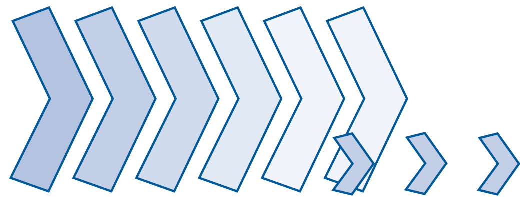
HUMBOLDT-UNIVERSITÄT ZU BERLIN

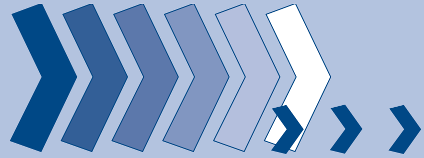


Betriebssysteme: OSEK und Pure

Dipl.-Inf. J. Richling

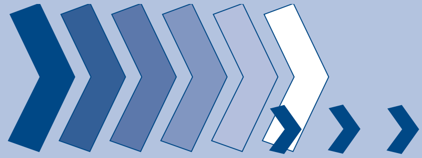
Wintersemester 2003/2004

EMES



Überblick

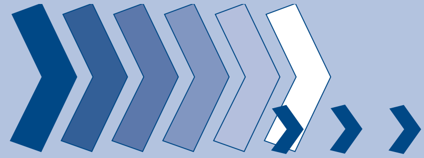
- Fünfeinhalb Vorlesungen:
  - Embedded- und RT-Betriebssysteme (19.1.04)
  - Beispiel: Windows CE (22.1.04)
  - Beispiel: Windows XP embedded (Selbststudium mit Material auf CD)
  - Beispiel: RT-Linux (26.1.04)
  - Beispiel: PalmOS (29.1.04)
  - Beispiel: OSEK und PURE (heute)



OSEK und PURE

- OSEK/VDX:
  - Industrie-Standard für eine offene Architektur verteilter Fahrzeugsysteme
  - Entwickelt von Fahrzeugherstellern und Zulieferern der Fahrzeugindustrie

- PURE
  - Forschungsprojekt der Universität Magdeburg
  - Konfigurierbares, adaptierbares und objektorientiertes Betriebssystem für “tiefst” eingebettete Systeme
  - Kann unter anderem als OSEK-konformer Betriebssystemkern konfiguriert werden



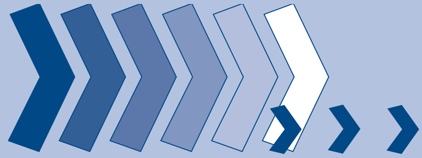
OSEK/VDX

OSEK: Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug

VDX: Vehicle Distributed eXecutive

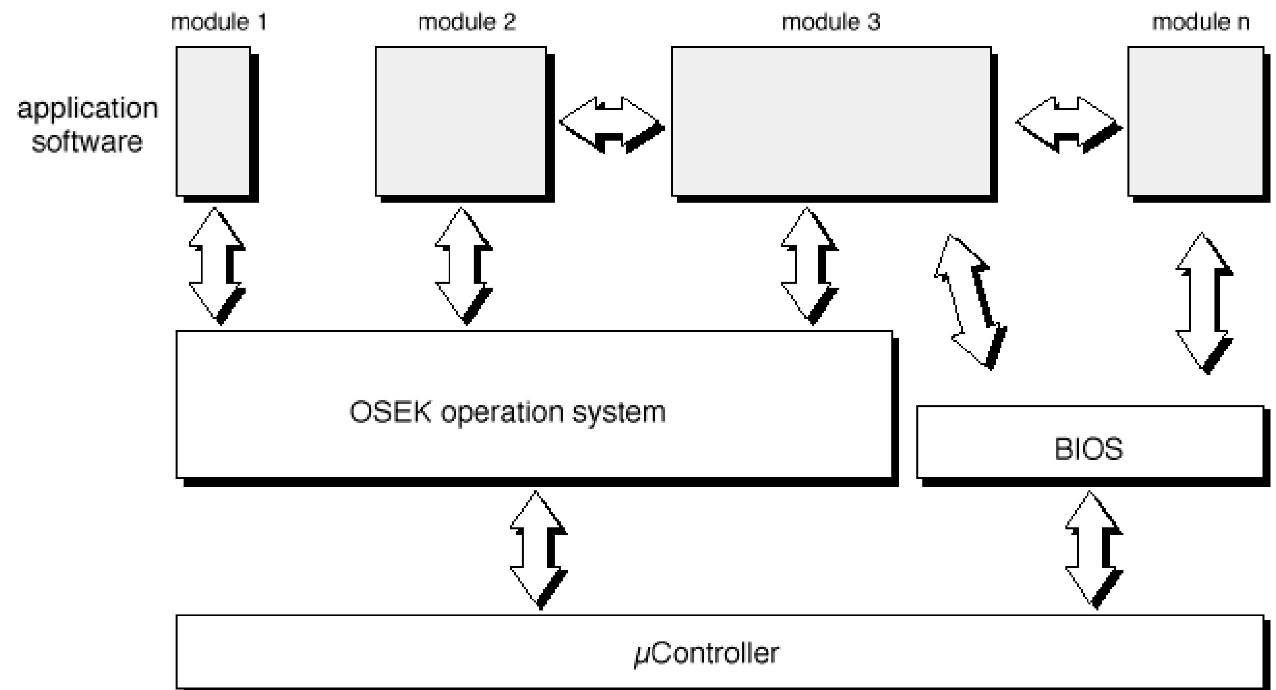
- Gemeinsames Projekt der Fahrzeugindustrie
- Ziel: Industriestandard für offene Architektur für verteilte Steuersysteme in Fahrzeugen, um Probleme der Interoperabilität/Kompatibilität zu lösen
- Beinhaltet:
  - Echtzeit-Betriebssystem
  - Software-Interfaces
  - Kommunikation
  - Netzwerk-Management

