EMES: Eigenschaften mobiler und eingebetteter Systeme

Betriebssysteme: Pure

Dr. Felix Salfner, Dr. Siegmar Sommer

Wintersemester 2010/2011

PURE

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

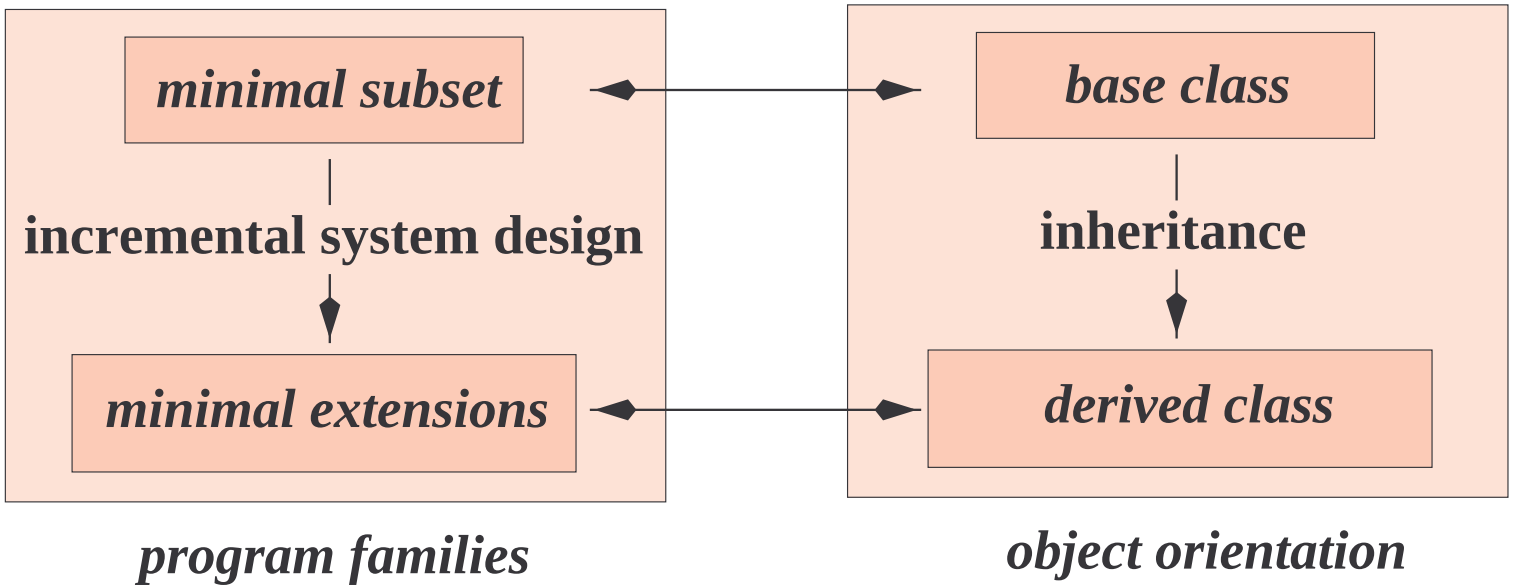
Abbildungen und Meßwerte aus Veröffentlichungen der PURE-Entwickler

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 zu OSEK 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

