Eingebettete Systeme:

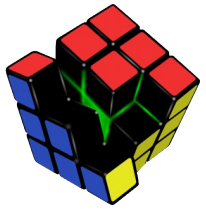
Statische Konfigurierung eingebetteter Systemsoftware

Olaf Spinczyk
Arbeitsgruppe Eingebettete Systemsoftware

Lehrstuhl für Informatik 12
Technische Universität Dortmund

olaf.spinczyk@udo.edu

http://ls12-www.cs.tu-dortmund.de/~spinczyk



ESS.udo

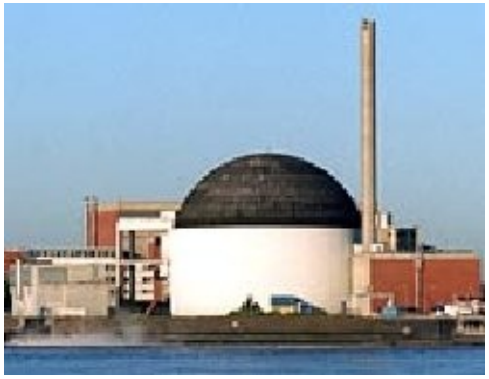
In dieser Vorlesung ...

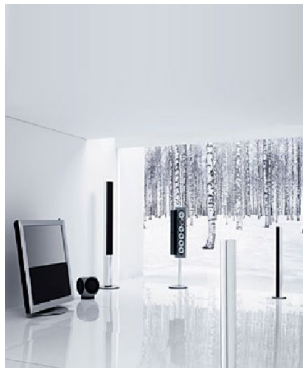
- Warum statische Konfigurierung?
- Warum kein Linux oder Windows?
- Wie baut man Spezialzwecksoftware?
- Welche Probleme gibt es in der Praxis?
- Welche Lösungsansätze werden verfolgt?

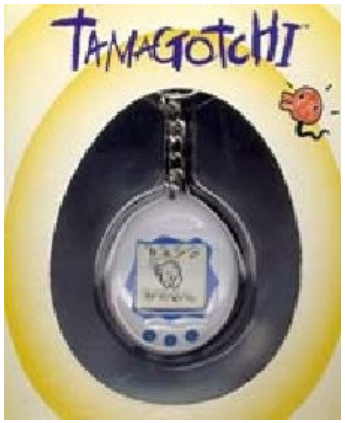
In dieser Vorlesung ...

- **Warum statische Konfigurierung?**
- Warum kein Linux oder Windows?
- Wie baut man Spezialzwecksoftware?
- Welche Probleme gibt es in der Praxis?
- Welche Lösungsansätze werden verfolgt?

Eingebettete Systeme sind überall

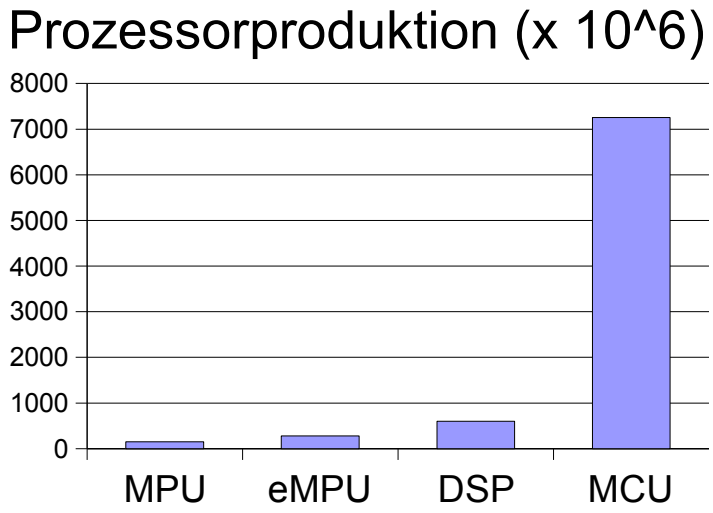


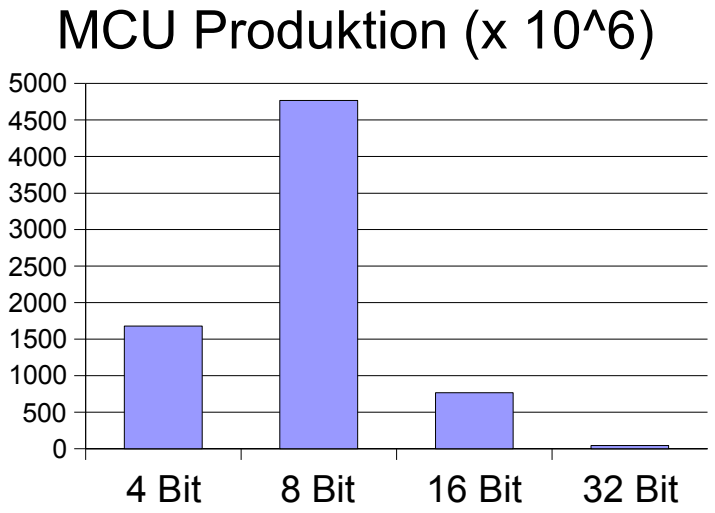




Eingebettete Systeme sind überall

- 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**

Beispiel: Eine modulare Wetterstation

- ein typisches kleines eingebettetes System

- Sensoren: Wind, Temperatur, Luftdruck, ...
- Aktoren: Display, Alarm, PC Verbindung, ...

- basiert auf einem AVR ATmega µ-controller
  - 8 Bit 4MHz RISC CPU
  - 2 – 128 kb Flash
  - 0.5 – 4 kb RAM
  - digitale, analoge, serielle und I²C basierte E/A

Beispiel: Eine modulare Wetterstation

Sensoren

Aktuatoren

Wind

Luftdruck

Temperatur

µController (AVR)