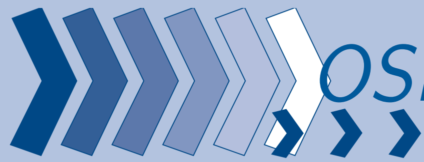
Eigenschaften mobiler und eingebetteter Systeme © J.Richling



OSEK — System Philosophie

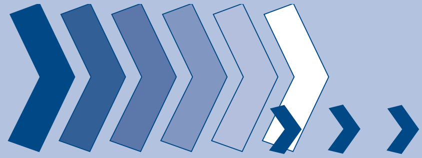
- Standardisierte Schnittstellen
- Skalierbarkeit
- Skalierbare Fehlerbehandlung
- Portierbare Anwendungssoftware
- Unterstützung für portable Systeme
- Unterstützung spezieller Anforderungen für Fahrzeuge



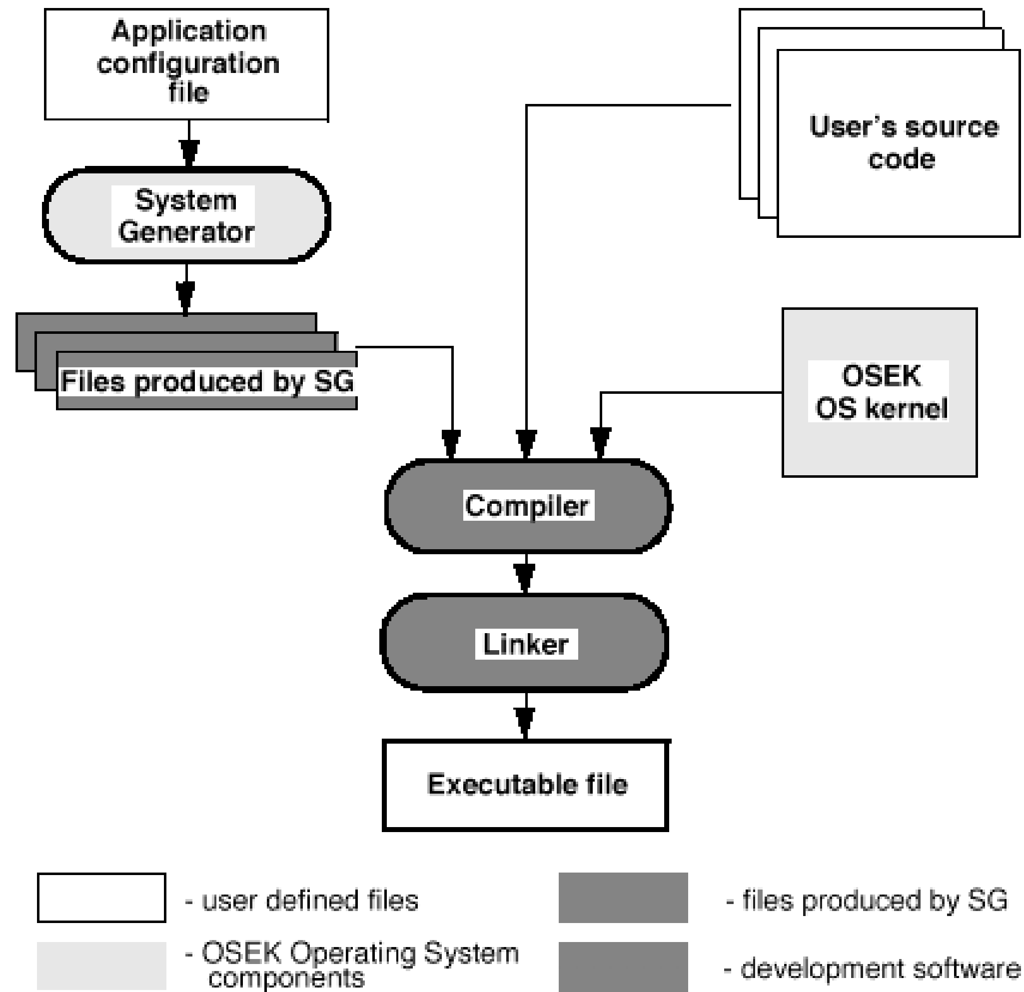


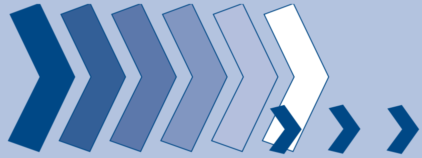
OSEK — Spezielle Anforderungen für Fahrzeuge

- OS ist statisch konfiguriert und skaliert.
- Ressourcennutzung (Anzahl Tasks, Ressourcen, Dienste) ist statisch vom Nutzer konfiguriert
- Ausführbarkeit aus dem ROM
- Portierbarkeit von Anwendungstasks
- Vorhersagbares und dokumentiertes Verhalten des OS
- Für jede OS-Implementierung müssen Performance-Parameter bekannt sein



