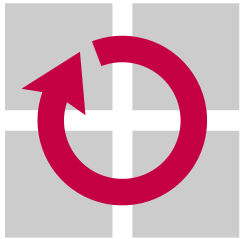
Betriebssysteme (er)fordern Softwaretechnik

Wolfgang Schröder-Preikschat

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik, Lehrstuhl Informatik 4
— Verteilte Systeme und Betriebssysteme —

www4.informatik.uni-erlangen.de



We take the view that operating systems should not be written in assembly language. (Löhr, SOSP-6, 1977)

Warum Softwaretechnik?

Definition Betriebssystem

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300)

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300) ✓

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300) ✓

Firmware

- Software in Festwertspeichern

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300) ✓

Firmware

- Software in Festwertspeichern ✗

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300) ✓

Firmware

- Software in Festwertspeichern ✗

Hardware ?

Warum Softwaretechnik?

Definition Betriebssystem

Software

- *Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.* (DIN 44300) ✓

Firmware

- Software in Festwertspeichern ✗

Hardware ?

- „Hardwarebetriebssystem“ zu googeln, kann sich lohnen. . .

Darum. . .

Betriebssystem — die ewige Baustelle: Entwurf und Implementierung

```
wosch@hawaii 37> uname -snrm
Linux hawaii 2.2.14 i686
wosch@hawaii 38> echo 'main(){printf("Hello world!\n");}' > hello.c
wosch@hawaii 39> gcc -O6 -c hello.c; gcc -static -o hello hello.o
wosch@hawaii 40> hello
Hello world!
wosch@hawaii 41> ls -l hello*
-rwxr-xr-x   1 wosch    ivs        932099 Dec  1 11:08 hello*
-rw-r--r--   1 wosch    ivs            33 Dec  1 11:06 hello.c
-rw-r--r--   1 wosch    ivs           908 Dec  1 11:07 hello.o
wosch@hawaii 42> size hello hello.o
   text    data     bss     dec     hex filename
 195562    5064    3340  203966   31cbe hello
     29       0       0      29      1d hello.o
```

Darum. . .

Betriebssystem — die ewige Baustelle: Werkzeuge

```
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_PRIORITY;
#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();
}
```

Darum. . .

Betriebssystem — die ewige Baustelle

```
wosch@hawaii 37> uname -snrm
Linux hawaii 2.2.14 i686
wosch@hawaii 38> echo 'main(){printf("Hello world!\n");}' > hello.c
wosch@hawaii 39> gcc -O6 -c hello.c; gcc -static -o hello hello.o
wosch@hawaii 40> hello
Hello world!
wosch@hawaii 41> ls -l hello*
-rwxr-xr-x   1 wosch    ivs        932099 Dec  1 11:08 hello*
-rw-r--r--   1 wosch    ivs            33 Dec  1 11:06 hello.c
-rw-r--r--   1 wosch    ivs           908 Dec  1 11:07 hello.o
wosch@hawaii 42> size hello hello.o
   text    data     bss     dec     hex filename
 195562    5064    3340  203966   31cbe hello
     29       0       0      29      1d hello.o
```

```
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_PRIORITY;
#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();
}
```

Allgemeinzwecksysteme einem Spezialzweck unterziehen zu wollen, ist selten zielführend.

Beherrschung der Variantenvielfalt ist eins der schwersten Probleme.

Inhalt

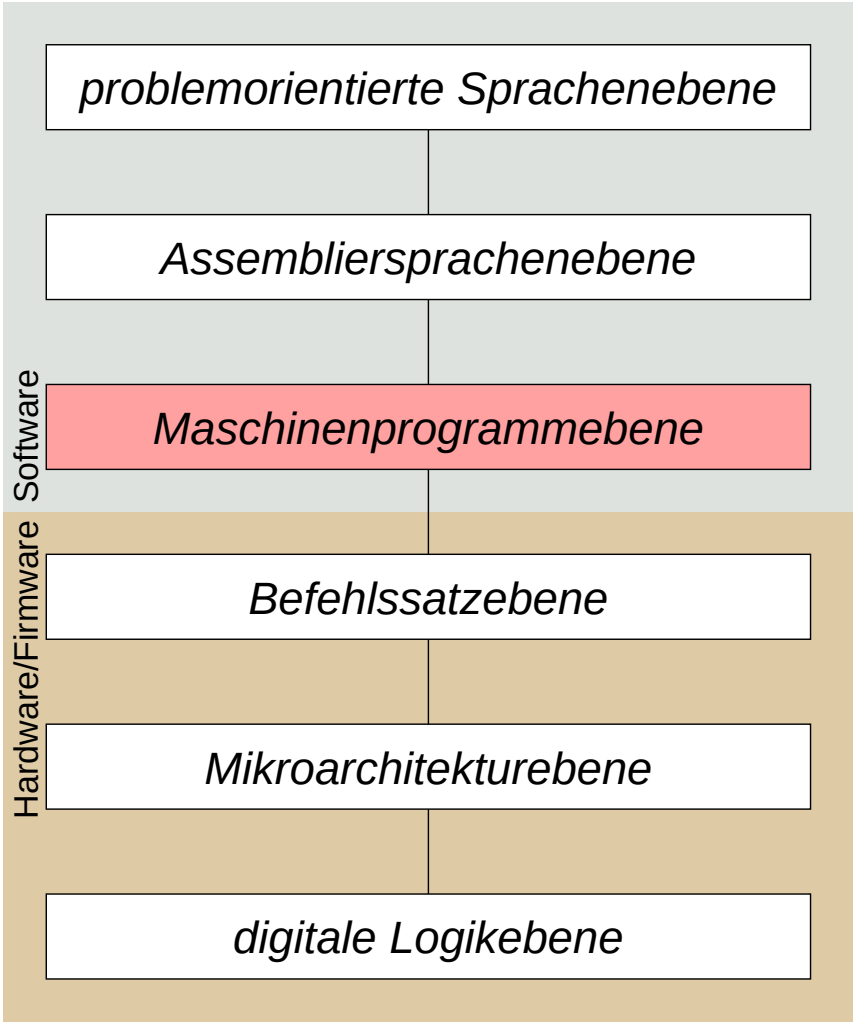
Betriebssysteme
- Abstraktion
- Anpassung

Softwaretechnik
- Programmfamilien
- Aspektorientierung

Abspann

Organisation von Rechensystemen

Betriebssystem als Interpret



Übersetzung (Kompilierer)

Übersetzung (Assemblierer) und Bindung (Binder)

partielle Interpretation (Betriebssystem)

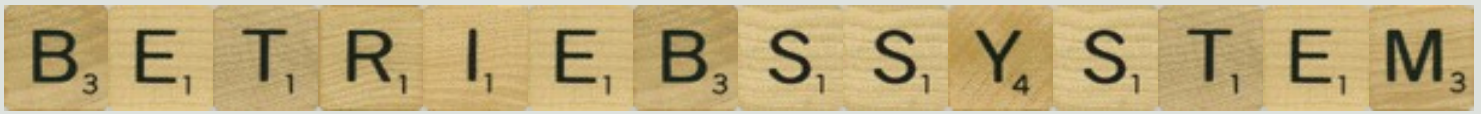
Interpretation (Mikroprogramm) oder Ausführung