Prozessorkern
RAM, FLASH
Peripherie

Display

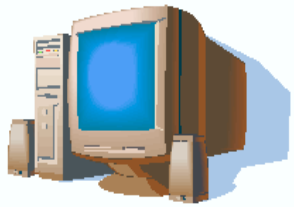
Lautsprecher

I^2C

USB

RS232

PC



Wetterstationsvarianten

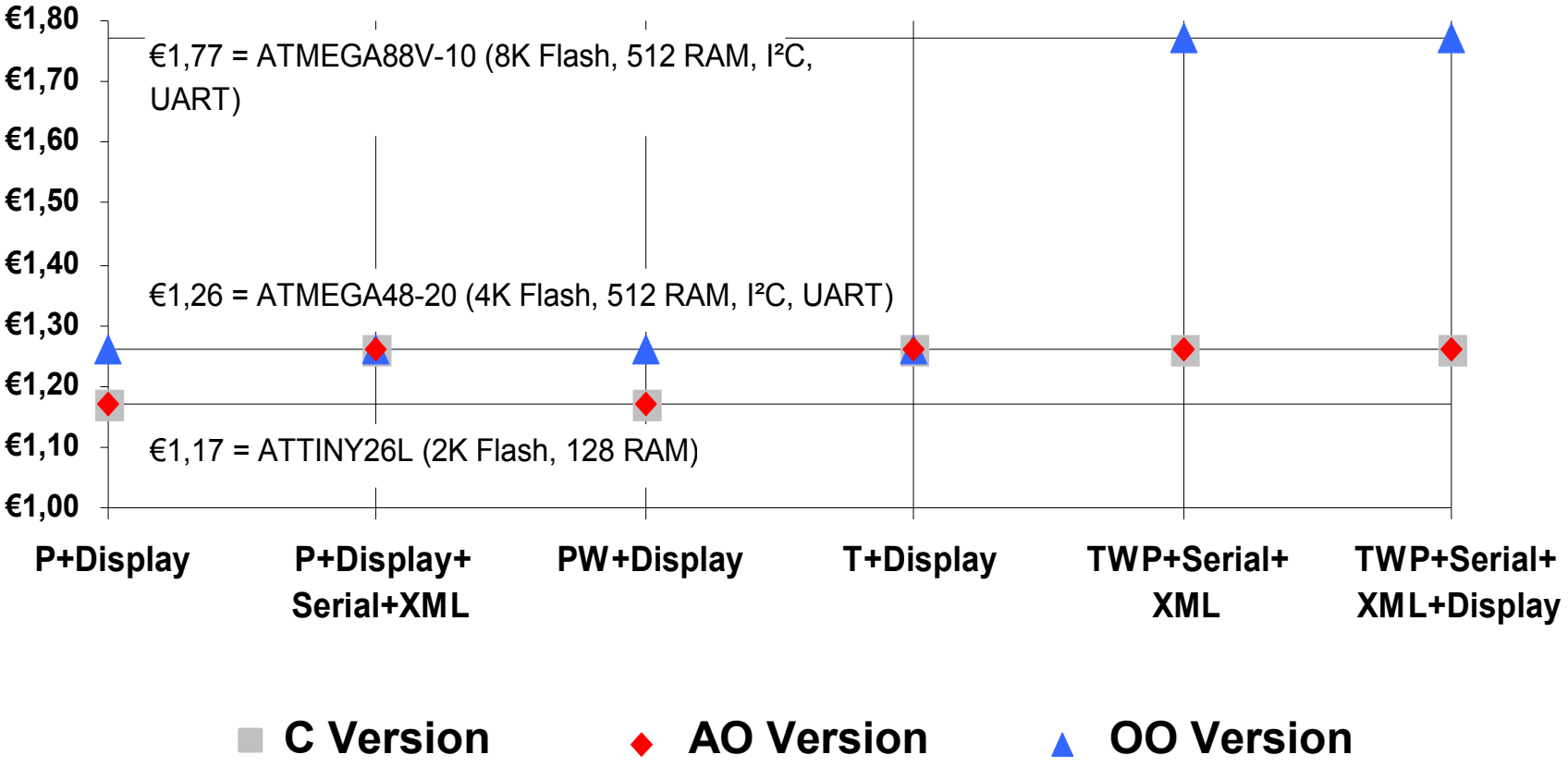
- *Thermometer*: LCD, Temperatur
- *Home*: LCD, Temperatur, Luftdruck
- *Outdoor*: LCD, Temp., Luftdruck, Wind

- *Deluxe*: + PC Verbindung
- *PC-only*: + PC Verbindung - LCD

- Serielle PC Verbindung
- USB PC Verbindung
- ...

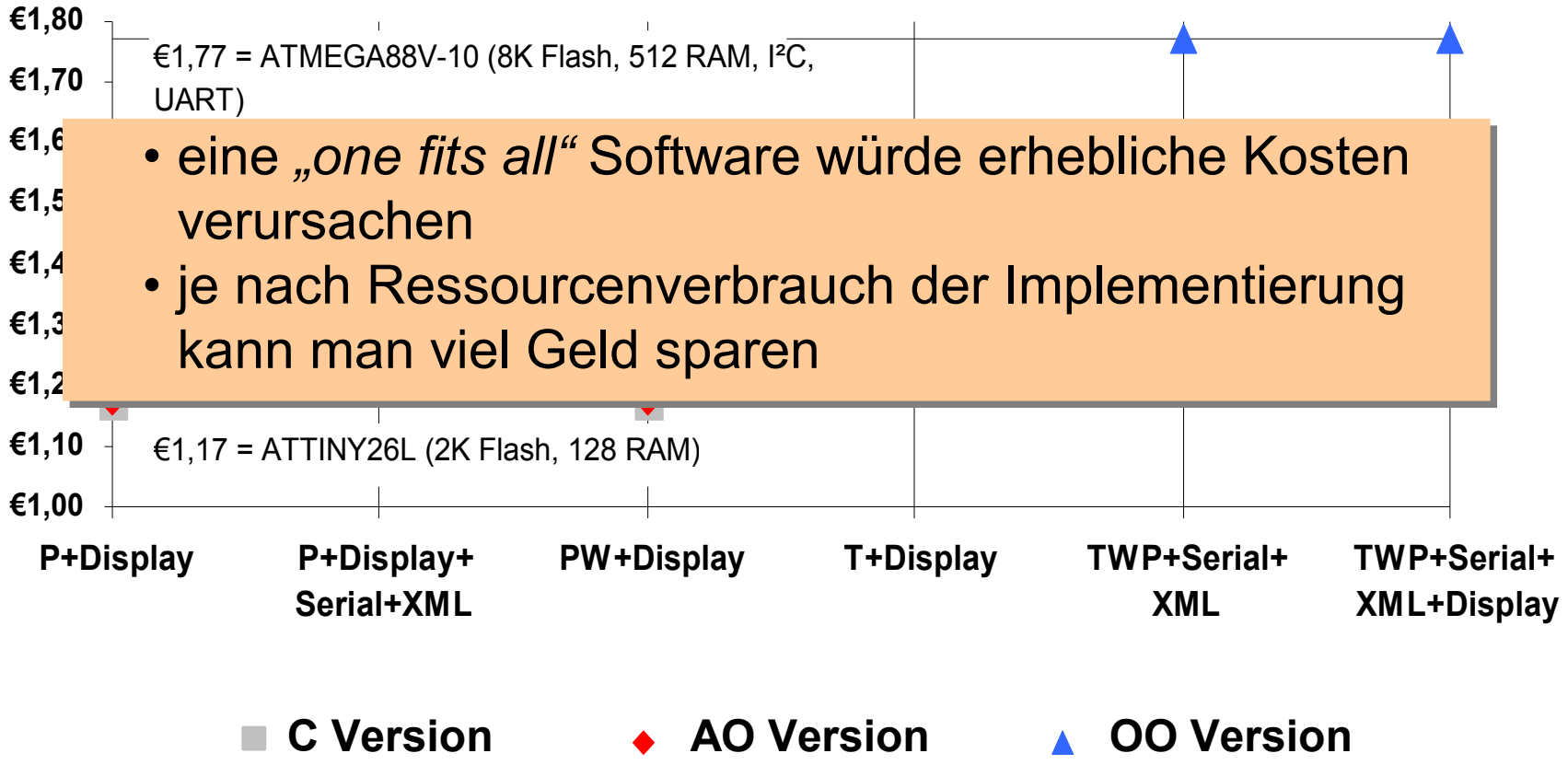
Kostenvergleich

AVR µ-Controller



Kostenvergleich

AVR µ-Controller



- eine *„one fits all“* Software würde erhebliche Kosten verursachen
- je nach Ressourcenverbrauch der Implementierung kann man viel Geld sparen

Zwischenfazit

- eingebettete Rechnersysteme werden oft in großen Stückzahlen produziert
  - Hardwarekosten sind erfolgbestimmend und müssen minimiert werden
- Aufgaben sind sehr unterschiedlich
- Hardware ist divers

Wichtige Folgerungen für die Systemsoftware:

- anwendungsspezifische Spezialisierung
- Portabilität und Abdeckung einer breiten Anwendungsspektrums

Statische Konfigurierung erlaubt beide Anforderungen gleichzeitig zu erfüllen!

In dieser Vorlesung ...