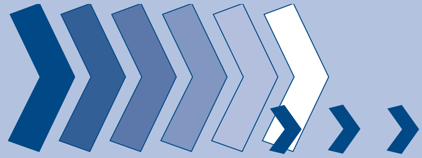
OSEK — Entwicklung von Applikationen



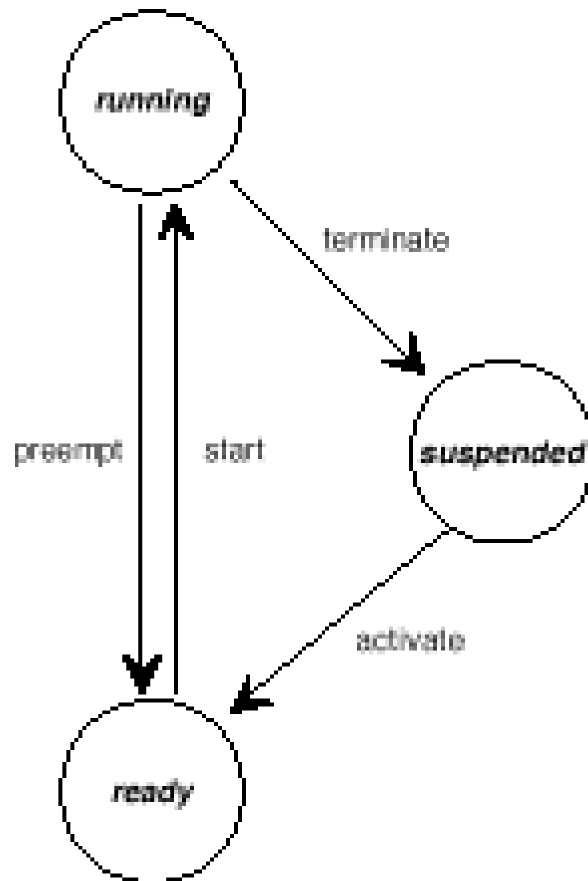


OSEK — Taskmodell

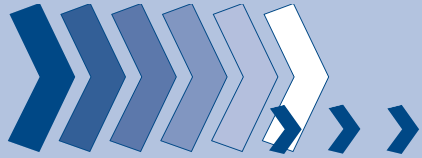
- Das OSEK-Modell unterscheidet verschiedene Konformitätsklassen, die über das unterstützte Taskmodell unterschieden werden
- Taskmodelle:
  - Basic Tasks
    Haben keinen "waiting" Zustand
  - Extended Tasks
    Können auf Events warten
- Hintergrund
  - Adaptierbarkeit in beiden Richtungen
    - Sehr einfache Systeme, die weder Synchronisation, noch kompliziertes I/O benötigen
    - Große und komplexe Systeme mit umfangreichen Tasksets, gemeinsamen Ressourcen-Zugriffen und Kommunikation zwischen den Tasks



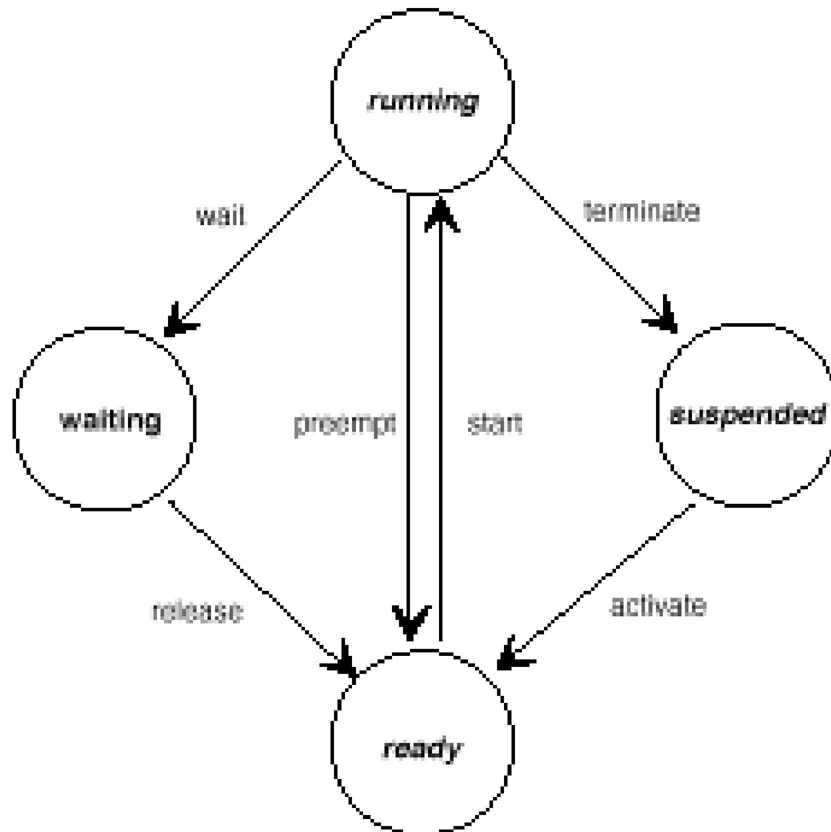
OSEK — Basic Tasks



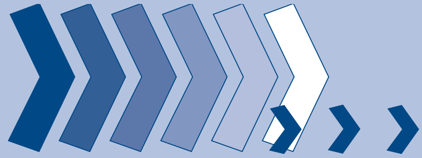
- running
  - CPU ist der Task zugewiesen
  - Instruktionen werden ausgeführt
  - Nur eine Task in diesem Zustand
- ready
  - Alle funktionalen Voraussetzungen für “ready” sind erfüllt
  - Andere Task ist der CPU zugewiesen
  - Warten auf Scheduler-Entscheidung
- suspended
  - Task ist passiv
  - Kann aktiviert werden