Ausführung

Abstrakte Maschine

Maschinenprogrammebene

Anwendung(en)

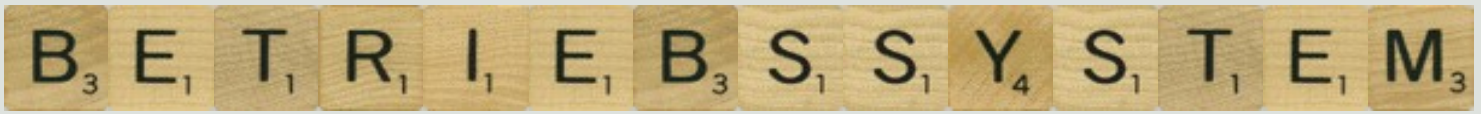


Abstrakte Maschine

Maschinenprogrammebene

Plattformabhängigkeit?

Anwendung(en)



Abschottung gegen Hardware

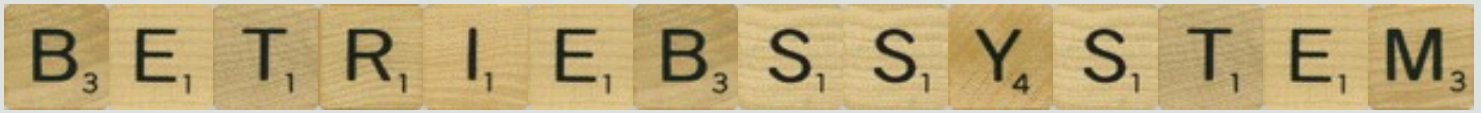
Hardware Abstraction Layer, HAL

Abschottung gegen Hardware

Hardware Abstraction Layer, HAL

Anwendbarkeit?

Anwendung(en)



HAL

Abschottung gegen Anwendungen

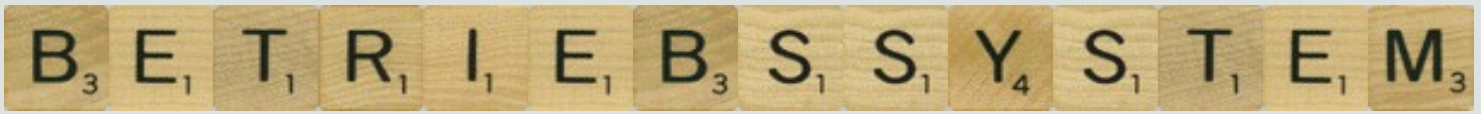
Application Abstraction Layer, AAL

Abschottung gegen Anwendungen

Weltformel „Universelle Schnittstelle“

Anwendung(en)

¿AAL?



HAL

Abschottung gegen Anwendungen (Forts.)

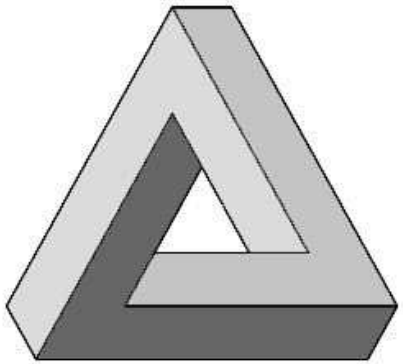
Betriebssystemdilemma — Software zwischen Baum und Borke

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 easily be identified; a common criticism of many systems is that, in attempting to be all things to all individuals, they end up being totally satisfactory to no-one. (Lister, 1975)

Universalität ist Illusion

Betriebssystem $\neq$ Eier legende Wollmilchsau

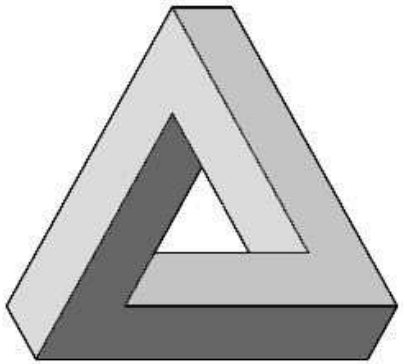
Genauso, wie wir uns in all den Jahrzehnten nicht auf die universelle Programmiersprache haben festlegen können, genauso können wir uns auch nicht auf die universelle Betriebssystemschnittstelle verständigen.



Universalität ist Illusion

Betriebssystem $\neq$ Eier legende Wollmilchsau

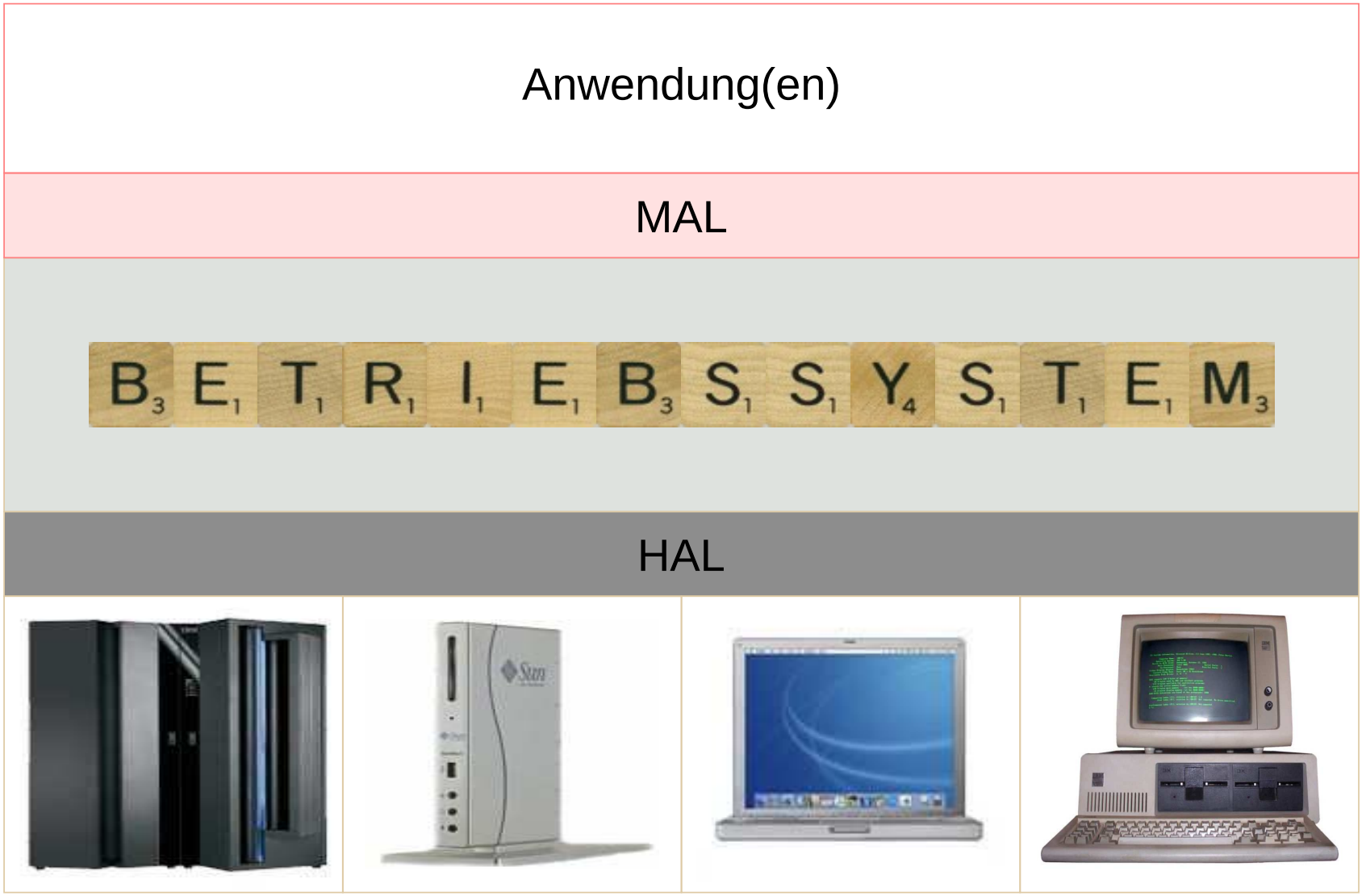
Genauso, wie wir uns in all den Jahrzehnten nicht auf die universelle Programmiersprache haben festlegen können, genauso können wir uns auch nicht auf die universelle Betriebssystemschnittstelle verständigen.