- Warum statische Konfigurierung?
- **Warum kein Linux oder Windows?**
- Wie baut man Spezialzwecksoftware?
- Welche Probleme gibt es in der Praxis?
- Welche Lösungsansätze werden verfolgt?

Das Betriebssystemdilemma

„Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with 'general purpose machines' that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being ***totally satisfactory to no-one.***“

A.M. Lister and R.D. Eager
Fundamentals of Operating Systems
Fourth Edition

Die Eier legende Wollmilchsau

Systeme wie Windows XP, Linux oder Solaris sind auf alle Eventualitäten vorbereitet, z.B.:

- fehlerhafte/böswillige Programme
  - präemptives *Scheduling*, Adressraumtrennung
- Mehrbenutzerbetrieb
  - Authentifizierung, Schutz von Dateien und Geräten
- vielfädige Programme, interagierende Prozesse
  - Semaphore, *Sockets*
- große/viele Programme
  - virtueller Speicher, *Swapping*, *Working Set* Konzept

Die Eier legende Wollmilchsau

- ein Vielzweckbetriebssystem ist für den wahrscheinlichsten Fall (den Normalfall) optimiert
- in allen Fällen, die von der künstlich definierten Norm abweichen, fallen Kosten an

- auch ungenutzte Funktionen haben einen Preis
  - Laufzeitverbrauch durch unnötige Fallunterscheidungen
  - Speicherplatzbedarf
  - erhöhte Startzeiten
  - Verschlechterung der *cache-hit* Raten
- besonders problematisch sind Eigenschaften, die sich auf viele Systemfunktionen auswirken
  - Linux 2.6 Kern: `grep EPERM` liefert **1243** Treffer!

Spezielle BS-Einsatzgebiete

- harte und weiche Echtzeitsysteme
  - sicherheitskritische Systeme in Fahrzeugen, Multimedia
  - → deterministische Laufzeiten
- Hochleistungsrechensysteme
  - Parallelrechner, Cluster
  - → minimale Laufzeiten
- kleine Systeme
  - **eingebettete Systeme** in Massenproduktion
  - → minimaler Speicherplatzverbrauch
- ... werden meist mit speziellen Betriebssystemen betrieben

In dieser Vorlesung ...

- Warum statische Konfigurierung?
- Warum kein Linux oder Windows?
- **Wie baut man Spezialzwecksoftware?**
- Welche Probleme gibt es in der Praxis?
- Welche Lösungsansätze werden verfolgt?

Vielzwecksystem – Vielzweckfunktion

eine Analogie:

- das Vielzwecksystem
  - stellt Systemdienste bereit
  - versucht, allen Nutzern gerecht zu werden
  - ist Resultat vieler Kompromisse
  - verbirgt die interne Struktur (black box)
- die Vielzweckfunktion
  - stellt Teilfunktionen bereit
  - ... sonst wie oben

Vielzweckfunktion printf()

```
> uname -snrm
Linux faui48 2.6.5-7.111.19-bigsmp i686
> echo 'main(){printf("hello, world\n");}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> ls -l hello*
-rwxr-xr-x  1 spinczyk i4staff 2008562 2005-04-06 17:14 hello
-rw-r--r--  1 spinczyk i4staff      34 2005-04-06 17:13 hello.c
-rw-r--r--  1 spinczyk i4staff     856 2005-04-06 17:14 hello.o
> size hello hello.o
   text    data     bss     dec     hex filename
 367486    3156    3220  373862   5b466 hello
     36       0       0      36      24 hello.o
```

Vielzweckfunktion `printf()`

ist die Größe ein Linux/x86 spezifisches Phänomen?

Programm	Größe (Bytes)					
	Linux				Solaris	Windows
	i686	arm	ppc	alpha	sparc	i586
hello	203966	221998	245452	453898	183373	30935
hello.o	29	42	60	62	38	29
%	0,014	0,018	0,024	0,013	0,020	0,094

wosch, OSE 2003

- nein, hello ist auf allen Plattformen sehr groß
- innerhalb der letzten Jahre hat sich das Problem auch noch verschlimmert: **203966 zu 373862**

Spezialzweckfunktion puts()

- vermutlich kann `printf()` einfach zu viel
- das spezialisierte `puts()` sollte weniger Speicher verbrauchen ...

```
> echo 'main(){puts("hello, world");}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss     dec     hex filename
 367486    3156    3220  373862   5b466 hello
     36       0       0      36      24 hello.o
```

373862 zu 373862!

Spezieller geht's nicht: `write()`

- der `write()` *system call* sollte wirklich wenig kosten ...

```
> echo 'main(){write(1,"hello, world\n",13);}' > hello.c
> gcc -O6 -c hello.c; gcc -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss     dec     hex filename
 367134    3100    3252  373486   5b2ee hello
     39       0       0      39      27 hello.o
```

373862 zu 373486 :-(

Der Einfluss des *Startup Codes*

Wieso wird der Speicherplatzverbrauch nicht signifikant kleiner?

`objdump --reloc -D hello | grep printf`
... liefert **1488 Treffer**. Es ist der *Startup Code*!

```
> echo '_start(){write(1,"hello, world\n",13);_exit(0);}' >
hello.c
> gcc -O6 -c hello.c; gcc -static -nostartfiles -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss     dec     hex filename
    354       0       8     362     16a hello
     46       0       0      46      2e hello.o
```

373862 zu 362!

Hello, World - Fazit

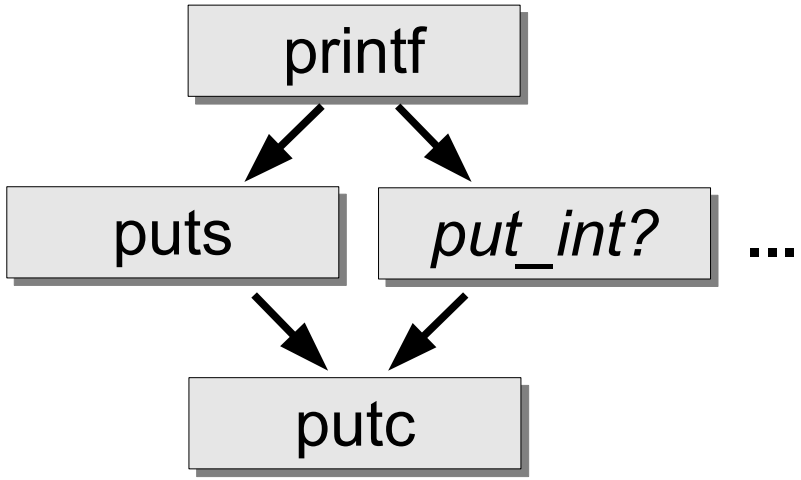
Ein typisches „hello, world“ Programm braucht unter Linux/x86 mindestens **1000 mal mehr Speicher als notwendig**. Ursachen?

- Softwarestruktur
  - Modularisierung und Offenlegung der Struktur
- Programmiersprache
  - Schnittstellen
- Werkzeugkette
  - Compiler, Objektmodule, Binder

Ursachen - Softwarestruktur

- `printf()` unterstützt vieles:
  - alle „plain old data types“
    - [signed|unsigned] [short|long] int, float, double, char, void*, char*
  - verschiedene Darstellungsformen
    - binär, oktal, dezimal, hexadezimal
  - unterschiedliche Ausrichtungen
    - rechts/links, benutzerdefinierte Feldbreiten
- nicht jede Applikation benötigt all diese Features
  - „Hello, World“: char*, linksbündig
- trotzdem „zahlt“ jede Applikation dafür!