OSEK — Extended Tasks

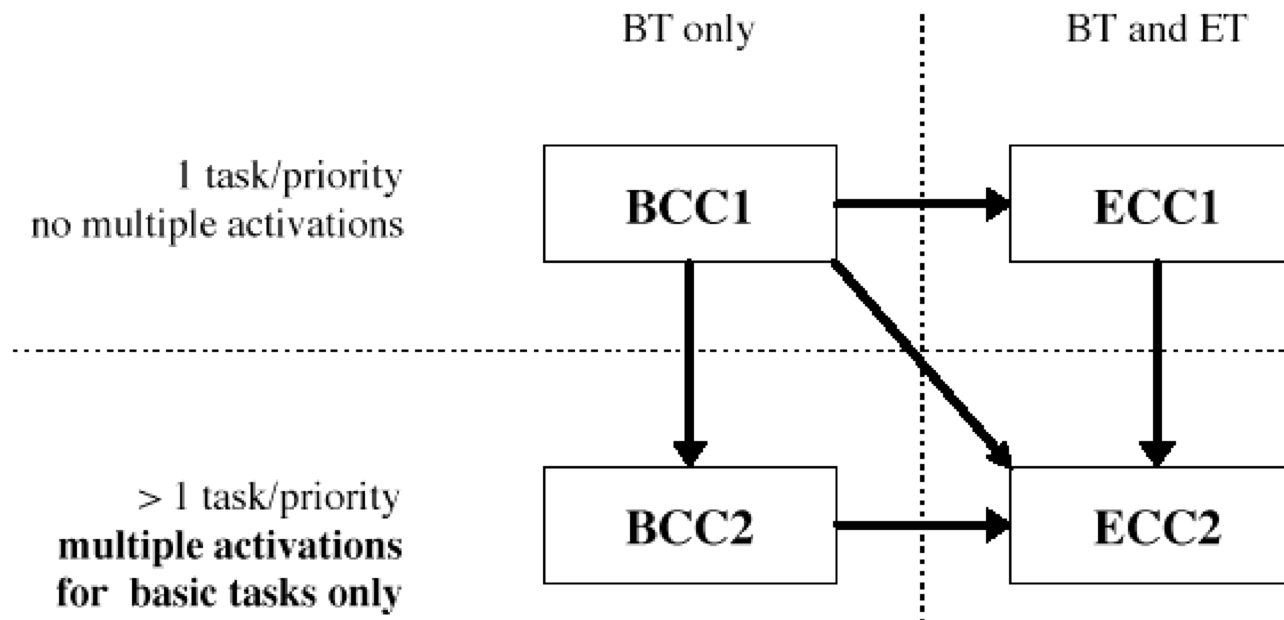


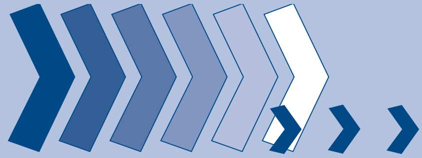
- running
  - CPU ist der Task zugewiesen
  - Instruktionen werden ausgeführt
  - Nur eine Task in diesem Zustand
- ready
  - Alle funktionalen Voraussetzungen für "running" sind erfüllt
  - Andere Task ist der CPU zugewiesen
  - Warten auf Scheduler-Entscheidung
- waiting
  - Task wartet auf wenigstens ein Ereignis
- suspended
  - Task ist passiv
  - Kann aktiviert werden



OSEK — Konformitätsklassen I

- Zwei Gruppen von Konformitätsklassen (CC — Conformance Classes):
  - Basic (BCC) erlaubt nur Basic Tasks
  - Extended (ECC) erlaubt auch Extended Tasks
- CCs sind aufwärtskompatibel:
  - BCC1, BCC2, ECC1, ECC2

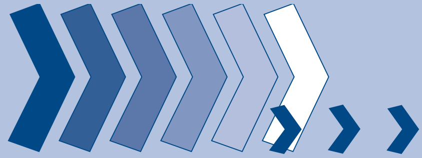




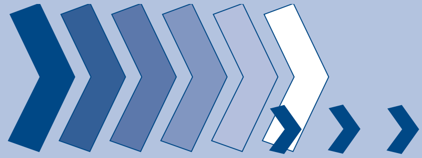
OSEK — Konformitätsklassen II

Minimale Parameter der Implementation

	BCC1	BCC2	ECC1	ECC2
Multiple requesting of task activation	no	yes	BT[9]: no ET: no	BT: yes ET: no
Number of tasks which are not in the *suspended* state	≥ 8		≥ 16 (any combination of BT/ET)	
Number of tasks per priority	1	> 1	1 (both BT/ET)	> 1 (both BT/ET)
Number of events per task	—		≥ 8	
Number of priority classes	≥ 8			
Resources	only scheduler	≥ 8 (including scheduler)		
Alarm	≥ 1 (single or cyclic alarm)			
Application Mode	≥ 1			

