Übung zu Betriebssystembau (Ü BS)

Zusammefassung und Ausblick

Daniel Lohmann

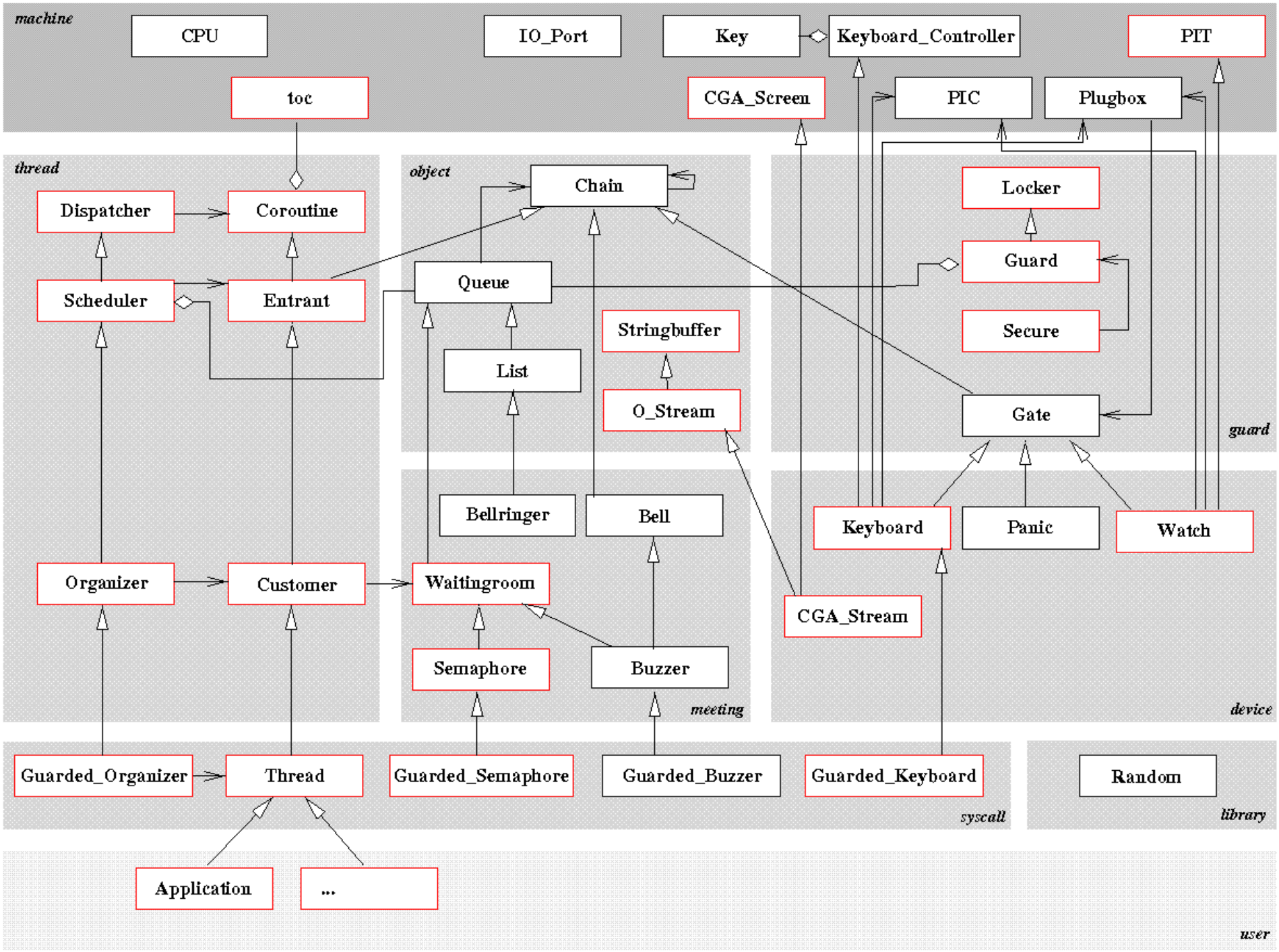
Lehrstuhl für Informatik IV

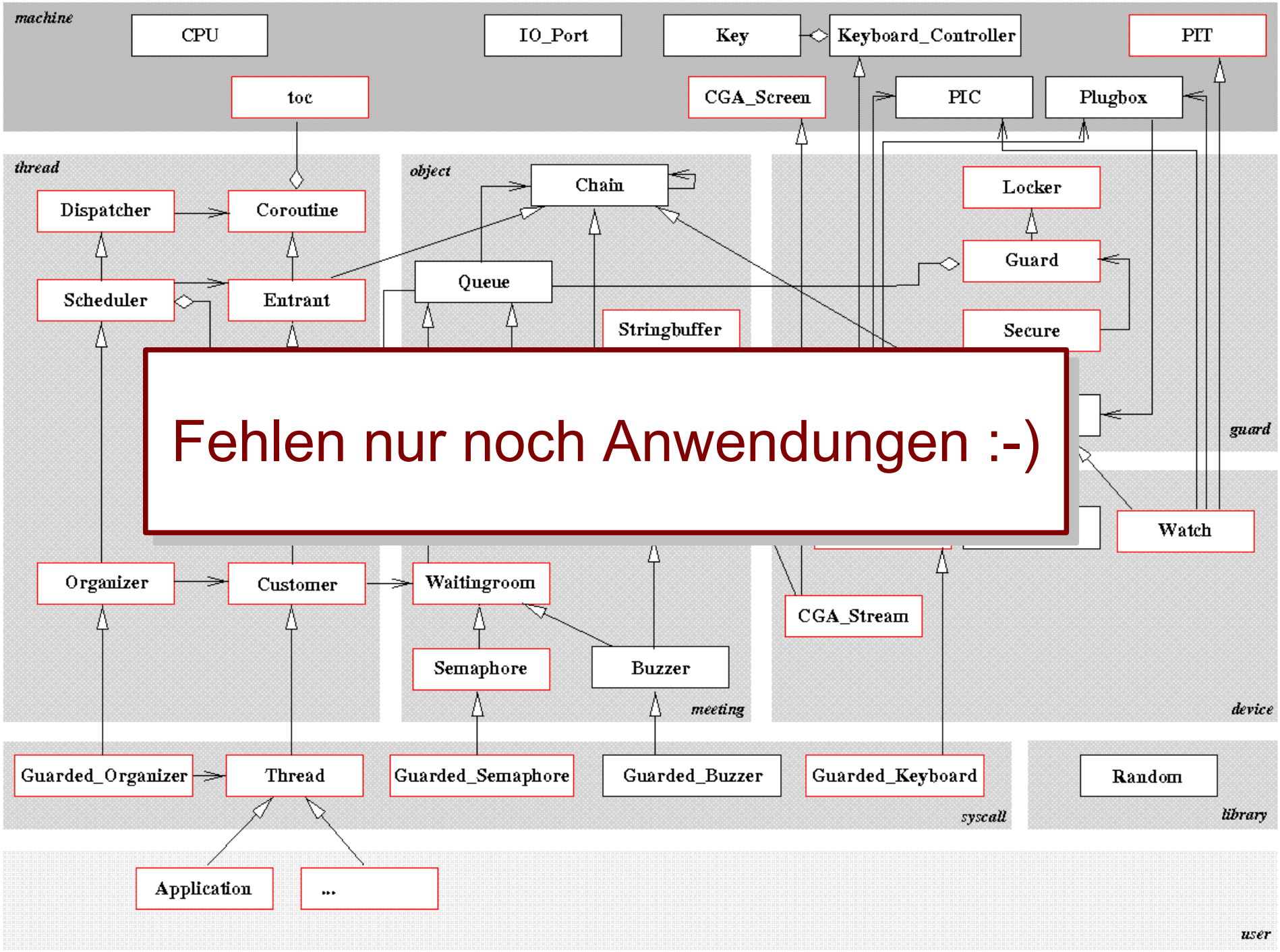
WS 05-06

OO-Stubs – Was haben wir erreicht

Ein (fast) vollständiges Betriebssystem für x86 PCs

- Geräte
  - Tastatur
  - CGA-Textausgabe
  - Timer
- IRQ-Behandlung
  - Pro-/Epilogmodell
- Threading
  - Coroutinen
  - Kooperatives Scheduling
  - Zeitgesteuertes Round-Robin Scheduling
  - Sempahoren
  - Alarme


machine
CPU
IO_Port
Key
Keyboard_Controller
PIT
toc
CGA_Screen
PIC
Plugbox
thread
Dispatcher
Coroutine
Scheduler
Entrant
object
Chain
Queue
Stringbuffer
List
O_Stream
Locker
Guard
Secure
Gate
guard
Bellringer
Bell
Organizer
Customer
Waitingroom
Semaphore
Buzzer
meeting
Keyboard
Panic
Watch
CGA_Stream
device
Guarded_Organizer
Thread
Guarded_Semaphore
Guarded_Buzzer
Guarded_Keyboard
syscall
Random
library
Application
...
user


machine
CPU
IO_Port
Key
Keyboard_Controller
PIT
toc
CGA_Screen
PIC
Plugbox
thread
object
Chain
Locker
Dispatcher
Coroutine
Guard
Queue
Scheduler
Entrant
Stringbuffer
Secure
guard
Fehlen nur noch Anwendungen :-)
Watch
Organizer
Customer
Waitingroom
CGA_Stream
Semaphore
Buzzer
meeting