- wir haben aber gelernt, vom Betriebssystem zu abstrahieren

Abschottung gegen Betriebssysteme — „Middleware“

Machine Abstraction Layer, MAL



Abstrakte Maschine (Forts.)

Maschinenprogrammebene für verteilte Systeme

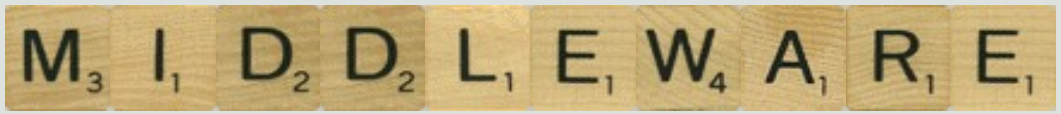
Abstrakte Maschine (Forts.)

Und das Spiel beginnt von vorne. . .

Anwendbarkeit?

Plattformabhängigkeit?

Anwendung(en)



Ausprägungen von Abhängigkeit

Querschneidende Belange (engl. *cross-cutting concerns*)

Abschottung *à la* [HAM]AL ist „leicht“ in funktionaler Hinsicht:

- Elementaroperation, Schicht
- abstrakter Datentyp, Modul, Klasse, . . . , Komponente

Ausprägungen von Abhängigkeit

Querschneidende Belange (engl. *cross-cutting concerns*)

Abschottung *à la* [HAM]AL ist „leicht“ in funktionaler Hinsicht:

- Elementaroperation, Schicht
- abstrakter Datentyp, Modul, Klasse, . . . , Komponente

Eigenschaften nicht-funktionaler Art zu kapseln, ist „schwieriger“:

- die (möglichen) Auswirkungen der benutzten Funktionen etwa aus räum-, zeitlichen und qualitativen Gesichtspunkten
- auf benutzender Ebene sodann getroffene implizite Annahmen

Ausprägungen von Abhängigkeit

Querschneidende Belange (engl. *cross-cutting concerns*)

Abschottung *à la* [HAM]AL ist „leicht“ in funktionaler Hinsicht:

- Elementaroperation, Schicht
- abstrakter Datentyp, Modul, Klasse, . . . , Komponente

Eigenschaften nicht-funktionaler Art zu kapseln, ist „schwieriger“:

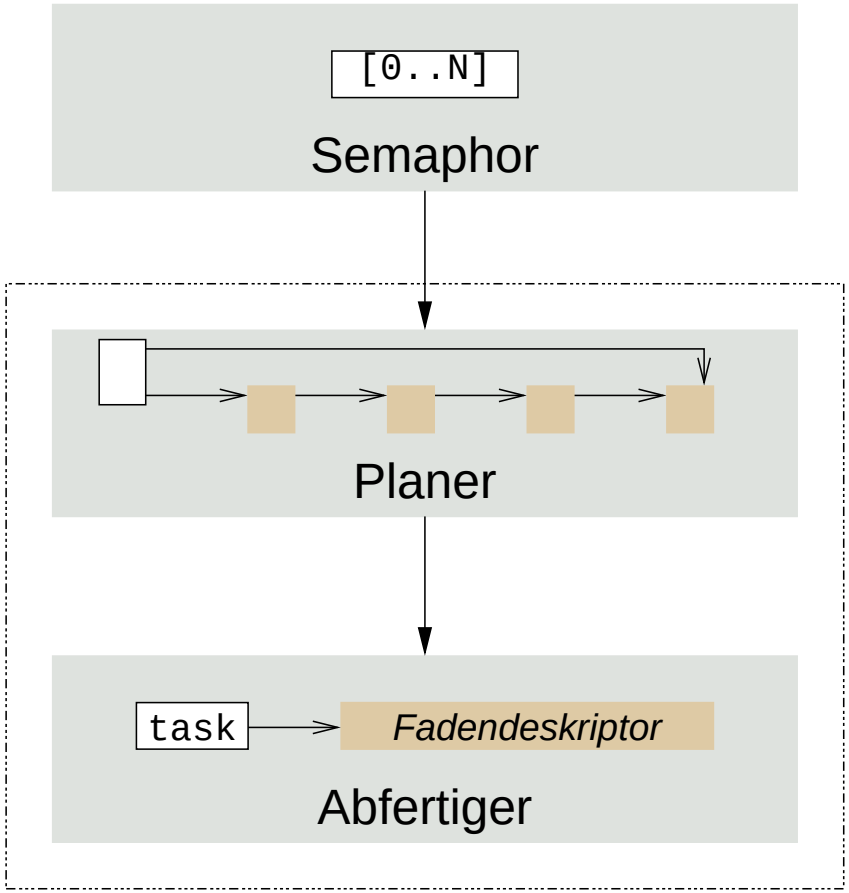
- die (möglichen) Auswirkungen der benutzten Funktionen etwa aus räum-, zeitlichen und qualitativen Gesichtspunkten
- auf benutzender Ebene sodann getroffene implizite Annahmen

☞ solche Belange sind (oft) kein Einzelfall auf höheren Ebenen

A well structured system can easily be understood and modified. (Goullon, Isle & Löhr, ICSE-3, 1978)