Ursachen - Softwarestruktur



- aber basiert printf() auf puts()?
  - oder vielleicht puts() auf printf()?
  - oder gibt es gar keine Beziehung
- die Beziehungen zwischen den Funktionen sind nicht Teil der libc Schnittstellendefinition
  - ein Nachteil des black box Prinzips

Ursachen - Programmiersprache

- printf() soll eine einheitliche Schnittstelle für die formatierte Ausgabe bereitstellen.
  - Interpretation der Format-Zeichenkette zur Laufzeit
  - Referenzierung von Ausgabefunktionen für alle unterstützten Datentypen und Formate
  - keine Möglichkeit der Optimierung durch den Übersetzer, da printf() eine Bibliotheksfunktion ist
- C++ stellt mit den << Operator ebenfalls eine einheitliche Schnittstelle zur formatierten Ausgabe bereit
  - der Übersetzer ermittelt die aufgerufene Ausgabefunktion statisch
  - unbenötigte Funktionen werden nicht referenziert

Ursachen - Programmiersprache

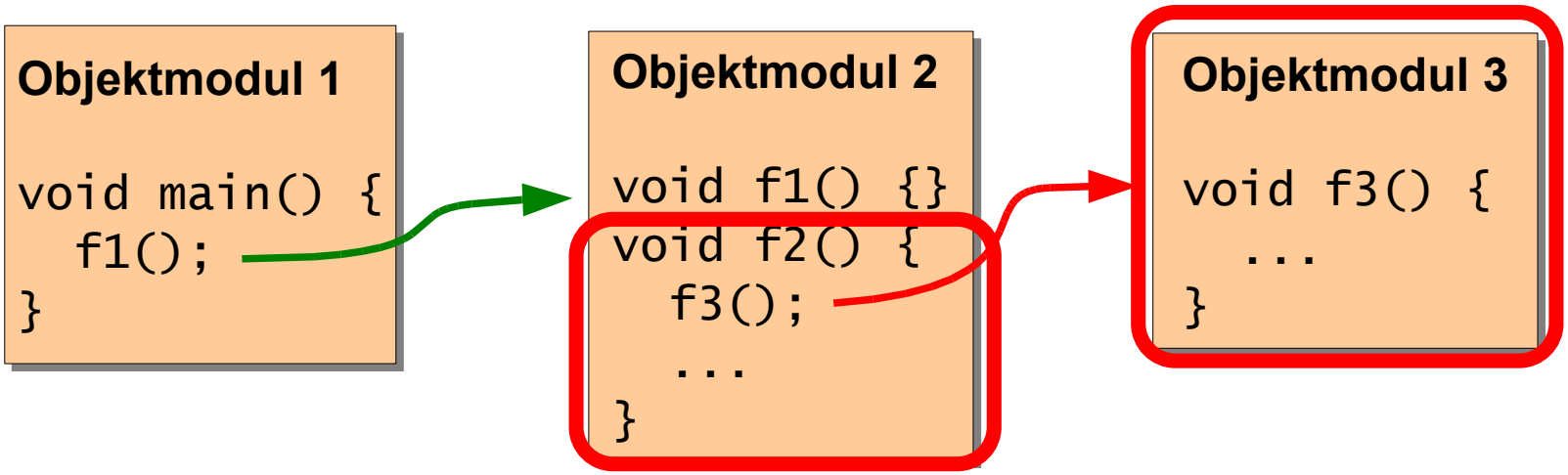
- ist `std::cout << „hello, world\n“;` schlanker?

```
> cat > hello.cc
#include <iostream>
int main() {
  std::cout << "hello, world\n";
}
> g++ -O6 -c hello.cc; g++ -static -o hello hello.o
> ./hello
hello, world
> size hello hello.o
   text    data     bss     dec     hex filename
 781498    3952   24116  809566   c5a5e hello
    202      28       1     231      e7 hello.o
```

373862 zu 809566!

Ursachen - Werkzeugkette

- die GNU Werkzeugkette (gcc/g++, ar, ld) unterstützt (standardmäßig) kein Binden auf Funktionsebene



- obwohl `f2()` und `f3()` nicht referenziert werden, landen sie als unbenutzer Code im gebundenen Programm!

Ist das Problem wirklich eines?

- `printf()` und `cout` sind keine Ausnahmen
  - Programme werden immer größer, selbst wenn sie funktional nicht reicher werden
  - grundlegende Prinzipien der Softwaretechnik werden vernachlässigt

> *„Some users may require only a subset of services or features other users need. These **'less demanding' users** may demand that they are not be forced to pay for the resources consumed by the unneeded features.“*

D.L. Parnas, 1979
Designing Software for Ease of Extension and Contraction

Ist das Problem wirklich eines?

- wie Vielzweckbetriebssysteme sind auch die GNU libc und die GNU libstdc++ für den „Normalfall“ optimiert
  - das Betriebssystem wird es schon richten
    - virtueller Speicher, *shared libraries*
  - Speicherplatzverbrauch ist im „Normalfall“ kein Problem
    - sollte man deshalb verschwenderisch damit umgehen?
- ***im*** Betriebssystem sieht es schon problematischer aus
  - virtueller Speicher und *shared libraries* helfen hier nicht
  - jede Anwendung hat zu leiden
- unbenutzbar wäre ein nach dem „`printf()` Prinzip“ gebautes Vielzweckbetriebssystem in Domänen mit speziellen Anforderungen

In dieser Vorlesung ...

- Warum statische Konfigurierung?
- Warum kein Linux oder Windows?
- Wie baut man Spezialzwecksoftware?
- **Welche Probleme gibt es in der Praxis?**
- Welche Lösungsansätze werden verfolgt?

Eingebettete Betriebssysteme

Wie sieht ein Betriebssystem aus, das speziellen Anwendungen auf spezieller Hardware möglichst optimale Unterstützung bieten kann?