device
Guarded_Organizer
Thread
Guarded_Semaphore
Guarded_Buzzer
Guarded_Keyboard
Random
syscall
library
Application
...
user

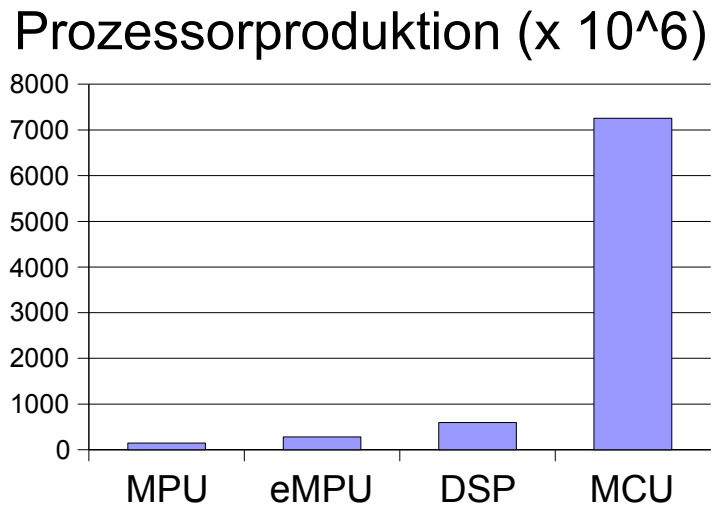
OO-Stubs – Aufgabe 7

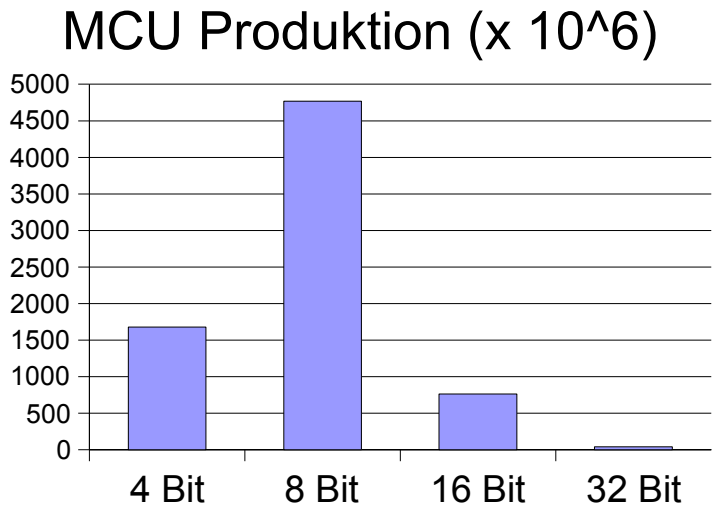
Eine Anwendung für OO-Stubs

- Bearbeitung ist **freiwillig**
  - Für den Schein reichen die Aufgaben 1—6
  - Macht aber Spaß :-)
- Vorschlag: Irgendwas mit Threading
  - Mehrere Threads, die sich mit Semaphoren synchronisieren
  - Die meisten programmieren ein Spiel
- Ein paar Beispiele aus den letzten Jahren
  - *Space Inviders* - Stefan Gabriel 2004
  - *Finest Undaground Street Fighting* - Martin Wahl 2005
  - *Stupidgame* - Michael Meier 2004
  - *Pacmännchen* - Olaf um 1800

Wo geht all das Silizium hin?

- eine Statistik aus dem Jahr 2000:





Quelle: David Tennenhouse, 2000

- von den etwa 8 Mrd. Prozessoren werden **mehr als 98%** im Bereich **eingebetteter Systeme** verwendet
- noch heute dominiert **8 Bit Technik**

AVR ATmega – Eine typische Hardware-Produktlinie

µ-Controllerfamilie, basierend auf 8 Bit RISC-Core

- 16 general-purpose Register
- Havard-Architektur, getrennter Programm und Datenspeicher
- On-Board IO-Pins, Timer, AD-Wandler, Bussysteme, ...