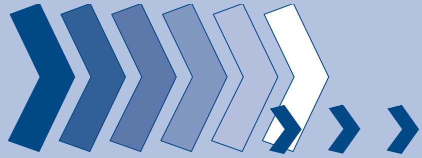


OSEK — Scheduling



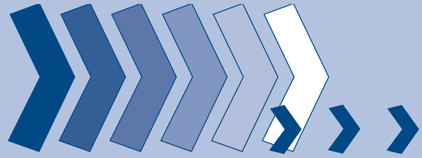
OSEK — Event Mechanismus

- Mittel der Synchronisation
- Nur für extended Tasks
  - festgelegte Anzahl von Events pro Task
- Bewirkt Zustandsänderungen zum und vom “waiting” Zustand
- Systemrufe zum: Erzeugen, Warten, Rücksetzen und Definieren von/auf Events
- Alarme als Form von Events



OSEK — Ressourcenverwaltung

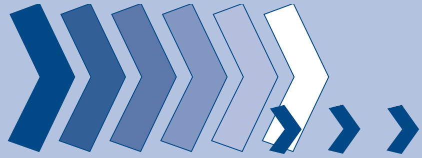
- Obligatorisch für alle Konformitätsklassen
- Verantwortlich für prioritätengesteuerte Zugriffsverwaltung auf Ressourcen:
  - Management Entities (z.B. Scheduler)
  - Programmstücke (kritische Sektionen)
  - Speicher
  - Hardware
- Gewährleistet:
  - Gegenseitigen Ausschluß
  - Verhinderung von Prioritäteninvertierung (durch Priority Ceiling)
  - Deadlock-Vermeidung
  - Zugriff auf Ressourcen führt nicht zum Zustand "waiting"



OSEK — Fehlerbehandlung I

- Fehlerbehandlung durch Hook-Routinen (nutzerdefinierte Aktionen im internen Ablauf des OS)

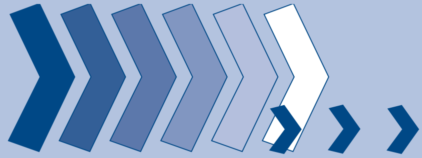
- Hook-Routinen:
  - Vom OS aufgerufen in einem speziellen Kontext
  - Höhere Priorität als alle Tasks
  - Implementationsabhängiges Interface
  - Teil des OS, aber vom Nutzer definiert und implementiert
  - Interface von OSEK standardisiert
  - Funktionalität nicht von OSEK standardisiert (und meist umgebungsabhängig und nicht portabel)
  - Dürfen nur eine Teilmenge der API-Funktionen nutzen
  - sind optional
  - Konventionen für Hooks müssen in einer OSEK-Implementierung beschrieben sein



OSEK — Fehlerbehandlung II

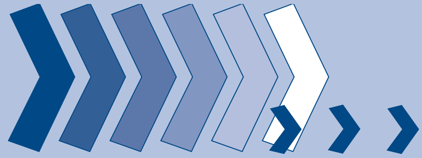
Beispiele für Hook-Routinen:

- Systemstart
  Entsprechende Hook-Routine wird nach Start des OS und vor Start des Schedulers ausgeführt
- System-Shutdown
  Herunterfahren des Systems durch Anwendung oder im Falle eines Fehlers
- Debugging
  Anwendungsabhängiges Tracing oder Hooks zur Erweiterung von Kontext-Wechseln
- Fehler-Behandlung
  Wird gerufen, wenn ein Systemruf kein E_OK zurückgegeben hat



OSEK — Kommunikation

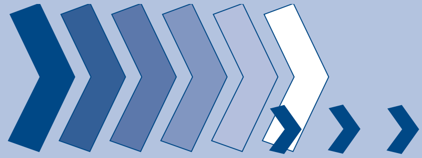
- Transparente lokale und Netzwerk-Kommunikation über Nachrichten
- Nachrichten sind in Nachrichtenobjekte eingebettet
- Nachrichtenobjekte werden zur Konfigurationszeit des Systems definiert
- Nachrichtenobjekte sind durch UIDs eindeutig identifiziert
- Tasks referenzieren Nachrichten über UIDs
- OSEK unterscheidet zwischen Event- und State-Nachrichten
- Für Event-Nachrichten: one-to-one und one-to-many
- Task-Aktivierung und Event-Erzeugung können statisch an die Ankunft einer Nachricht gekoppelt werden



PURE

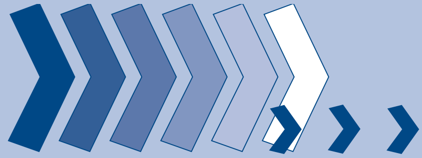
- Entwickelt an der Universität Magdeburg und GMD unter Leitung von Prof. Schröder-Preikschat
- Ziele
  - Maßgeschneidertes OS für verschiedenste Anwendungen
  - Skalierbarkeit, insbesondere nach “unten”
- Prinzipien
  - Programmfamilien mittels OO-Techniken realisieren
  - Feingranulare Konfigurationen ermöglichen
  - Keine unnötigen Features aufzwingen
- PURE ist nicht nur ein OS, sondern eine OS-Familie
  - Minimale Teilmenge von Systemfunktionen
  - Abstraktionen der Hardware