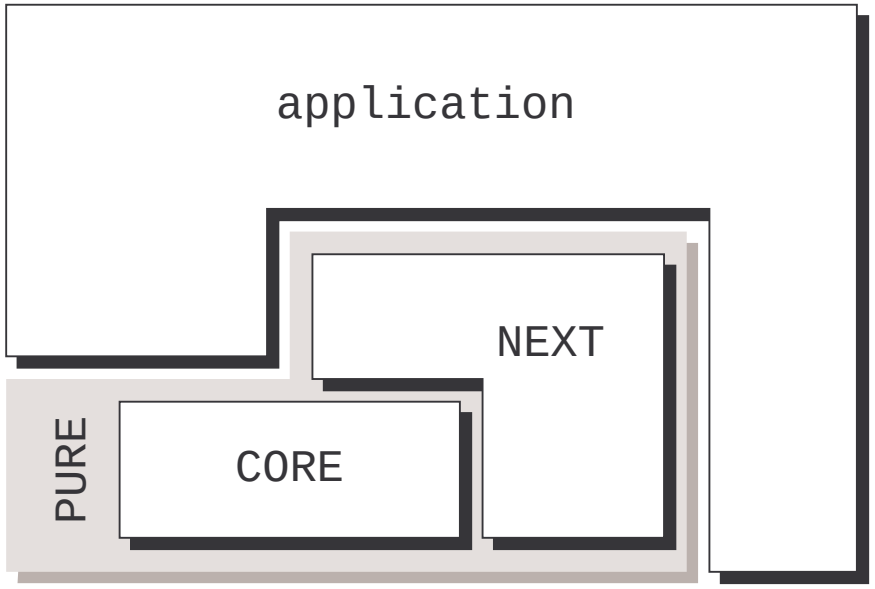


PURE — Probleme und Lösungen mit OO

- 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

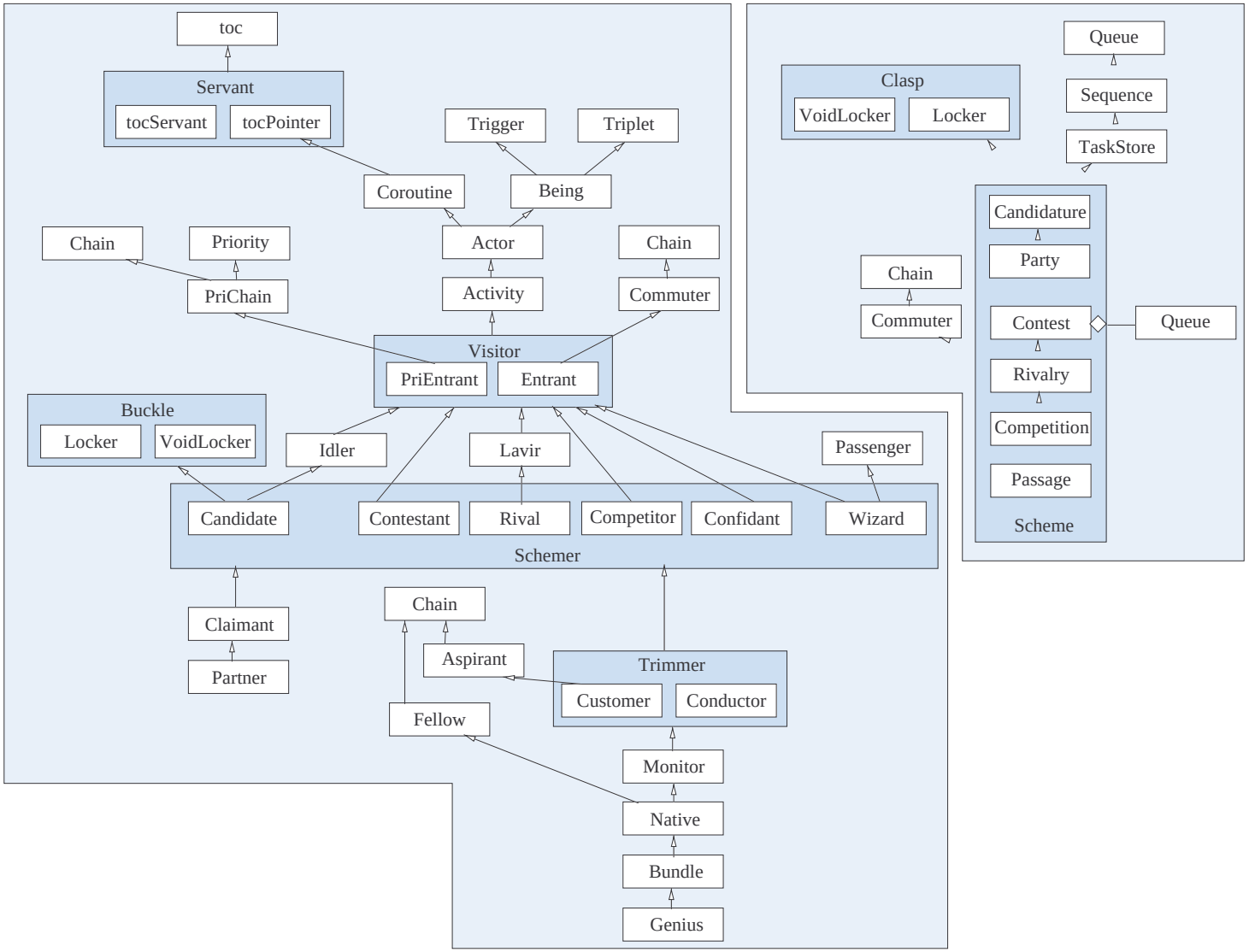
- Offene Architektur
- Mikrokernel mit minimaler Funktionalität
- Objektorientierter Aufbau
  - Kleine Objektmodule
  - Zusammenfügbar nach Bedarf/Konfiguration