Ein kleiner Ausschnitt aus den angebotenen Varianten:

Bezeichnung	Prog.	RAM	IO	Timer 8/16	UART	I²C	AD	Preis
ATTINY11	1024		6	1/-	-	-	-	€0,31
ATTINY13	1024	64	6	1/-	-	-	4*10	€0,66
AT90S2323	2048	128	3	1/-	-	-	-	€1,72
ATMEGA8515	8192	512	35	1/1	1	-	-	€2,04
ATMEGA8535	8192	512	32	2/1	1	1	-	€2,67
ATMEGA169	16384	1024	54	2/1	1	1	8*10	€4,03
ATMEGA64	65536	4096	53	2/2	2	1	8*10	€5,60
ATMEGA128	131072	4096	53	2/2	2	1	8*10	€7,91

[Quelle: Digi-Key Produktkatalog, Sommer 2005]

AVR ATmega – Eine typische Hardware-Produktlinie

µ-Controllerfamilie, basierend auf 8 Bit RISC-Core

- 16 general-purpose Register
- Havard-Architektur, getrennter Programm und Datenspeicher
- On-Board IO-Pins, Timer, AD-Wandler, Bussysteme, ...

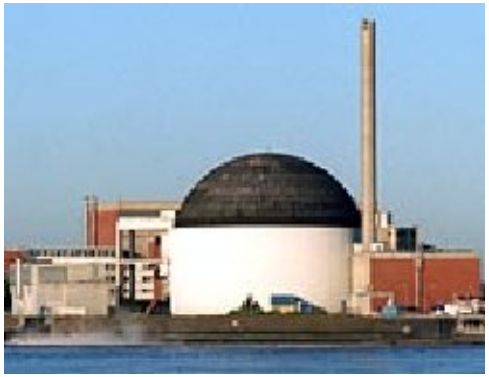
Ein kleiner Ausschnitt aus den angebotenen Varianten:

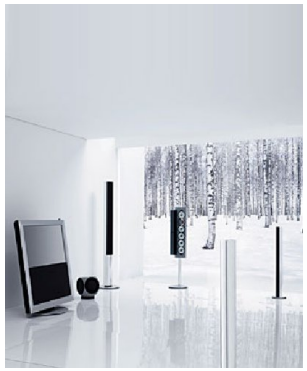
Bezeichnung	Prog.	RAM	IO	Timer 8/16	UART	I²C	AD	Preis
ATTINY11	1024		6	1/-	-	-	-	€0,31
ATTINY13	1024	64	6	1/-	-	-	4*10	€0,66
AT90S2323								€1,72
ATMEGA8515								€2,04
ATMEGA8535								€2,67
ATMEGA169								€4,03
ATMEGA64								€5,60
ATMEGA128	131072	4096	53	2/2	2	1	8*10	€7,91

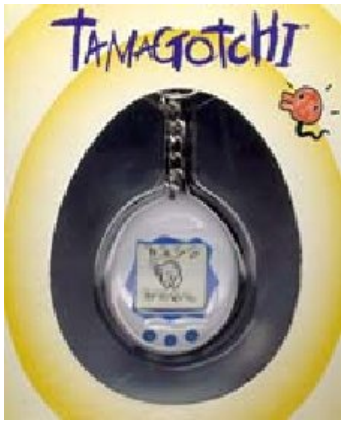
Systemsoftware muss ähnlich gut skalieren!

[Quelle: Digi-Key Produktkatalog, Sommer 2005]