Abbildungen und Meßwerte aus Veröffentlichungen der PURE-Entwickler



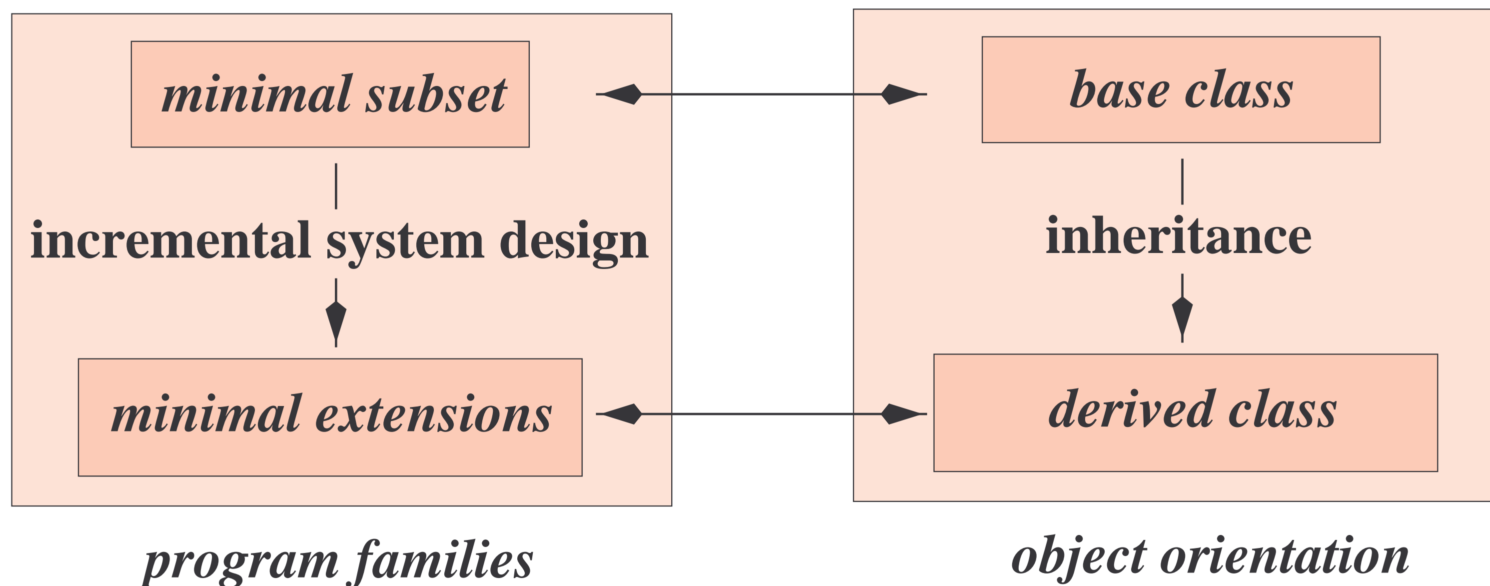
PURE — Anforderungen

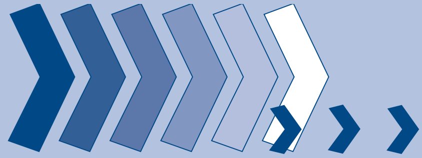
- Verschiedene Plattformen
  - Gastebenen- und native Implementationen
  - Unterschiedliche Hardware
  - Ausstattung an RAM + ROM
- Verschiedene Benutzeranforderungen
  - Programmiersprachen: Assembler, C, C++
  - Vorgefertigte Schablonen als Hilfsmittel, nicht aufzwingen
- Verschiedene Eigenschaften anbieten
  - Statische oder dynamische Konfiguration
  - Verschiedene Scheduling-Verfahren
  - Verschiedene Kommunikationsmöglichkeiten
  - ...
  - u.a. Konformität zum OSEK-Modell durch entsprechende Konfiguration



PURE — Familienkonzept

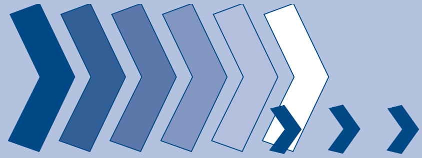
- Programmfamilien...
  - drücken polymorphe Strukturen aus
  - basieren auf Gemeinsamkeiten der Familienmitglieder
  - bieten Wiederverwendung existierender Komponenten an
- ... können am einfachsten objekt-orientiert implementiert werden