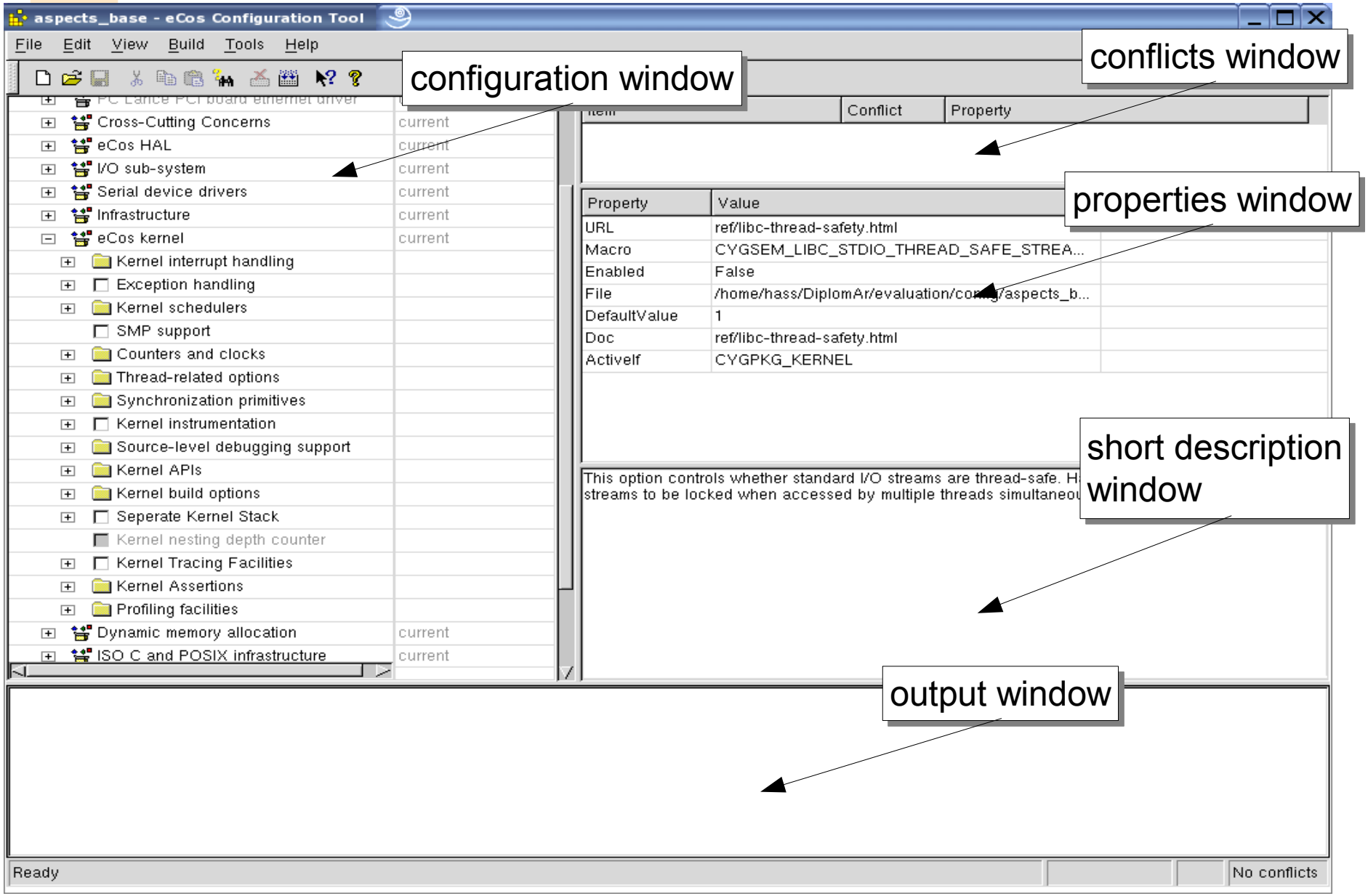
- der Markt hat vielfältige Angebote
  - . . . , C{51, 166, 251}, CiAO, CMX RTOS, C-Smart/Raven, eCos, eRTOS, Embos, Ercos, Euros Plus, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, OSEK {Flex, Turbo, Plus}, OSEKtime, Precise/MQX, Precise/RTCS, proOSEK, pSOS, PURE, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, RTA, RTX{51, 166, 251}, RTXC, Softune, SSXS RTOS, ThreadX, TinyOS, VRTX, VxWorks, . . .
- über 50% des Marktes werden von proprietären Systemen abgedeckt

eCos – eine Betriebssystemfamilie

... dient hier als Beispiel für den Stand der Kunst

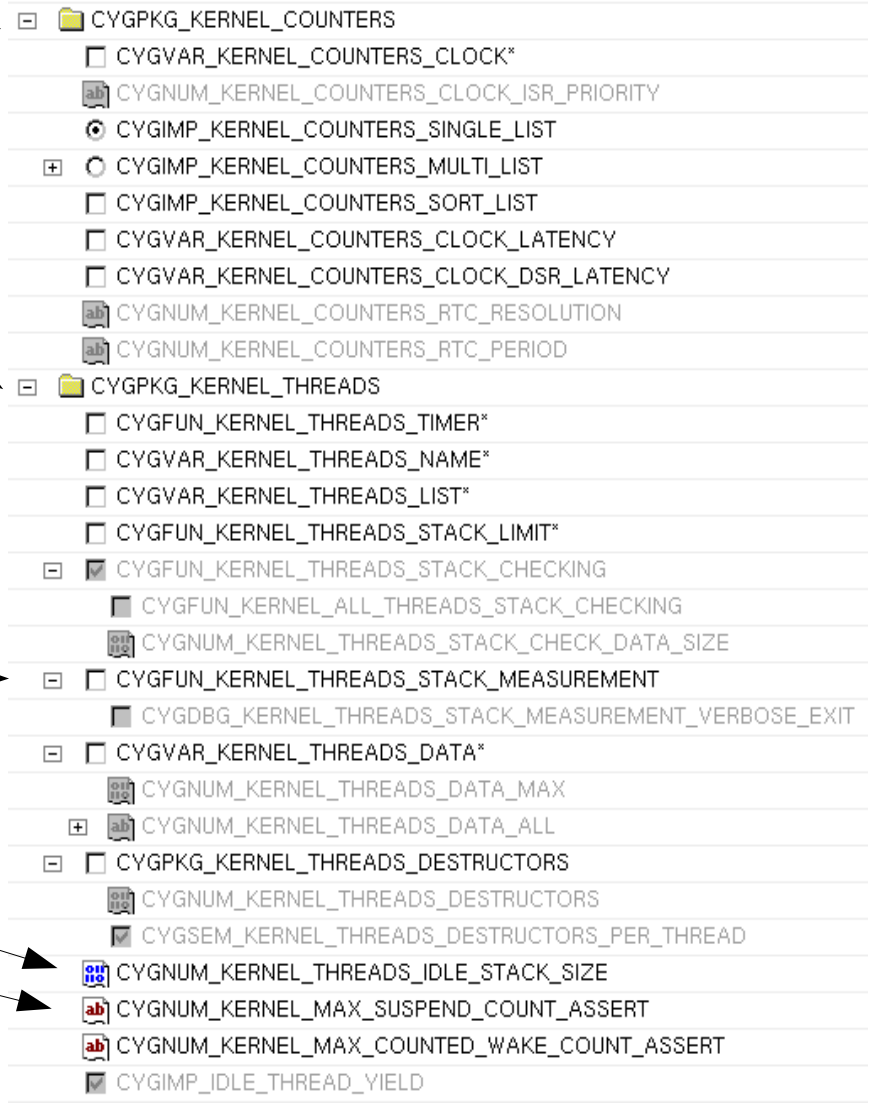
- Zieldomäne: eingebettete Systeme
- Ansatz: Ressourcen sparen durch statische anwendungsspezifische Konfigurierung
- Implementierungssprache: C und C++ (Kernel!)
- Lizenz: Open Source
  (früher Cygnus Solutions, heute RedHat)

Konfigurationswerkzeug



Konfigurierungseinheiten

- Pakete
  - Quellcodebündel, oberste Konfigurierungsebene
- Komponenten
  - Logische Konfigurierungs-einheiten unterhalb der Packages (hierarchisch)
- Optionen
  - logisch
  - Integer Werte
  - Zeichenketten
  - Aufzählungstypen



Komponentenkonfigurierung

```
#include <pkgconf/kernel.h>
#include <cyg/infra/cyg_trac.h>

void some_func() {
  CYG_REPORT_FUNCTION();
  ...
#ifdef SOME_OPTION
  ...
#endif
  ...
  CYG_REPORT_RETURN();
}
```

```
#define SOME_OPTION
// #define TRACE_KERNEL
```

```
#include <pkgconf/kernel.h>
#ifdef TRACE_KERNEL
#define CYG_REPORT_RETURN() \
  ...
#else // leer!
#define CYG_REPORT_RETURN()
#endif
```

- Berücksichtung der Konfiguration erfolgt über bedingte Übersetzung (#ifdef)
- Konfigurierbare quer schneidende Belange werden über Makros realisiert, um #ifdefs zu reduzieren