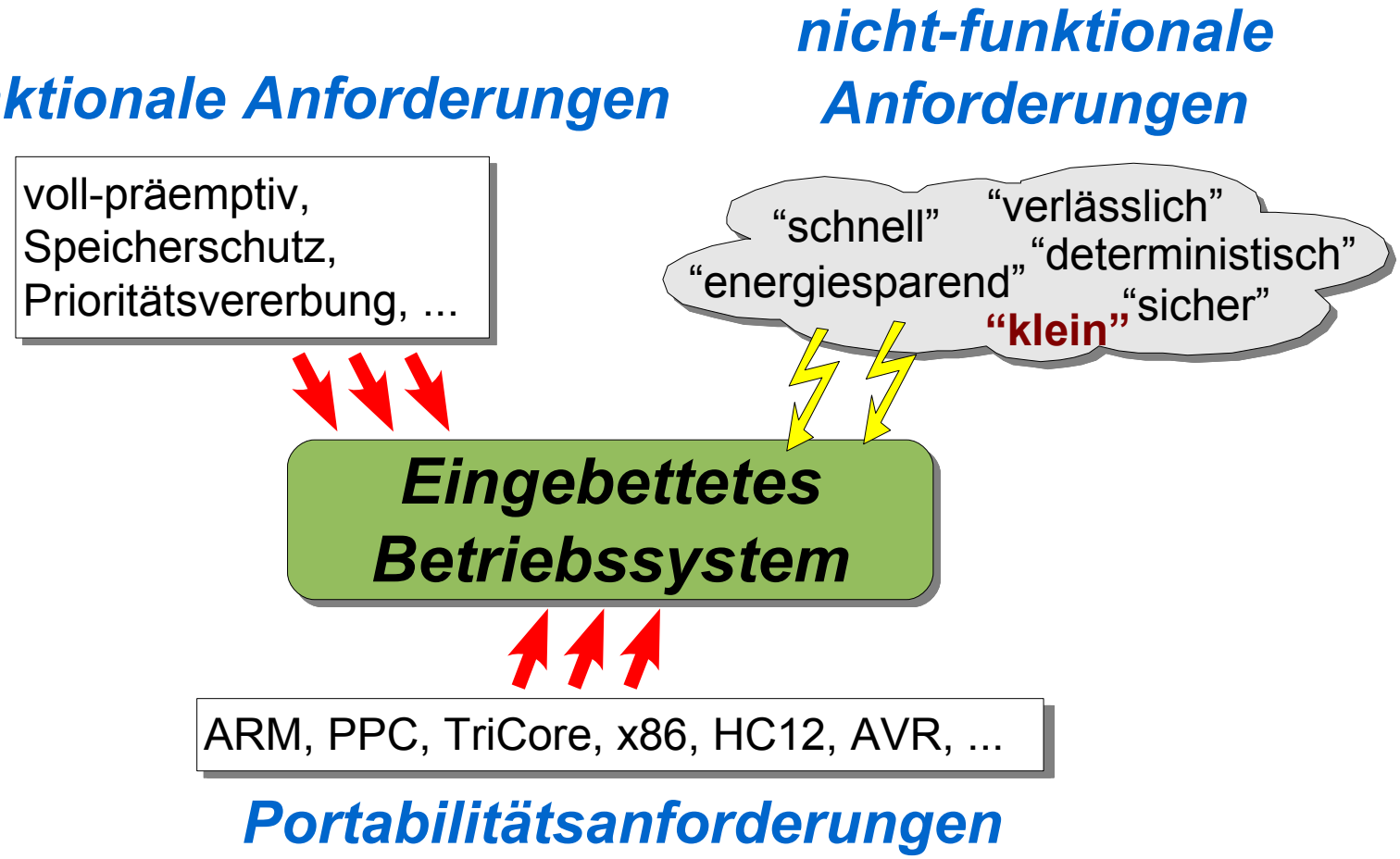
Eingebettete Systeme sind überall







Anforderungen sind vielfältig



- Anforderungen sind hochgradig **anwendungsspezifisch**
  - Vielzweckbetriebssysteme (wie im PC Bereich) versagen hier

Eingebettete Betriebssysteme

. . . , C{51, 166, 251}, CMX RTOS, C-Smart/Raven, eCos, eRTOS, Embos, Ercos, Euros Plus, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, OSEK {Flex, Turbo, Plus}, OSEKtime, Precise/MQX, Precise/RTCS, proOSEK, pSOS, PURE, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, RTA, RTX{51, 166, 251}, RTXC, Softune, SSXS RTOS, ThreadX, TinyOS, VRTX, VxWorks, . . .

Markt mit > 100 Systemen
> 50% Eigenentwicklungen

- „das Rad wird neu erfunden“
  - auch die selben Fehler werden wiederholt
- oftmals bietet **ein** BS Hersteller **mehrere** Systeme an
  - mit getrennter Code-Basis
  - getrieben durch die speziellen Anforderungen seiner Kunden

Konfigurierbare Betriebssysteme

- **Ansatz**: feingranulare, anwendungsspezifische, statische Konfigurierung
  - Ersparnis von Ressourcen
  - Abdeckung eines breiteren Anwendungsspektrums (=Marktes!)
  - Wiederverwendung und damit höhere Produktivität

Herausforderderungen

... beim Bau konfigurierbarer Systemsoftware

- Beherrschung der Variantenvielfalt
  - Analyse und Modellierung der Variabilität
- Minimierung der Modulabhängigkeiten, "Plug&Play"
  - Systementwurf
- Wahl geeigneter Sprachmittel für die Programmierung
  - Generizität und Wiederverwendbarkeit vs. Effizienz
- Werkzeugunterstützung
  - Technik zur Konfigurierung
- ...

Herausforderderungen

... beim Bau konfigurierbarer Systemsoftware

- Beherrschung der Variantenvielfalt
  - Analyse und Modellierung der Variabilität
- Mi
  - S
- W
  - G
- Werkzeugunterstützung
  - Technik zur Konfigurierung
- ...

Veranstaltung
Operating System Engineering
im Sommersemester 2006

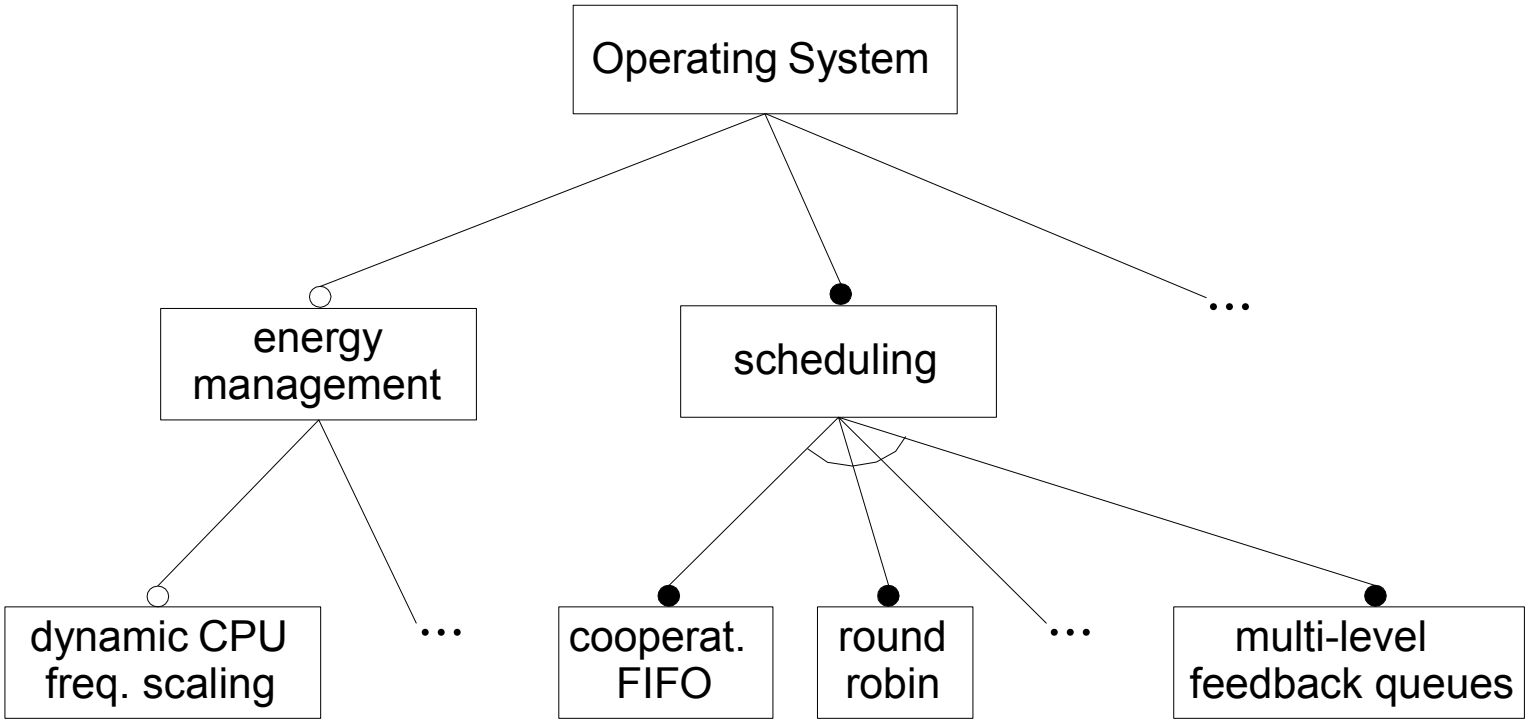
Operating System Engineering (OSE)

aka *Betriebssystemtechnik (BSE)*

- Vorlesung mit Übung (4 SWS, benoteter Schein)
- Inhaltliche Schwerpunkte: Vorlesung
  - Methoden, Techniken, Werkzeuge für die Entwicklung von konfigurierbarer Systemsoftware
  - Merkmalsmodellierung
  - Aspektorientierte Programmierung (AOP)
- Inhaltliche Schwerpunkte: Übung
  - Entwicklungsprojekt: Feingranulare BS-Produktline für RCX
  - Wie bekommt man Software klein?
  - Wie bekommt man Software konfigurierbar?