Fadenablauf — kooperativ

Funktionale Hierarchie von Semaphor (optional), Planer und Abfertiger



Signalisierung $\sim$ Dijkstra

- nicht-negative ganze Zahl
- keine Laufgefahr

Einplanung $\mapsto$ FCFS

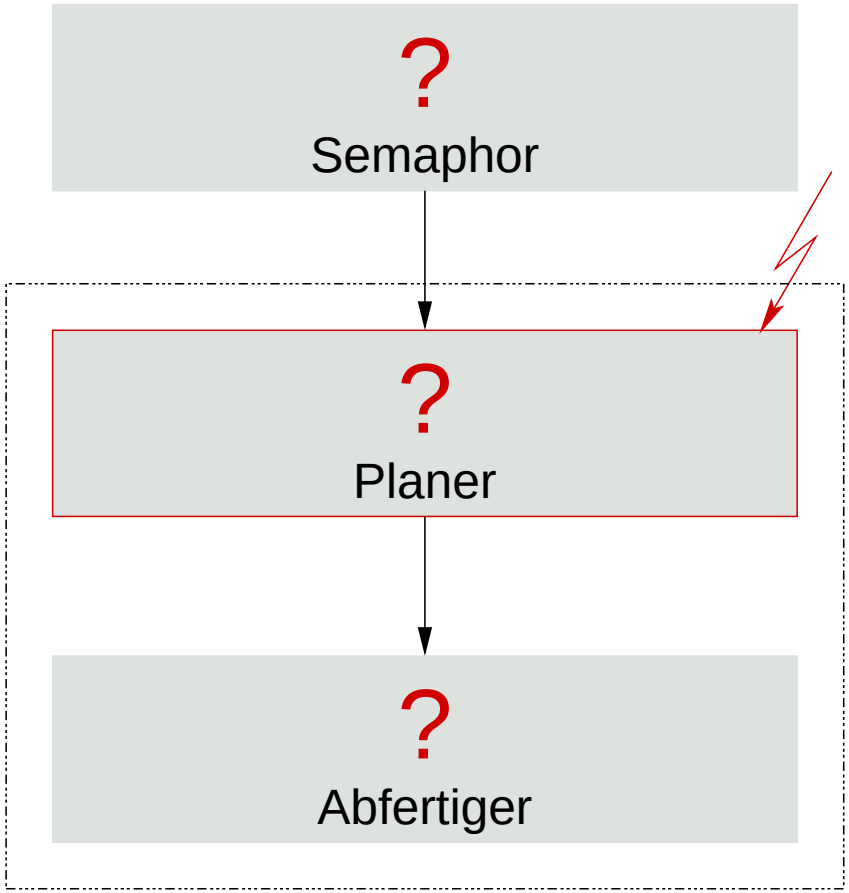
- Bereitliste
- keine Laufgefahr

Einlastung $\mapsto$ Koroutinenkonzept

- Instanzenvariable
- keine Laufgefahr

Fadenablauf — verdrängend

Funktionale Hierarchie von Semaphor (optional), Planer und Abfertiger



Signalisierung ∼ Dijkstra

- Zahl verändern und abfragen
- bedingte Fadensteuerung

Einplanung ↦ RR

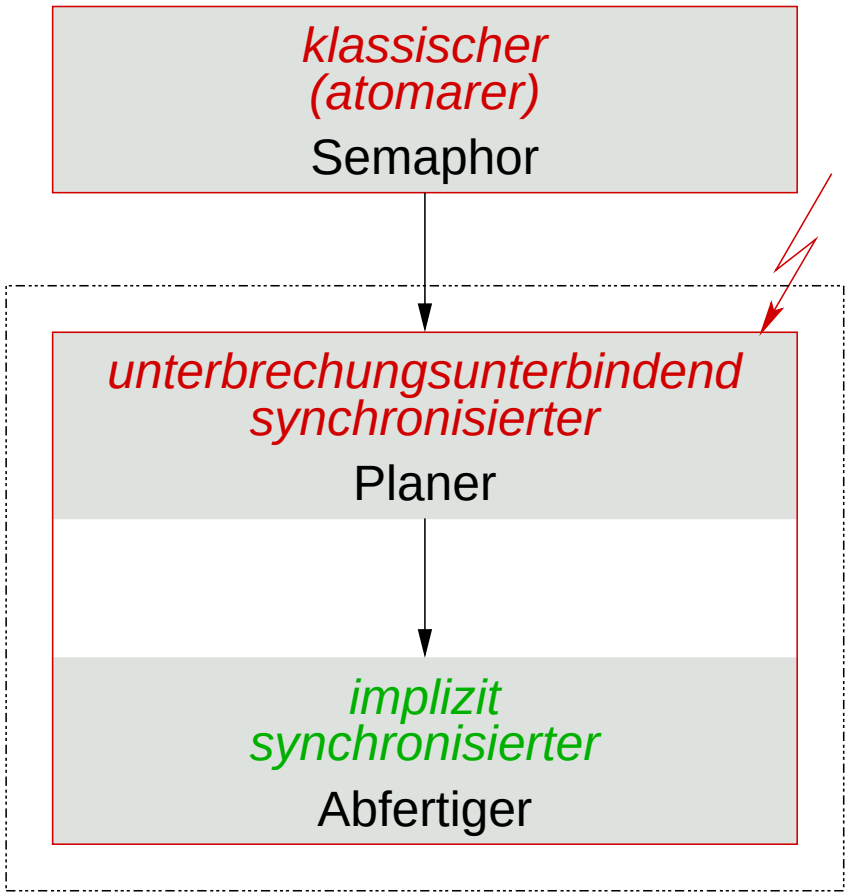
- *Interrupt*
- Fadenauswahl treffen

Einlastung ↦ Koroutinenkonzept

- Instanzenvariable umsetzen