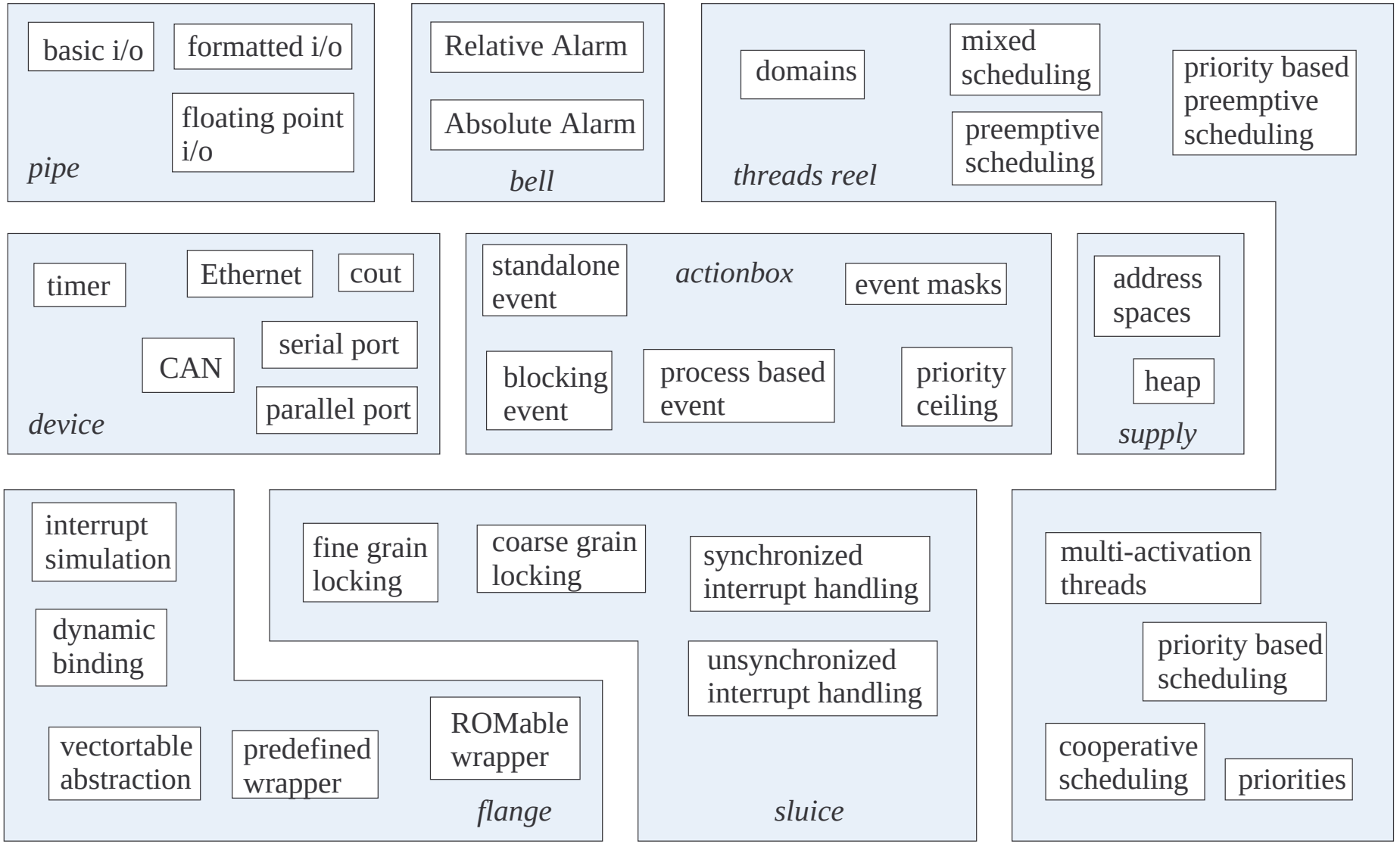
PURE — Systemarchitektur II

- 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



PURE — Building Block Konfiguration



PURE — Implementationen und Meßwerte

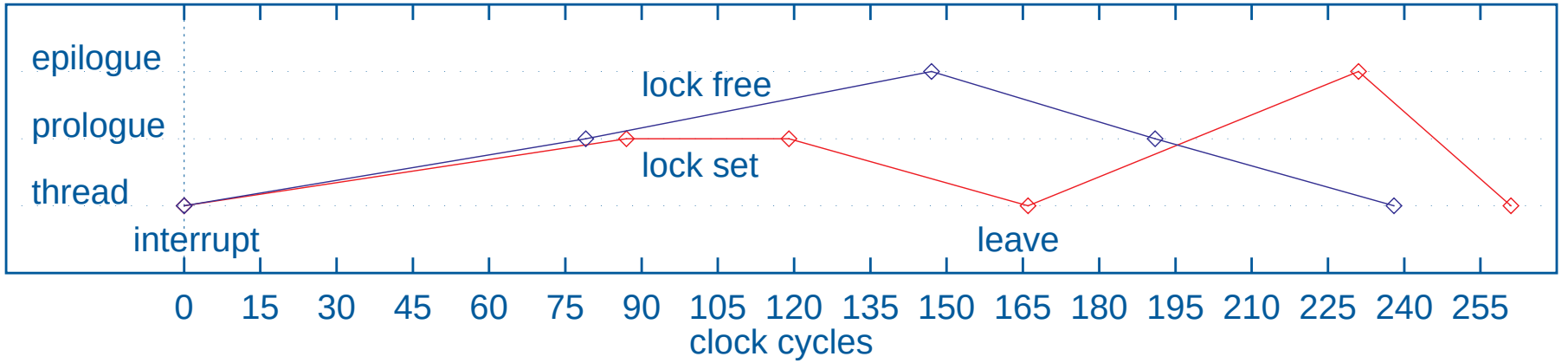
- Unterstützte Hardware (Guestlevel und native):
  - x86, i860, Alpha, Sparc, PPC60x sind implementiert
  - C167 (Controller mit CAN) in Arbeit
- Hochmodular
  - Über 100 Klassen
  - Über 600 Methoden
- Messungen und Werte
  - Basis: x86 nativ
  - egcs-1.0.2 Compiler
  - Alle Zeiten in Zyklen auf Pentium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

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

```
forever do begin
    start time measurement
    set event for thread 2
    wait for event from thread 2
    clear eventmask
end
```

Thread 2

```
forever do begin
    wait for event from thread 1
    stop time measurement
    clear eventmask
    set event for thread 1
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