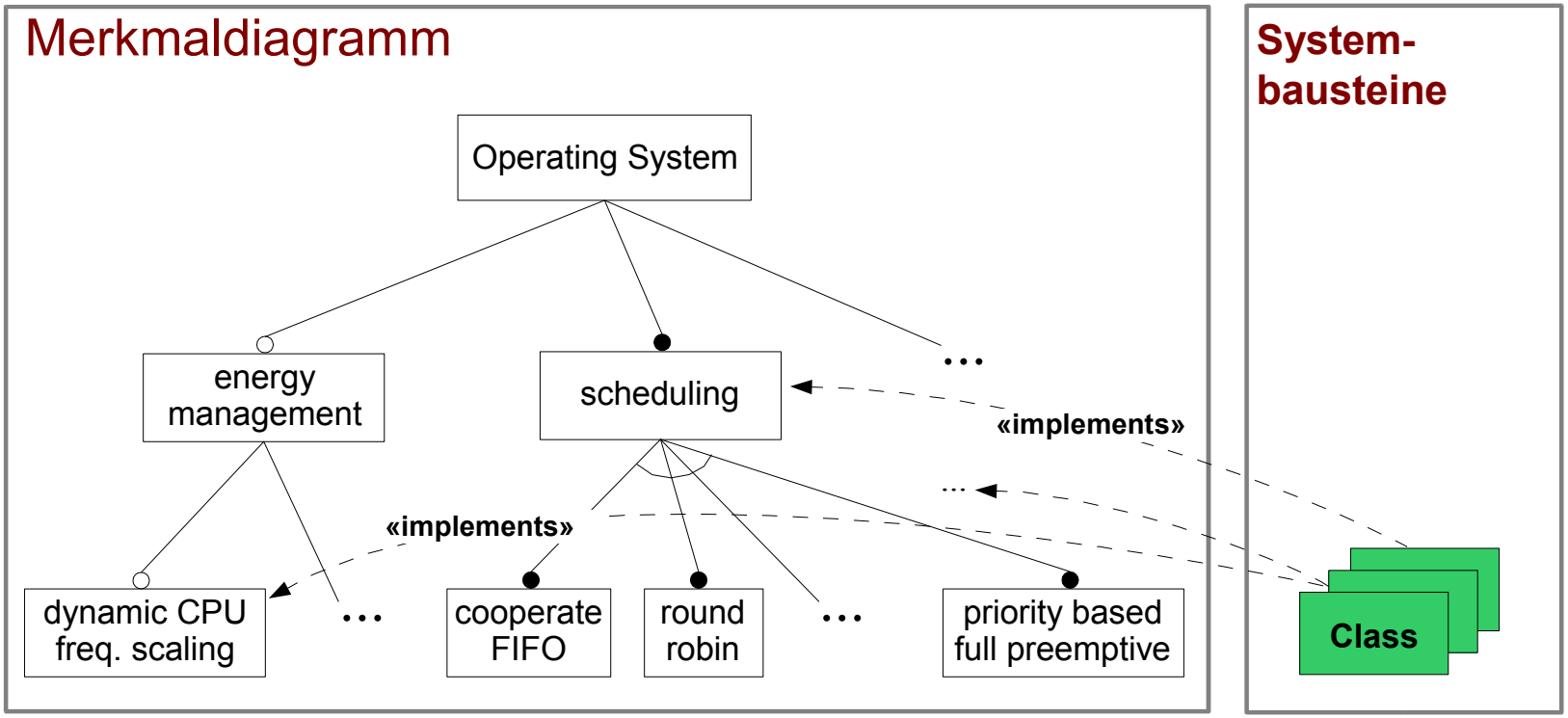
Merkmalmodelle

Merkmaldiagramm einer BS-Produktlinie



Systembausteine

Idealfall: ein Merkmal wird durch
eine **Klasse** implementiert



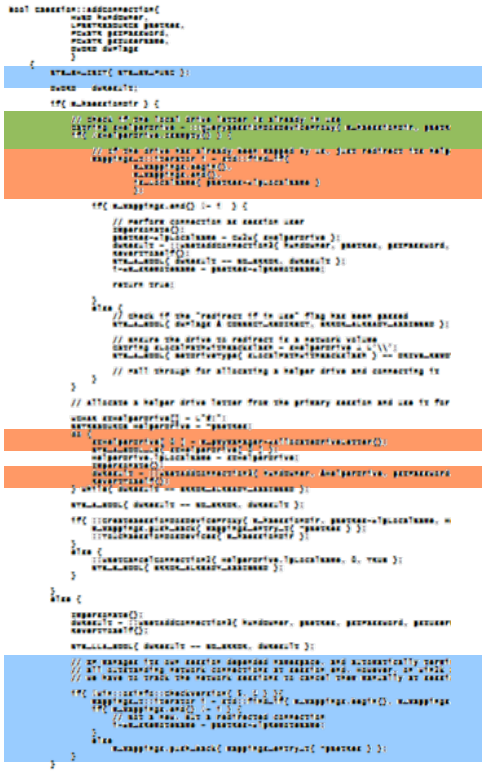
Problem: Querschneidende Belange

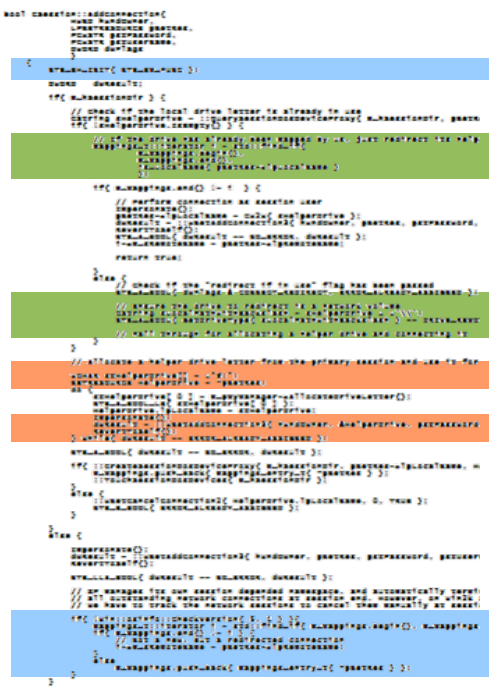
... behindern Konfigurierung und Wiederverwendung