- Stapel umschalten

Fadenablauf — verdrängend

Funktionale Hierarchie von Semaphor (optional), Planer und Abfertiger



Signalisierung $\mapsto$ Dijkstra

- kritischer Abschnitt
- explizit zu synchronisieren

Einplanung $\mapsto$ RR

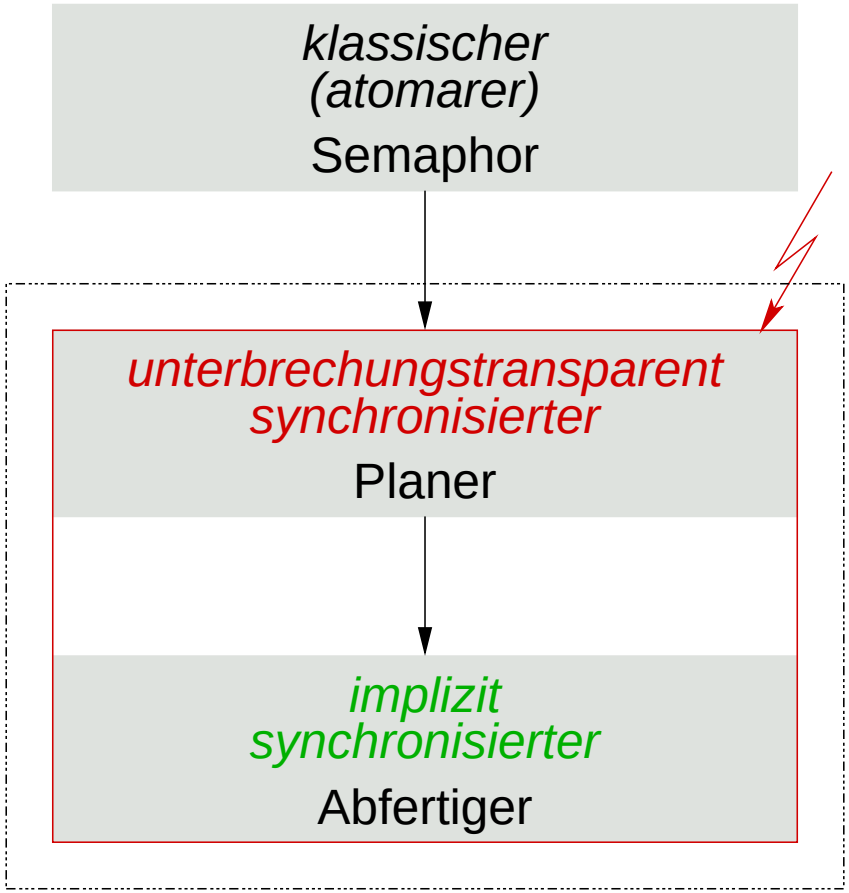
- kritischer Abschnitt
- Unterbrechungssperre (hart)

Einlastung $\mapsto$ Koroutinenkonzept

- kritischer Abschnitt
- keine Laufgefahr

Fadenablauf — verdrängend

Funktionale Hierarchie von Semaphor (optional), Planer und Abfertiger



Signalisierung $\mapsto$ Dijkstra

- kritischer Abschnitt
- explizit zu synchronisieren

Einplanung $\mapsto$ RR

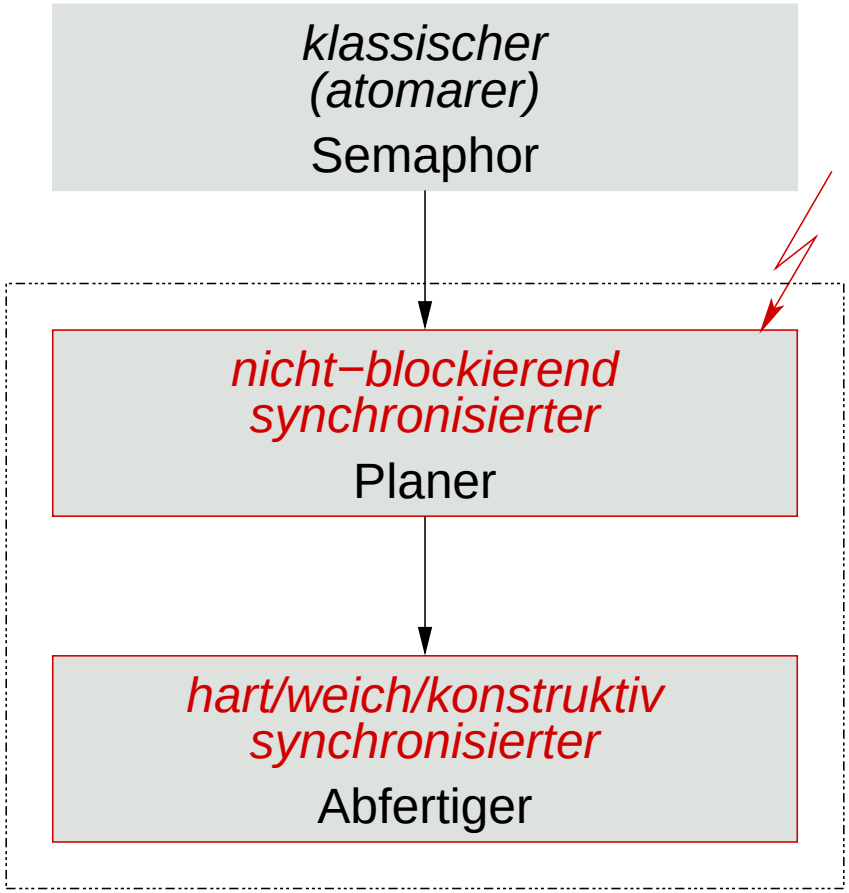
- kritischer Abschnitt
- Fortsetzungssperre (weich)