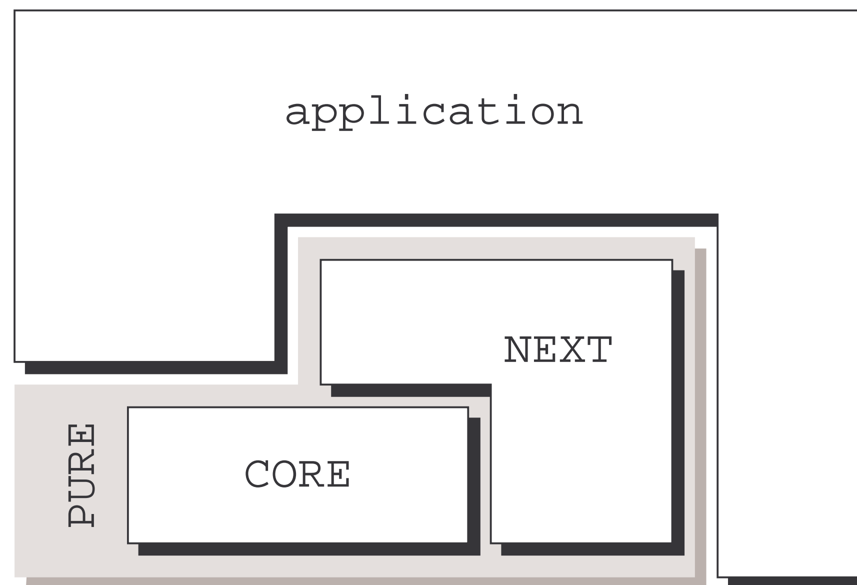
PURE — Probleme und Lösungen mit OO

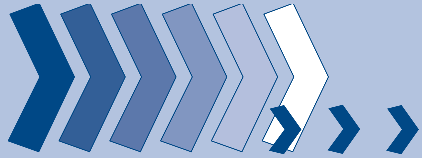
- Compilate enthalten oft eine Menge ungenutzten Codes
  - Vermeidung von Standard-Bibliotheken
  - Pro Methode ein File
- High-Level-C++-Features haben ihren Preis (Overhead)
  - Templates und virtuelle Funktionen mit Bedacht benutzen
  - Exception-Handling und Run-Time-Typinformation abschalten
- Tiefe Vererbungshierarchien führen zu vielen Konstruktor-Rufen
  - Benutzung von Inline-Funktionen, wenn möglich



PURE — Systemarchitektur I

- Offene Architektur
- Mikrokernel mit minimaler Funktionalität
- Objektorientierter Aufbau
  - Kleine Objektmodule
  - Zusammenfügbar nach Bedarf/Konfiguration

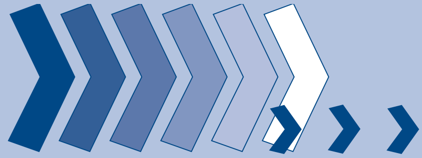




PURE — Systemarchitektur II

- CORE (concurrent runtime executive) — Kern von Pure
  - Passive und aktive Objekte
  - Minimale Systemfunktionen für Scheduling von
    * Events (implementiert über Hardware-Interrupts)
    * Aktionen (implementiert als leichtgewichtige Threads)
  - Möglichkeit der Integration von Treiber-Modulen

- NEXT (nucleus extentions)
  - Anwendungsorientiertes Prozeß- und Adressraummodell
  - Threadsynchronisation
  - Problemorientiertes (remote) Messagepassing
  - . . .
  - Nur bei Bedarf vorhanden
  - Verwandeln den Kern in einen verteilten abstrakten Thread-Prozessor



PURE — Beispiel: Threadklassenhierarchie

toc

Servant

tocServant tocPointer

Trigger Triplet

Coroutine Being

Chain Priority Actor Chain

PriChain Activity Commuter

Visitor

PriEntrant Entrant

Buckle

Locker VoidLocker

Idler Lavir Passenger

Candidate Contestant Rival Competitor Confidant Wizard

Schemer

Chain

Claimant Aspirant

Partner

Trimmer

Customer Conductor

Fellow

Monitor

Native

Bundle

Genius

Queue

Clasp

VoidLocker Locker

Sequence

TaskStore

Candidature

Party

Chain

Commuter

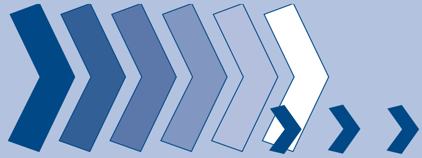
Contest Queue

Rivalry

Competition

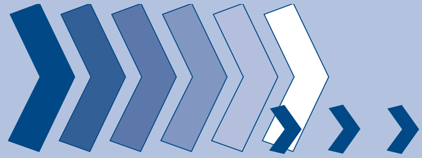
Passage

Scheme



PURE — OSEK

- PURE ist durch Konfiguration an sehr viele Anforderungen anpaßbar
  - Ziel: OSEK-Konformität als “proof of concept”
- Nötige Anpassungen/Erweiterungen:
  - Implementation von
    - OSEK-API
    - OSEK-Thread-Management
    - Alarm-Behandlung nach OSEK-Spezifikation
  - OSEK-Subfamilie: 61 Klassen
    - 19 dediziert für OSEK
    - 42 als wiederverwendbare Erweiterungen zu bereits existierenden Klassen
- Ergebnis
  - Implementation aller vier OSEK-Konformitätsklassen
  - Durch vorhandene Konfigurationsmöglichkeiten flexibel konfigurierbar



PURE — Building Block Konfiguration

pipe
- basic i/o
- formatted i/o
- floating point i/o

bell
- Relative Alarm
- Absolute Alarm

threads reel
- domains
- mixed scheduling
- preemptive scheduling
- priority based preemptive scheduling

device
- timer
- Ethernet
- cout
- CAN
- serial port
- parallel port

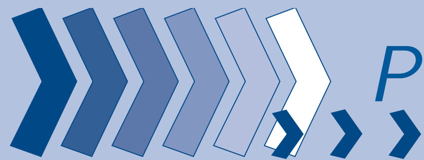
actionbox
- standalone event
- event masks
- blocking event
- process based event
- priority ceiling

supply
- address spaces
- heap

flange
- interrupt simulation
- dynamic binding
- vectortable abstraction
- predefined wrapper
- ROMable wrapper

sluice
- fine grain locking
- coarse grain locking
- synchronized interrupt handling
- unsynchronized interrupt handling

- multi-activation threads
- priority based scheduling
- cooperative scheduling
- priorities



PURE — Building Block Konfiguration: OSEK

pipe
basic i/o
formatted i/o
floating point i/o

bell
Relative Alarm
Absolute Alarm

threads reel
domains
mixed scheduling
preemptive scheduling
priority based preemptive scheduling

device
timer
Ethernet
cout
CAN
serial port
parallel port

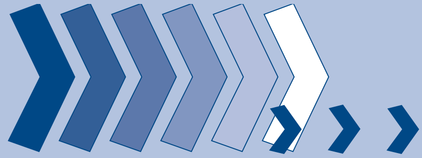
actionbox
standalone event
event masks
blocking event
process based event
priority ceiling

supply
address spaces
heap

flange
interrupt simulation
dynamic binding
vectortable abstraction
predefined wrapper
ROMable wrapper