Multi-User Support

Synchronisation

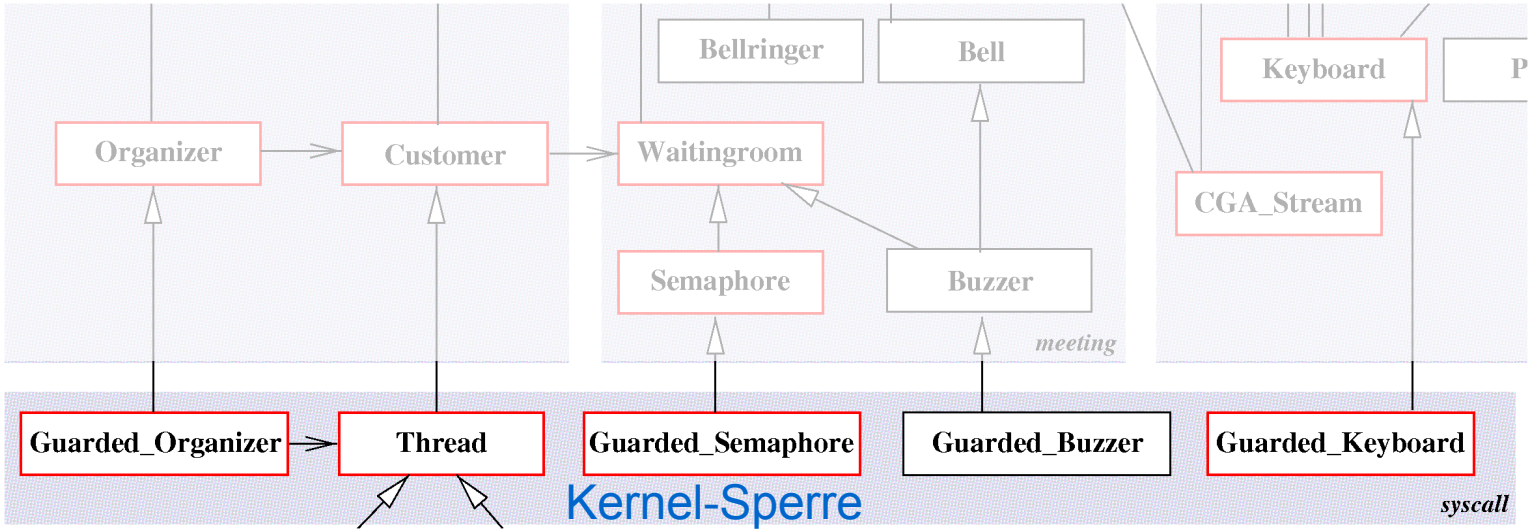
Entwicklungsunterst.





Beispiel aus OO-Stubs: Kernel-Sperre

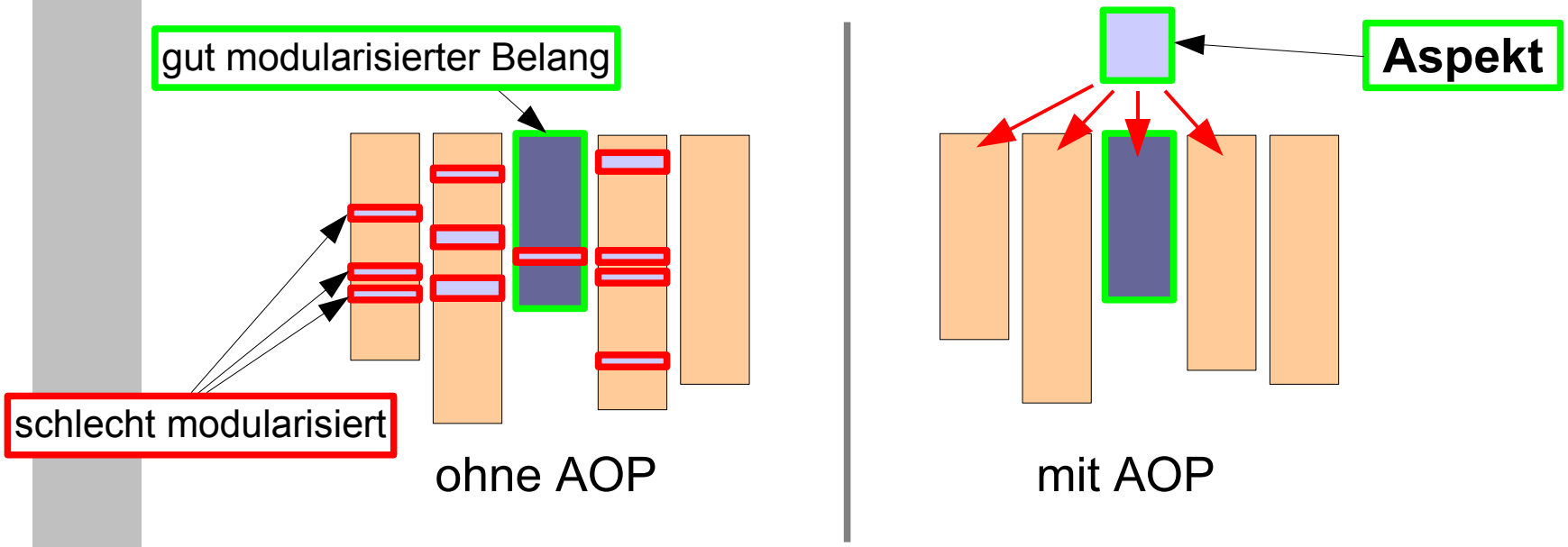
- Pro/Epilogmodell fordert:
  - *guard* sperren, bevor der Kern betreten wird
  - *guard* freigeben, nachdem der Kern verlassen wird
- Ansatz: Eigene Systemschnittstelle **durch Ableitung**



- Das ist gar nicht so doll! (Warum?)