Einlastung $\mapsto$ Koroutinenkonzept

- kritischer Abschnitt
- keine Laufgefahr

Fadenablauf — verdrängend

Funktionale Hierarchie von Semaphor (optional), Planer und Abfertiger



Signalisierung $\mapsto$ Dijkstra

- kritischer Abschnitt
- explizit zu synchronisieren

Einplanung $\mapsto$ RR

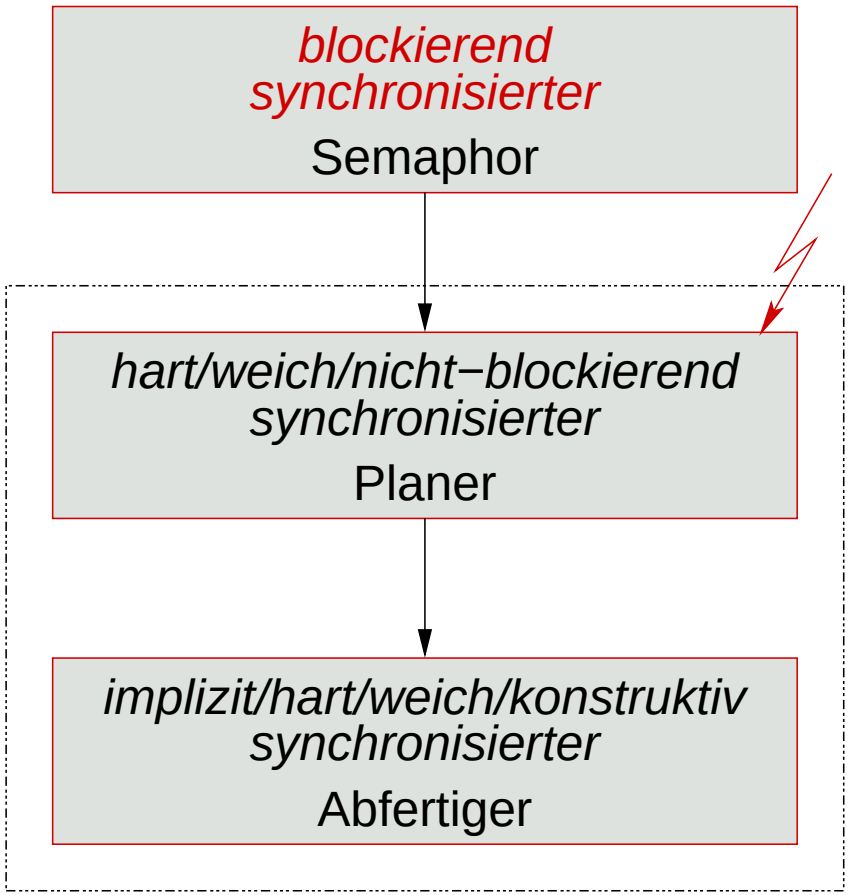
- kritischer Abschnitt
- CAS, LL/SC

Einlastung $\mapsto$ Koroutinenkonzept

- kritischer Abschnitt
- explizit zu synchronisieren

Synchronisation im Semaphor

Zusammenspiel zwischen Semaphor und Planer/Abfertiger

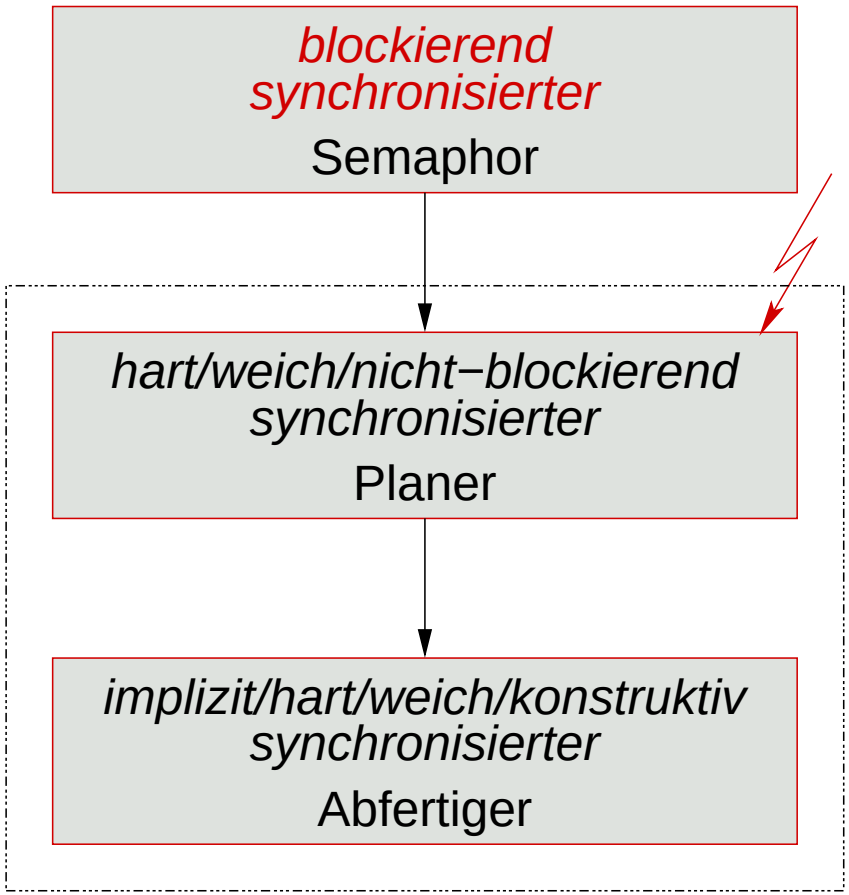


gegenseitiger Ausschluss

- verbietet die Verwendung von V zur Signalisierung bei der Unterbrechungsbehandlung

Synchronisation im Semaphor

Zusammenspiel zwischen Semaphor und Planer/Abfertiger

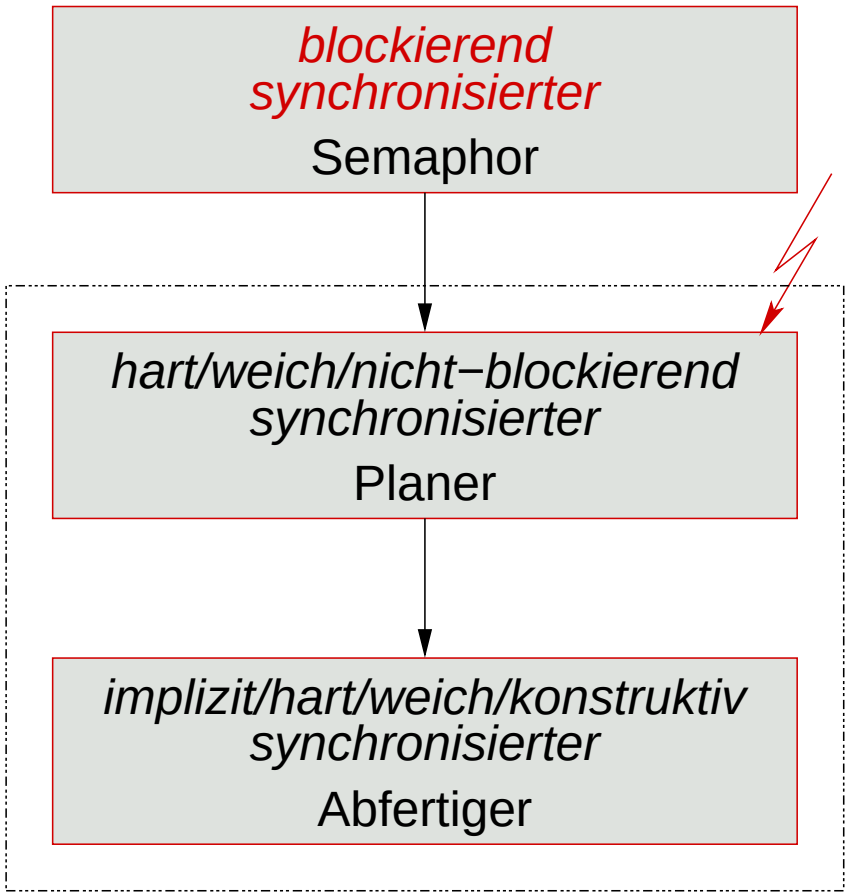


gegenseitiger Ausschluss

- verbietet die Verwendung von V zur Signalisierung bei der Unterbrechungsbehandlung
- zyklische Benutztbeziehung

Synchronisation im Semaphor

Zusammenspiel zwischen Semaphor und Planer/Abfertiger



gegenseitiger Ausschluss

- verbietet die Verwendung von V zur Signalisierung bei der Unterbrechungsbehandlung
- zyklische Benutztbeziehung
  - der Semaphor hängt ab von einer seiteneffektfreien Unterbrechungsbehandlung
  - die Unterbrechungsbehandlung von einer nicht-blockierenden Signalisierung