Eine Beispielkomponente

27 Zeilen Quelltext

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked   = false;
    owner    = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
     CYG_REPORT_RETURN();
}
```

Eine Beispielkomponente

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked   = false;
    owner    = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```

2 Zeilen für die
Kontrollflussverfolgung

Eine Beispielkomponente

21 (unleserliche) Zeilen für **optionale Merkmale**

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked   = false;
    owner    = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```

Eine Beispielkomponente

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked   = false;
    owner    = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifdef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
     CYG_REPORT_RETURN();
}
```

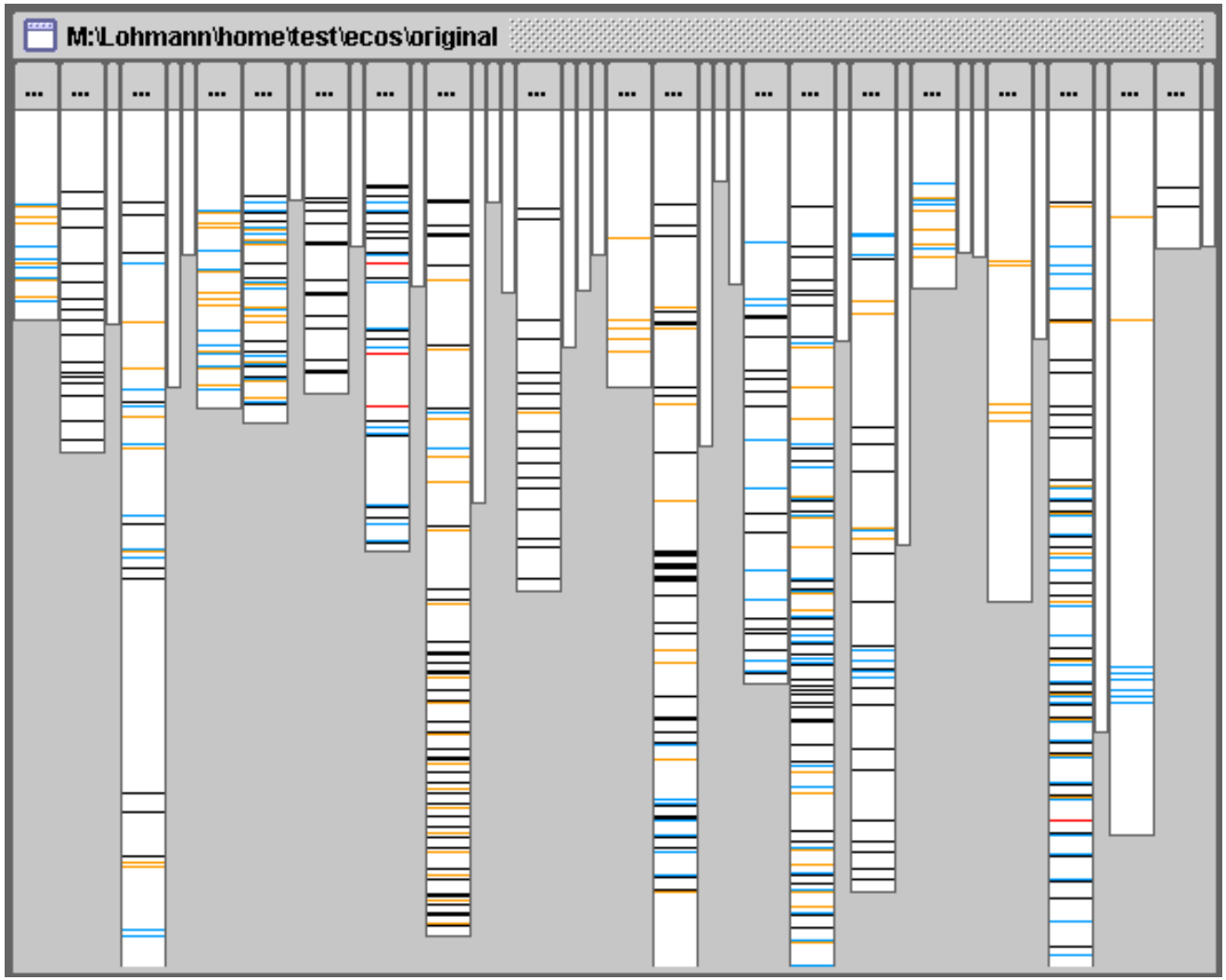
4 Zeilen für die
eigentliche Implementierung

Quer schneidende Belange

synchronization

instrumentation

tracing



Anteil quer schneidender Belange

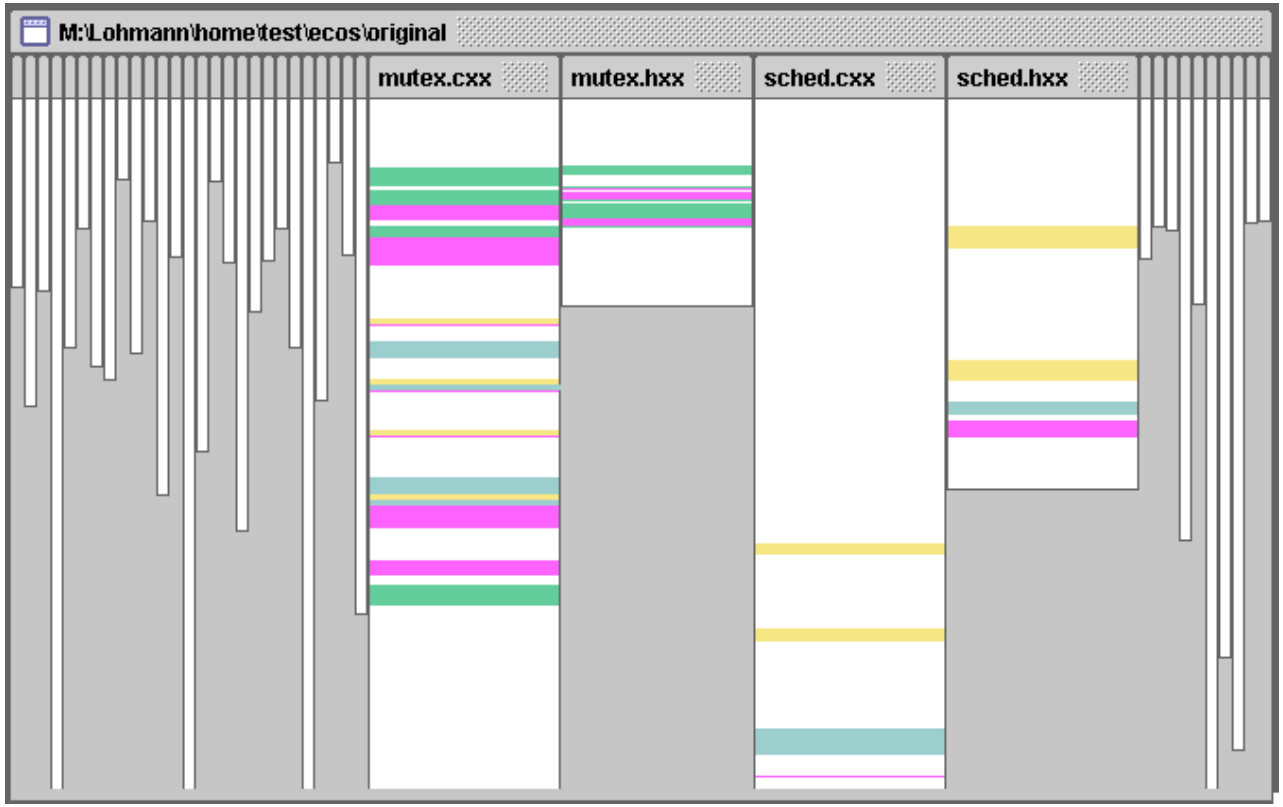
- Untersucht wurden in C++ implementierte Pakete
  - Kernel
  - libc
  - Memory Management
  - Wallclock/Watchdog
  - POSIX/µITRON