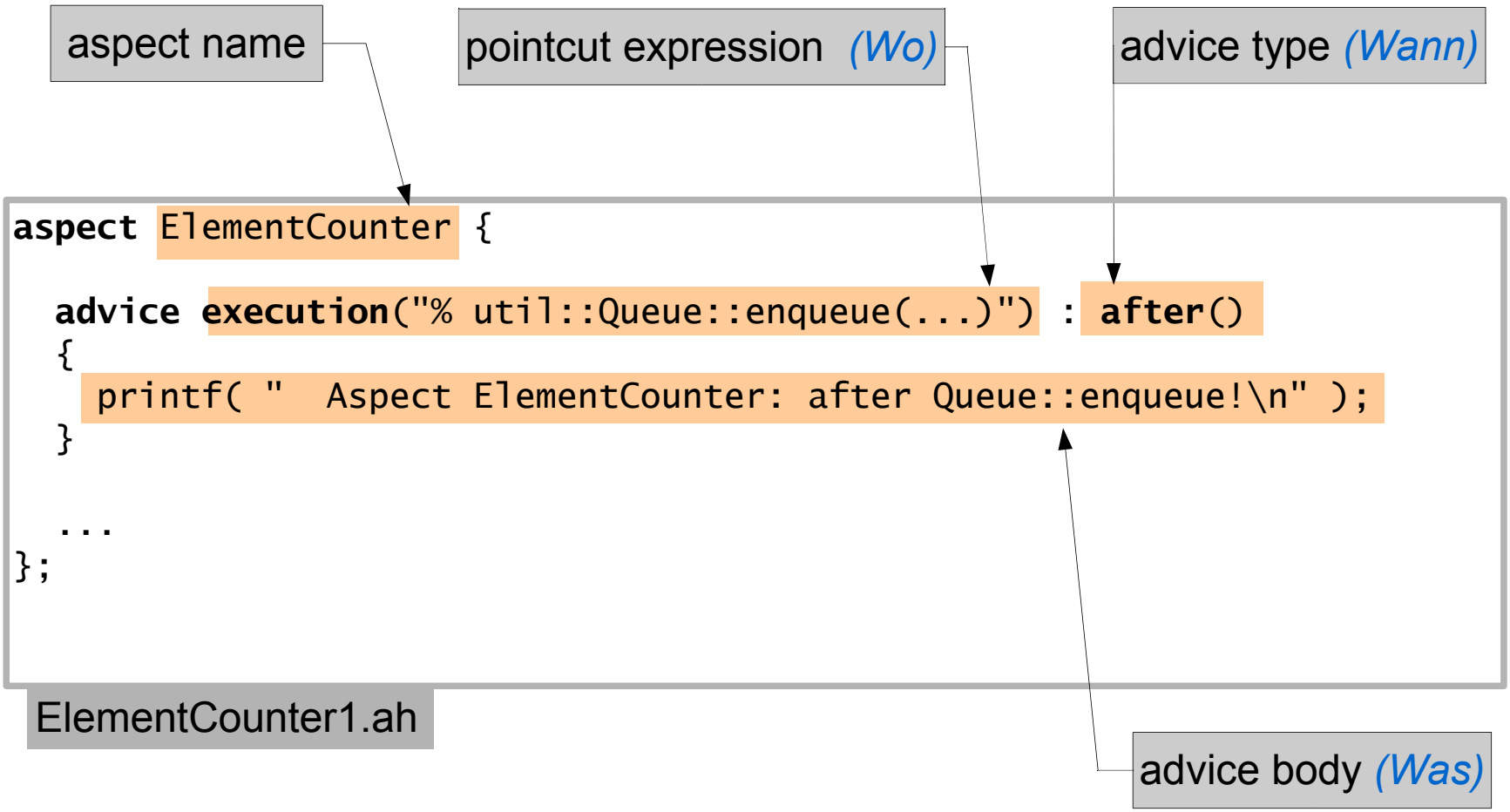
Aspect-Oriented Programming

- ...erlaubt die modulare Implementierung querschneidender Belange



- Beispiele: Kontrollflussverfolgung, Synchronisation, Sicherheit, Pufferung, Fehlerbehandlung, Überprüfung von Kontrakten, ...

AspectC++


aspect name
pointcut expression (Wo)
advice type (Wann)
aspect ElementCounter {
advice execution("% util::Queue::enqueue(...)") : after()
{
printf(" Aspect ElementCounter: after Queue::enqueue!\n");
}
...
};
ElementCounter1.ah
advice body (Was)

Before / After Advice

```
aspect B {

  advice execution("void ClassA::foo()") : before() {
  ...
  }

  advice execution("void ClassA::foo()") : after() {
  ...
  }

  advice call ("void ClassA::foo()") : before() {
  ...
  }

  advice call ("void ClassA::foo()") : after() {
  ...
  }
};
```

```
class ClassA {
public:
  void foo(){
    printf("ClassA::foo()"\n);
  }
}
```

```
int main(){
  printf("main()\n");
  ClassA a;
  a.foo();
}
```

Around Advice

with execution joinpoints:

advice execution(“void ClassA::foo()”) : **around**()
before code

tjp->proceed()

after code

```
class ClassA {
public:
  void foo(){
    printf(“ClassA::foo()”\n);
  }
}
```

with call joinpoints:

advice call(“void ClassA::foo()”) : **around**()
before code

tjp->proceed()