Synchronisation im Semaphor

Einseitige Synchronisation

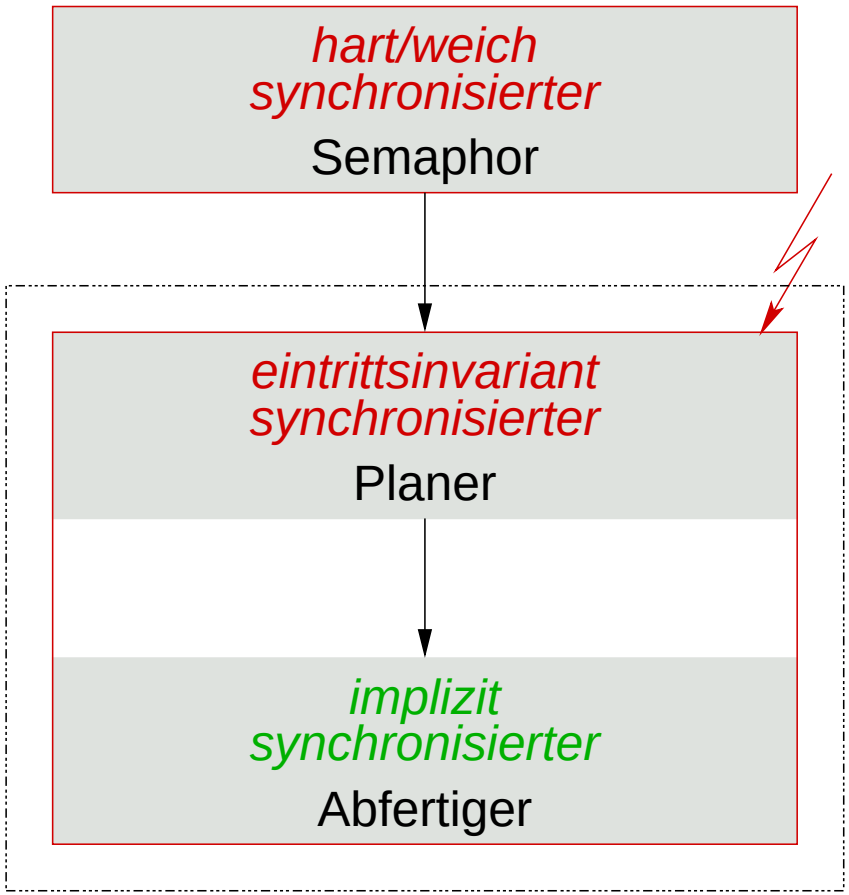
hart/weich synchronisierter
Semaphor

Signalisierung

hart		weich	
enter	*leave*	*enter*	*leave*
`cli`	`sti`	`f=1`	`f=0`

Synchronisation im Semaphor

Auswirkungen auf den Planer



Signalisierung

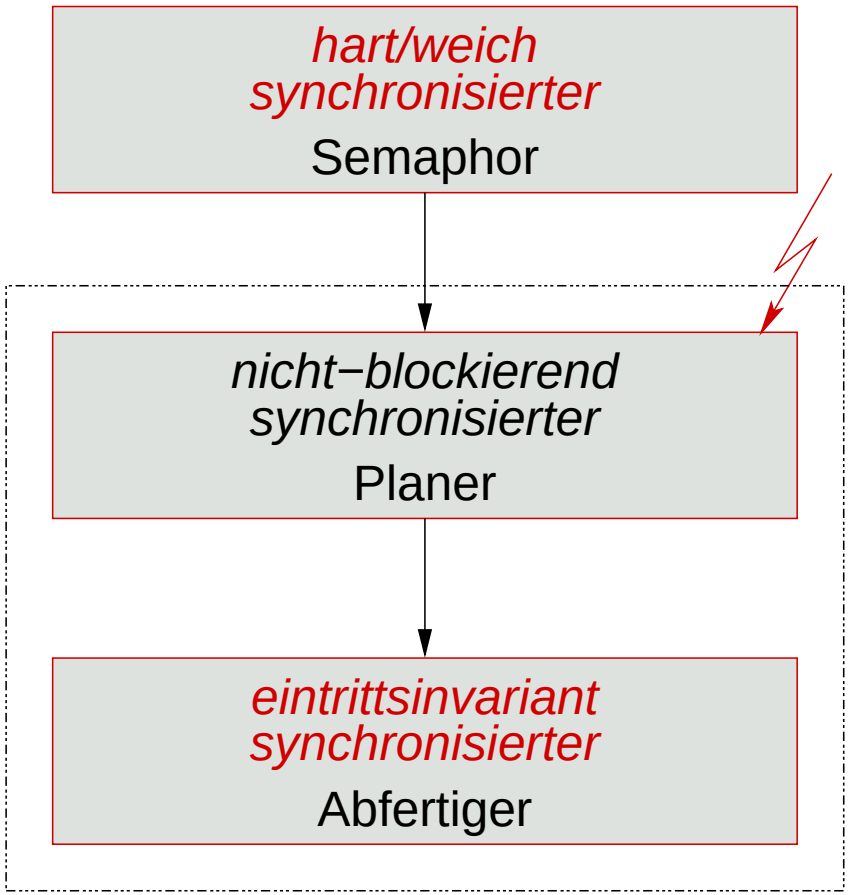
hart		weich	
enter	*leave*	*enter*	*leave*
`cli`	`sti`	`f=1`	`f=0`

Einplanung (frei kombinierbar)

hart		weich	
enter	*leave*	*enter*	*leave*
`pushf` `cli`	`popf`	`f++`	`f--`

Synchronisation im Semaphor

Auswirkungen auf den Abfertiger, je nach Synchronisation im Planer



Signalisierung

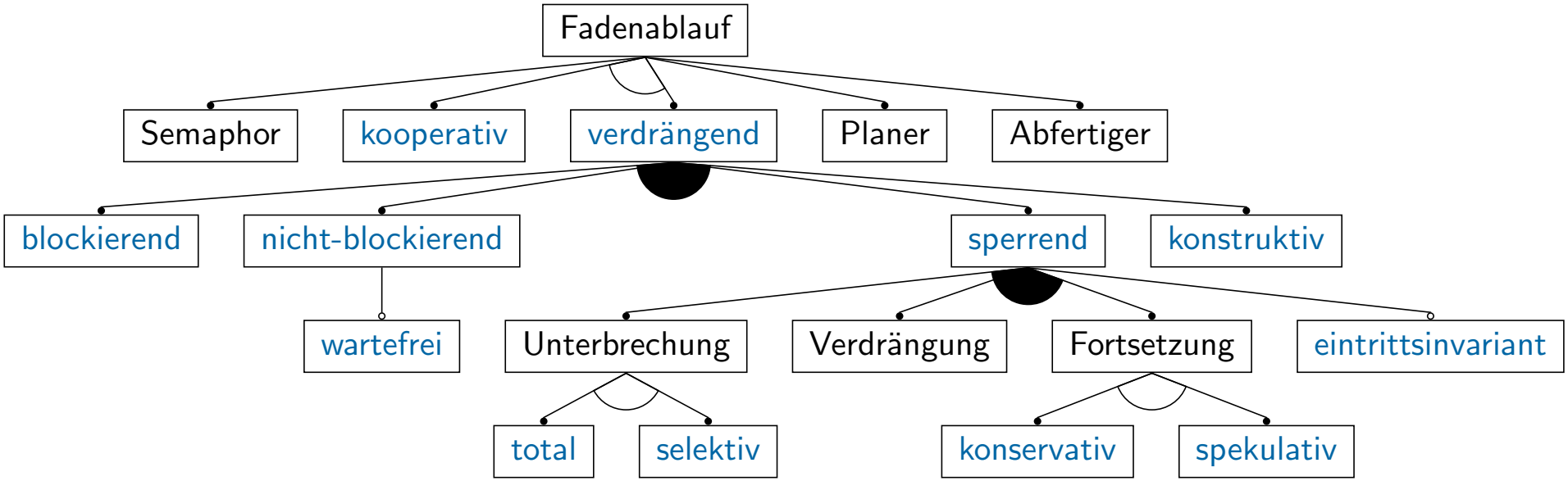
hart		weich	
enter	*leave*	*enter*	*leave*
`cli`	`sti`	`f=1`	`f=0`