	Kernel		Memory Management		Gesamt	
LOC	5205	100,00%	2813	100,00%	16535	100,00%
Tracing	336	6,46%	66	2,35%	938	5,67%
Assertions	384	7,38%	151	5,37%	793	4,80%
Profiling	319	6,13%	0	0,00%	319	1,93%
Locking	186	3,57%	40	1,42%	300	1,81%
Gesamt	1225	23,54%	257	9,14%	2350	14,21%

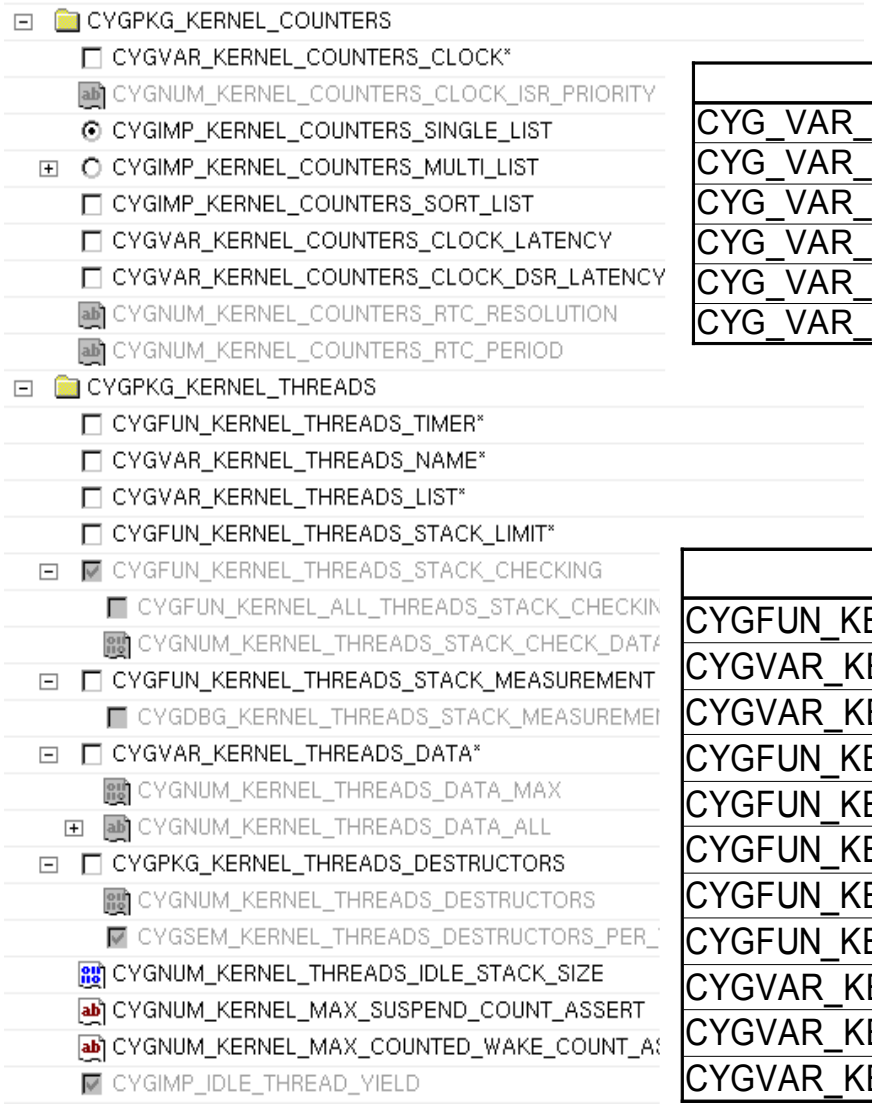
Konfigurationsoptionen

Varianten des Protokolls zur Vermeidung der Prioritätsumkehr bei *Mutex*-Verwendung

- simple
- ceiling
- inheritance
- dynamic



Variationspunkte pro Option



Option	#
CYG_VAR_KERNEL_COUNTERS_CLOCK	42
CYG_VAR_KERNEL_COUNTERS_SINGLE_LIST	7
CYG_VAR_KERNEL_COUNTERS_MULTI_LIST	7
CYG_VAR_KERNEL_COUNTERS_SORT_LIST	2
CYG_VAR_KERNEL_COUNTERS_CLOCK_LATENCY	20
CYG_VAR_KERNEL_COUNTERS_CLOCK_DSR_LATENCY	3

Option	#
CYGFUN_KERNEL_THREADS_TIMER	95
CYGVAR_KERNEL_THREADS_NAME	15
CYGVAR_KERNEL_THREADS_LIST	10
CYGFUN_KERNEL_THREADS_STACK_LIMIT	9
CYGFUN_KERNEL_THREADS_STACK_CHECKING	10
CYGFUN_KERNEL_ALL_THREADS_STACK_CHECKING	1
CYGFUN_KERNEL_THREADS_STACK_MEASUREMENT	10
CYGFUN_KERNEL_THREADS_STACK_MEASUREMENT_.	2
CYGVAR_KERNEL_THREADS_DATA	8
CYGVAR_KERNEL_THREADS_DESTRUCTORS	6
CYGVAR_KERNEL_THREADS_DESTRUCTORS_PER_...	13

Diskussion

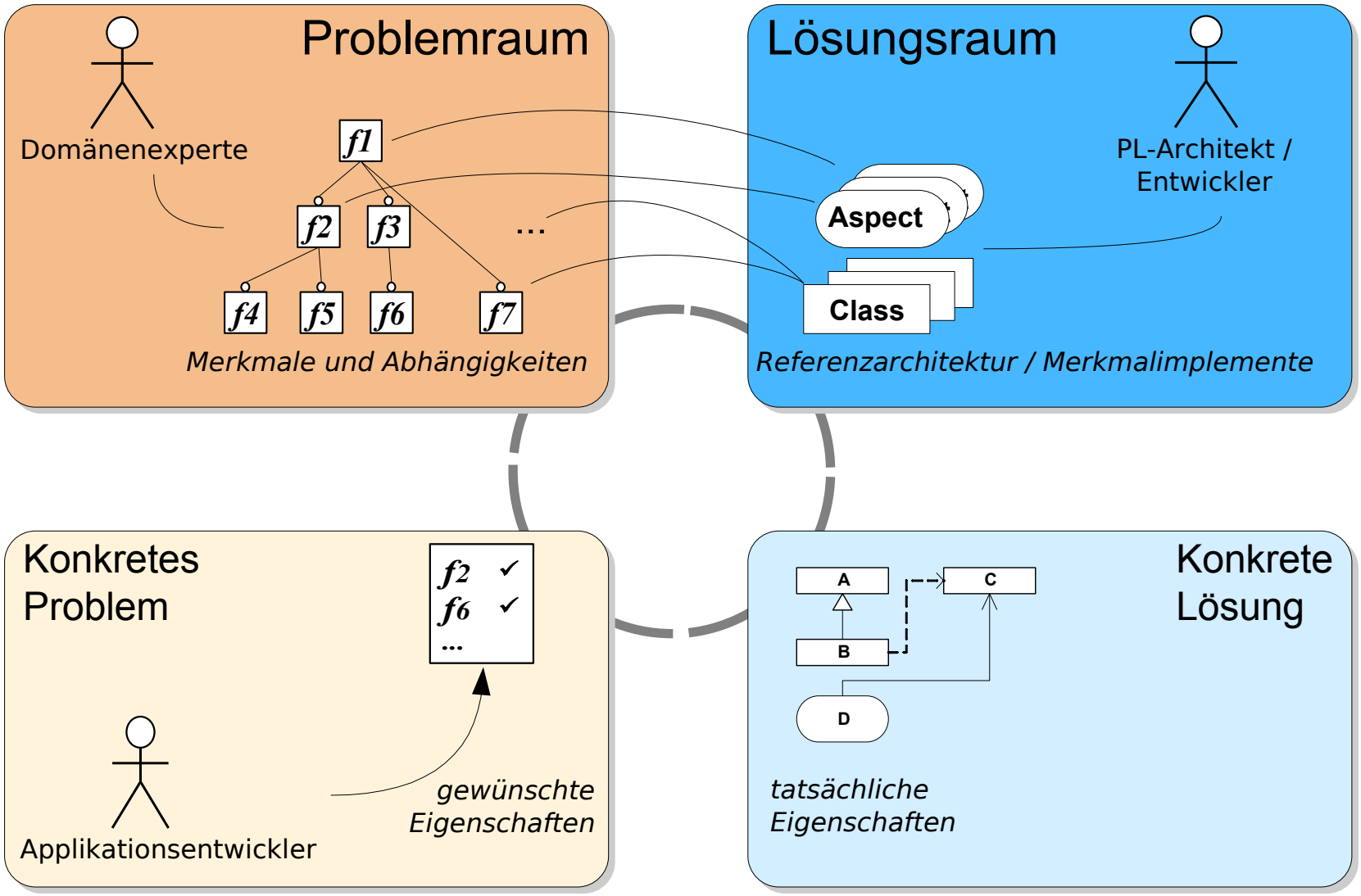
- eCos ist ein modernes konfigurierbares Betriebssystem
- einfache Konfigurierung durch GUI Unterstützung

