>	<i>Exec. Time (ms)</i>	<i>Util. (%)</i>
Timer_Interrupt	101	1.00	0.051	5.10
Weapon_Release	98	200.00	3.000	1.05
Radar_Tracking_Filter	84	25.00	2.000	8.00
RWR_Contact_Mgmt	72	25.00	5.000	20.00
Data_Bus_Poll_Device	68	40.00	1.000	2.50
Weapon_Aiming	64	50.00	3.000	6.00
Radar_Target_Update	60	50.00	5.000	10.00
Nav_Update	56	59.00	8.000	13.56
Display_Graphic	40	80.00	9.000	11.25
Display_Hook_Update	36	80.00	2.000	2.50
Tracking_Target_Upd	32	100.00	5.000	5.00
Weapon_Protocol	28	A	1.000	
Nav_Steering_Cmds	24	200.00	3.000	1.50
Display_Stores_Update	20	200.00	1.000	0.50
Display_Keyset	16	200.00	1.000	0.50
Display_Stat_Update	12	200.00	3.000	1.50
BET_E_Status_Update	8	1000.00	1.000	0.10
Nav_Status	4	1000.00	1.000	0.10
<i>Total Utilization (w/o Timer_Interrupt task)</i>				<i>84.06</i>



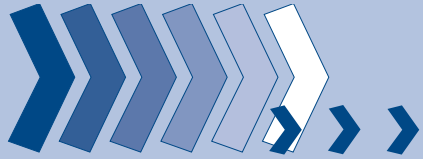
### RMA:

- Die acht höchstpriorären Tasks halten immer ihre Deadlines ein
- Zwei mittelprioräre Tasks verpassen ihre Deadlines
  - dennoch: niederprioräre Tasks halten Deadlines ein!
  - Ursache: kurzzeitige Überlastsituationen durch aperiodische Tasks
  - Abhilfe: Diese Situation tritt nur theoretisch auf durch sich aufgrund der Anwendungslogik ausschließende Tasks (Weapon\_Aim und Weapon\_Release)



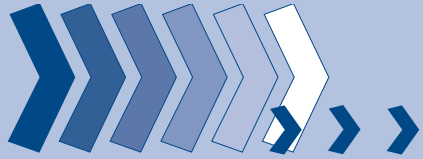
## *Beispiel: Steuer- und Diagnosesysteme in Fahrzeugen*

- Elektronik in Fahrzeugen hat sich in den letzten Jahren immer weiter entwickelt durch
  - Neue Anforderungen: ABS, Airbags, ESP, ...
  - Integration verschiedener Systeme
  - Kostenersparnis trotz neuer Funktionen
  - Kostenersparnis durch Übergang Mechanik zu Elektronik
- Vorteil: Komplexe elektronische Baugruppen als Blackbox, “einfacher” Service
- Nachteil: Service-Personal ist ungeachtet erweiterter Möglichkeiten “überfordert”

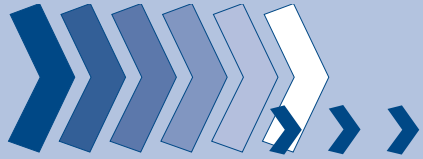


## *Evolution eines KFZ-Systems*

- Elektrische Verkabelung ohne jede Elektronik, Steuerungen elektromechanisch
- Vereinzelt Benutzung elektronischer Baugruppen (Zündung, Radio)
- Komplexere Einzelsysteme
- Zusammenfassung von Einzelsystemen (Zündung + Einspritzung + Abgasreglung = Motorsteuergerät)
- Aufbau von Punkt-zu-Punkt-Verbindungen zwischen einzelnen Systemen (Beispiel: Motorsteuerung zu Instrumenten)
- Diagnosemöglichkeiten für komplexere Einzelsysteme
- Diagnosemöglichkeiten für alle Systeme zusammen (VAG: K-Signal)
- Datenbus zur Verbindung von Steuergeräten (VAG: CAN)
- Vereinheitlichung auf ein Bus-System (VAG: CAN)



- “Mischung” von verschiedenen kritischen Tasks in System-Komponenten oder Kommunikationssystemen erfordert
  - Saubere Trennung
  - Nichtbeeinflußung der kritischen Tasks durch unkritische
- Architekturunterstützung für “Komponierbarkeit”
  - Einfaches Hinzufügen und Entfernen von Komponenten (Komposition)
  - Vorhersagbares zeitliches Verhalten bei Kompositionen



## *Beispiel: VAG-Diagnose I*

---

038906018FB 1.9l R4 EDC G000SG 2170  
Codierung 00005 WSC 06402

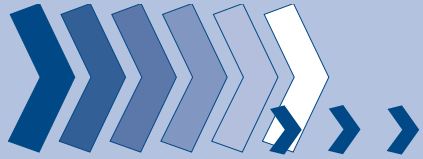
kein Fehler erkannt

---

1J0907379AF ASR 20 IE CAN 0001  
Codierung 13404 WSC 00001

kein Fehler erkannt

---



## Beispiel: VAG-Diagnose II

6K0035186C Radio GRO

1V32

Codierung 00403

WSC 06402

3 Fehler erkannt

00855 049

Verbindung zum CD-Wechsler keine Kommunikation

00856 036

Antenne, am Radio Unterbrechung

01465 049

Signalleitung vom Kombiinstrument keine Kommunikation

-----