after code

```
int main(){
  printf(“main()\n”);
  ClassA a;
  a.foo();
}
```

OO-Stubs Kernel-Sperre mit AOP

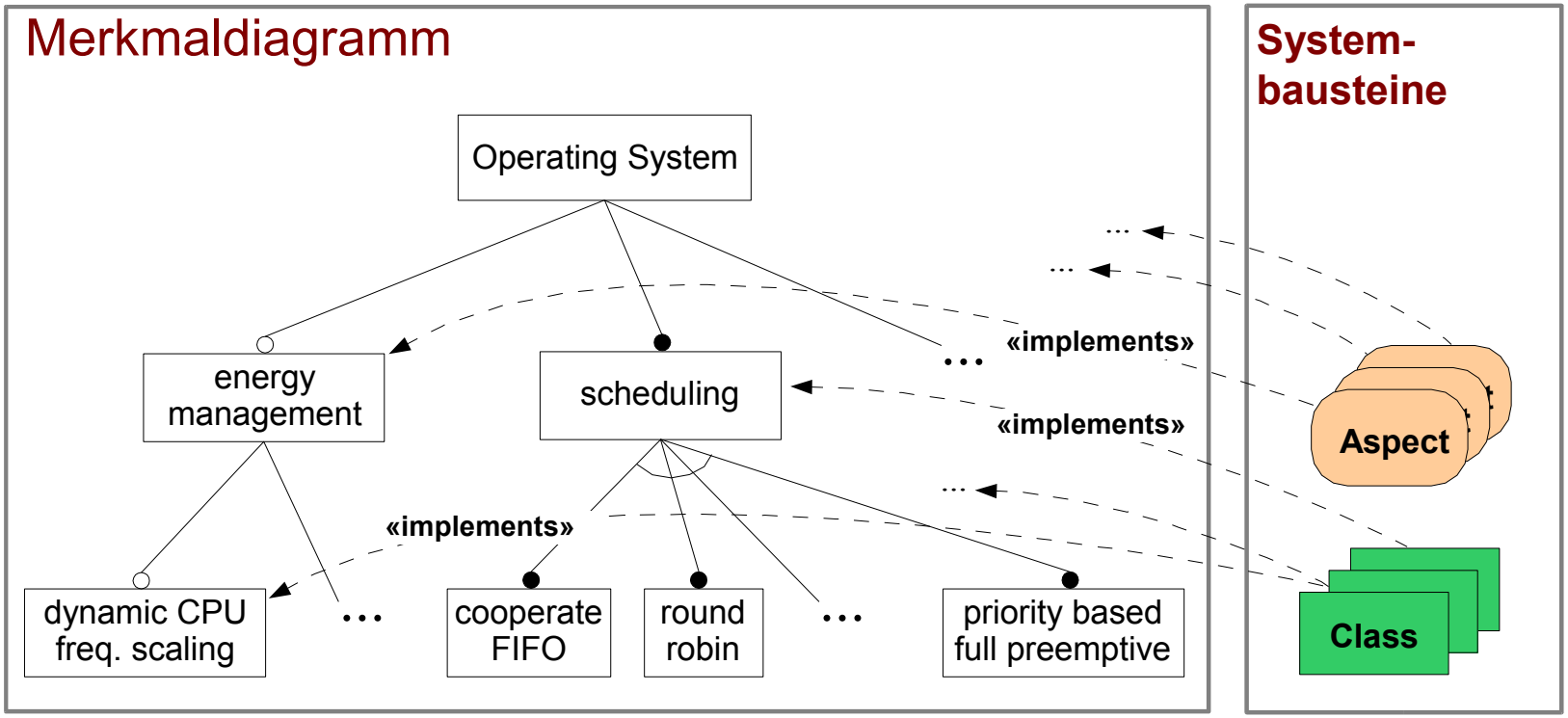
```
#include "guard/guard.h"
extern Guard guard;

aspect KernelLock {
  pointcut user() = "Application" || "Board" ...;
  pointcut kernel() = "Buzzer" || "Keyboard" ...;
  pointcut kernel_enter() = call(kernel())
                          && within(user());

   advice kernel_enter() : before() {
     guard.enter();
   }
   advice kernel_enter() : after() {
     guard.leave();
 }};
```

Aspekte als Systembausteine

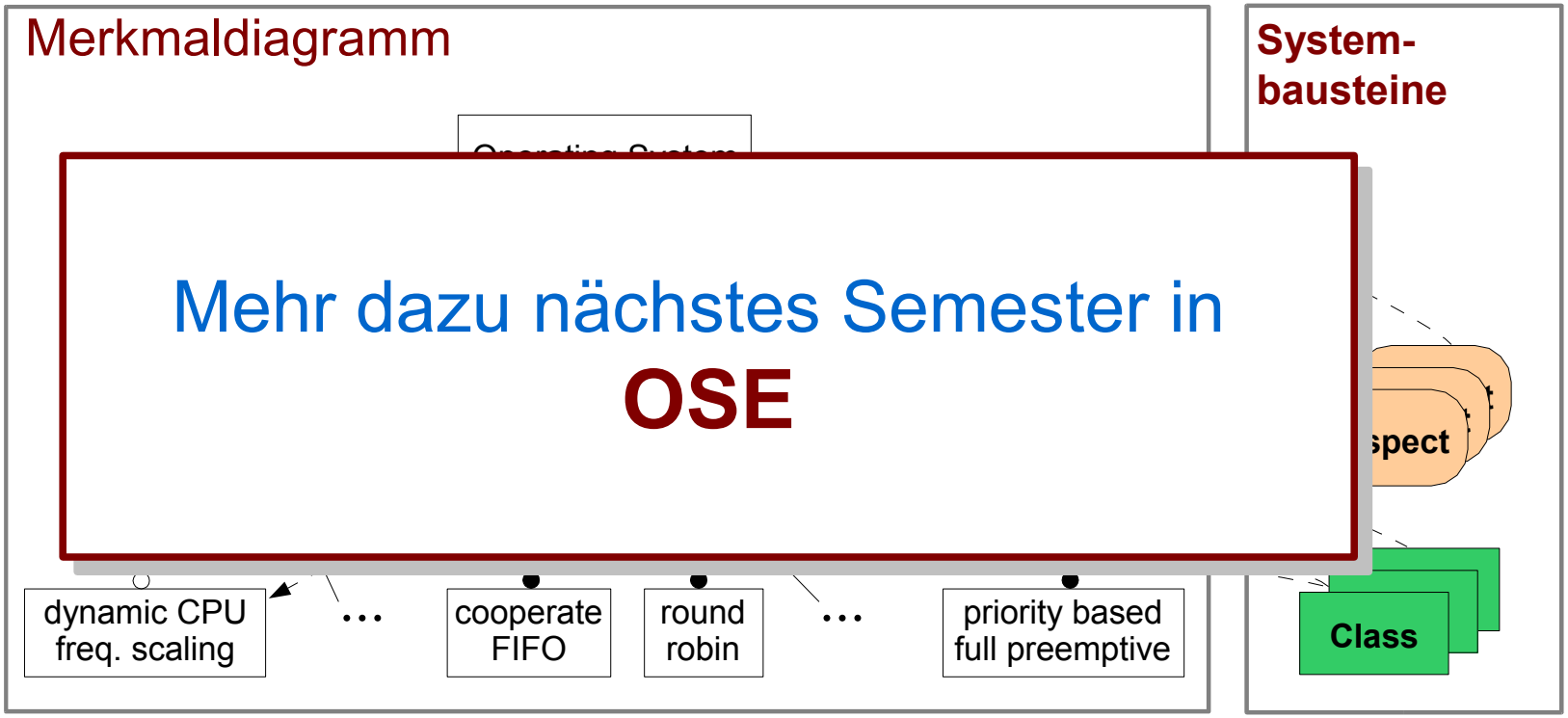
- **Querschneidende Belange** werden durch **Aspekte** implementiert



Merkmale mit querschneidender Implementierung können nun direkt auf ***ein*** Modul abgebildet werden.

Aspekte als Systembausteine

- **Querschneidende Belange** werden durch **Aspekte** implementiert



Merkmale mit querschneidender Implementierung können nun direkt auf ***ein*** Modul abgebildet werden.