- im Hinblick auf die **Umsetzung** der Konfigurierung gibt es aber einige **Mängel**:
  - Klassische Implementierung der Konfigurierungsentscheidungen in den Komponenten mit Hilfe von **#ifdef und Makros**
    - Schutz vor ungewollten Ersetzungen nur durch strikte Namenskonvention
    - bedingte Übersetzung macht den Code schwer verständlich, zu warten und wiederzuverwenden
  - mangelnde **Trennung der Belange**
    - viel Konfigurierungswissen ist im Quellcode verankert
    - quer schneidende Belange blähen die Funktionen auf

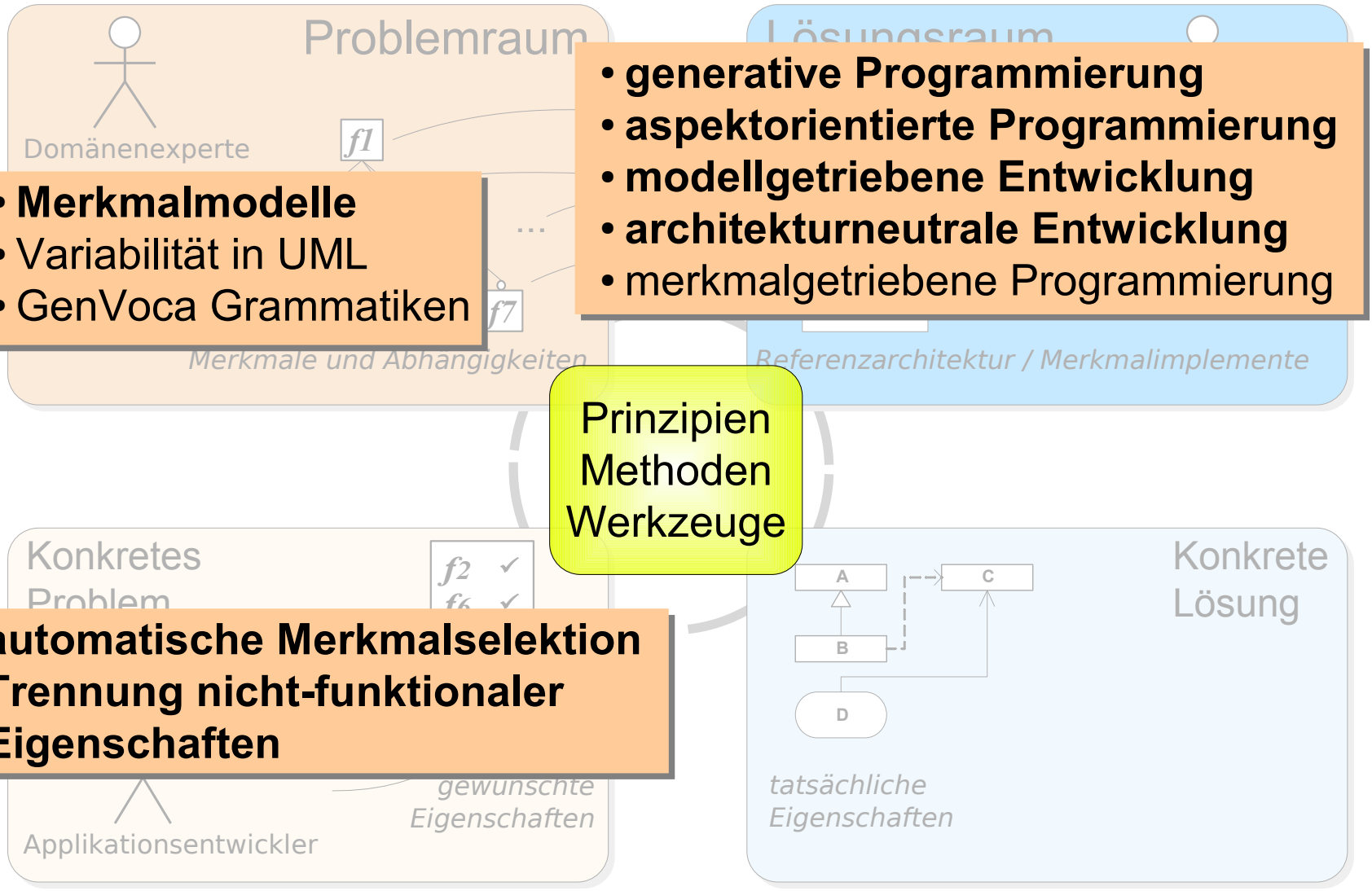
In dieser Vorlesung ...

- Warum statische Konfigurierung?
- Warum kein Linux oder Windows?
- Wie baut man Spezialzwecksoftware?
- Welche Probleme gibt es in der Praxis?
- **Welche Lösungsansätze werden verfolgt?**

Software-Produktlinien



Software-Produktlinien



- **Merkmalmodelle**
- Variabilität in UML
- GenVoca Grammatiken

- **generative Programmierung**
- **aspektorientierte Programmierung**
- **modellgetriebene Entwicklung**
- **architekturneutrale Entwicklung**
- merkmalgetriebene Programmierung

- **automatische Merkmalselektion**
- **Trennung nicht-funktionaler Eigenschaften**

Forschungsfragen

- Beschreibung der Variantenvielfalt einer „Systemsoftwarefamilie“
- Strukturierung von Systemsoftwarefamilien
- Konfigurierungstechniken
- Umgang mit „querschneidenden Belange“
- Umgang mit nicht-funktionalen Eigenschaften
- Grenzen der Konfigurierbarkeit
- Komposition von Produktlinien
- ...

Interessierte Studentinnen und Studenten erwünscht!