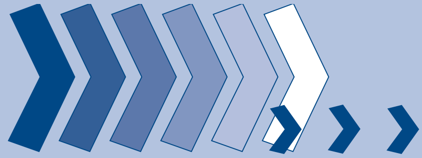
sluice
fine grain locking
coarse grain locking
synchronized interrupt handling
unsynchronized interrupt handling

multi-activation threads
priority based scheduling
cooperative scheduling
priorities



PURE — Implementationen und Meßwerte

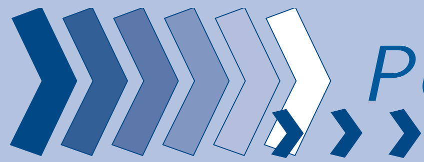
- Unterstützte Hardware (Guestlevel und native):
  - x86, i860, Alpha, Sparc, PPC60x sind implementiert
  - C167 (Controller mit CAN) in Arbeit
- Hochmodular
  - Über 100 Klassen
  - Über 600 Methoden
- Messungen und Werte
  - Basis: x86 nativ
  - egcs-1.0.2 Compiler
  - Alle Zeiten in Zyklen auf Pentium II



PURE — Codegrößen

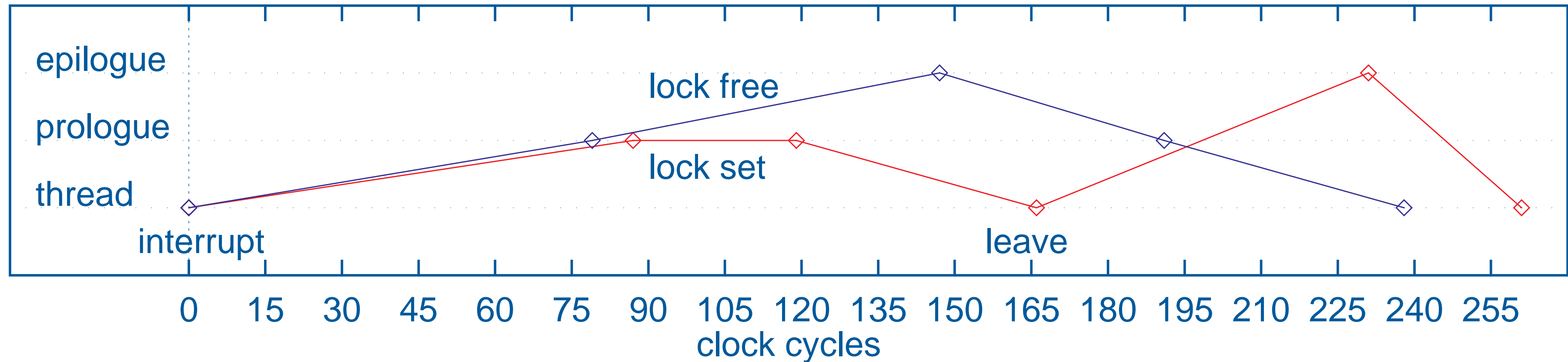
$(PURPOSE)	scenario	size (in bytes)			
		text	*data*	*bss*	total
develop	interrupt handler	6130	12	404	6546
	null	10708	4	412	11124
	null (preemptive)	12724	4	412	13140
	signaller	11141	4	412	11557
	consumer/producer	11917	4	412	12333
	epilogue signalling	14709	12	424	15145
product	interrupt handler	1910	12	404	2326
	null	1945	4	20	1969
	null (preemptive)	2053	4	20	2077
	signaller	2025	4	20	2049
	consumer/producer	4131	4	28	4163
	epilogue signalling	5256	12	424	5692

Eigenschaften mobiler und eingebetteter Systeme © J.Richling



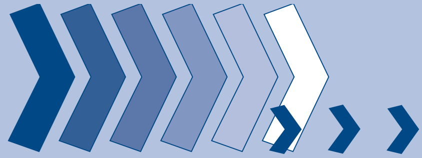
PURE — Interrupt-Latenz und Kontextwechsel

- latency of a software-triggered interrupt



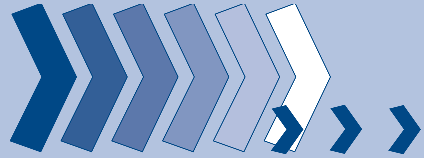
- context switches

cooperative						preemptive	
threads		same bundle		different bundle		threads	bundles
unlocked	locked	unlocked	locked	unlocked	locked		
49	57	68	76	80	94	277	300



PURE — Beispielanwendung: Aufbau

Thread 1	Thread 2
forever do begin	*forever do begin*
start time measurement	*wait for event from thread 1*
set event for thread 2	*stop time measurement*
wait for event from thread 2	*clear eventmask*
clear eventmask	*set event for thread 1*
end	*end*



PURE — Beispielanwendung: Meßwerte

scheduling strategy	code	data	thread switch time
FCFS thread	2871	1052	94
FCFS same bundle	3391	1052	126
FCFS diff. bundle	3371	1052	154
OSEK cooperative	3242	2248	148
OSEK preemptive	3674	2248	202
OSEK mixed preemptive	3922	2248	218