Einlastung (frei kombinierbar)

hart		weich	
enter	*leave*	*enter*	*leave*
`pushf`	`popf`	`f++`	`f--`
`cli`			

Fadenablauf — Synchronisationseigenschaften

Zusammenfassung als Merkmaldiagramm



- jede Variante ist sinnvoll in bestimmten Anwendungsfällen
- freie Kombinierbarkeit scheidet in manchen Fällen aus
- Varianten bedingen andere Varianten oder schließen diese aus
- Variantenauswahl ist kaum eine Frage in funktionaler Hinsicht

Intermezzo

Die Lehre aus der Geschichte: „Unterbrechungsgesteuerte Systeme sind nicht einfach“

Nebenläufigkeit

Intermezzo

Die Lehre aus der Geschichte: „Unterbrechungsgesteuerte Systeme sind nicht einfach“

Nebenläufigkeit

- ist eine nicht-funktionale Eigenschaft im Betriebssystem
  - Verdrängung von Fäden auf tieferer Ebene ermöglicht Laufgefahr (engl. *race hazard*) auf höheren Ebenen
  - je nach „Funktion“ der höheren Ebenen pflanzt sich die Laufgefahr möglicherweise fort

Intermezzo

Die Lehre aus der Geschichte: „Unterbrechungsgesteuerte Systeme sind nicht einfach“

Nebenläufigkeit

- ist eine nicht-funktionale Eigenschaft im Betriebssystem
  - Verdrängung von Fäden auf tieferer Ebene ermöglicht Laufgefahr (engl. *race hazard*) auf höheren Ebenen
  - je nach „Funktion“ der höheren Ebenen pflanzt sich die Laufgefahr möglicherweise fort
- zu beherrschen ist ein querschneidender Belang des Systems
  - ggf. identische Synchronisationsanweisungen sind an vielen (voneinander unabhängigen) Programmstellen zu platzieren
  - Implementierungsvarianten sind die Folge, die auch die Frage der Kombinierbarkeit nach sich ziehen

Intermezzo

Die Lehre aus der Geschichte: „Unterbrechungsgesteuerte Systeme sind nicht einfach“

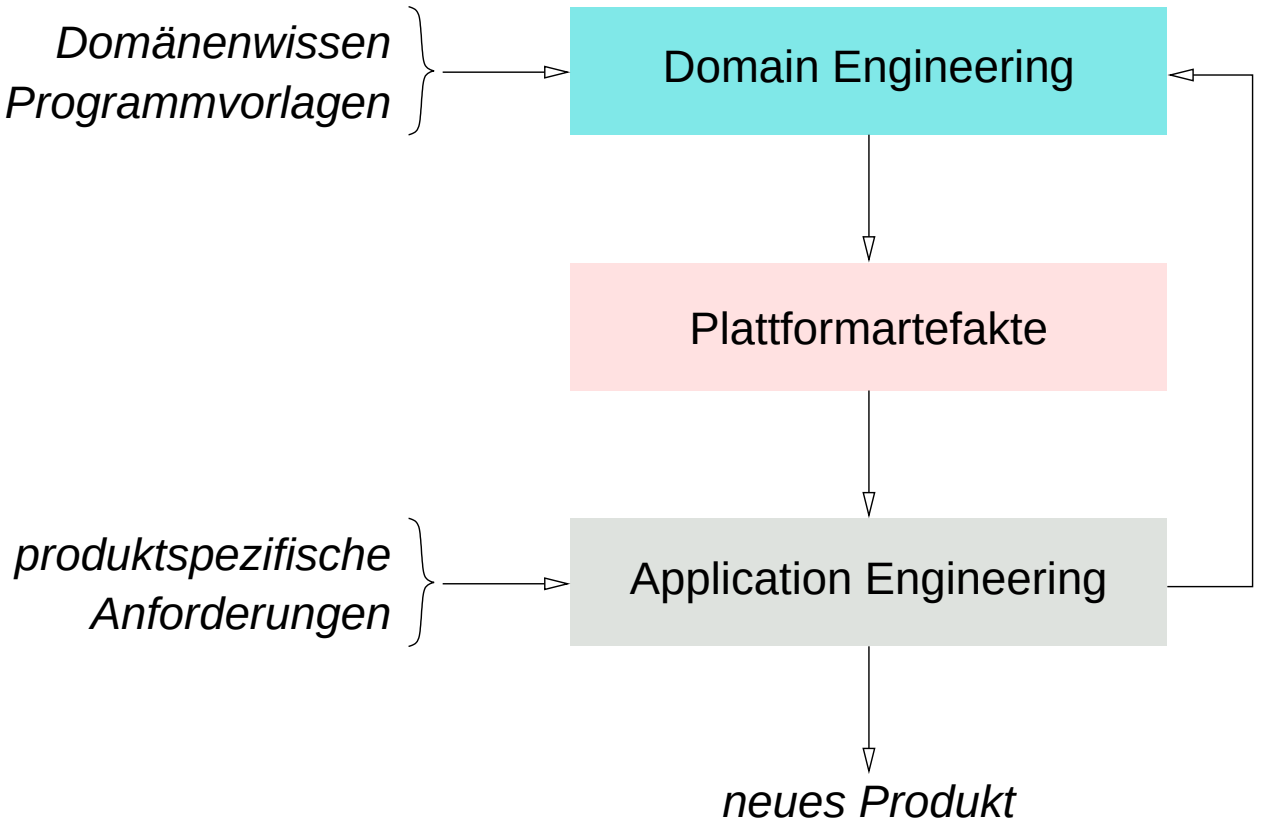
Nebenläufigkeit

- ist eine nicht-funktionale Eigenschaft im Betriebssystem
  - Verdrängung von Fäden auf tieferer Ebene ermöglicht Laufgefahr (engl. *race hazard*) auf höheren Ebenen
  - je nach „Funktion“ der höheren Ebenen pflanzt sich die Laufgefahr möglicherweise fort
- zu beherrschen ist ein querschneidender Belang des Systems
  - ggf. identische Synchronisationsanweisungen sind an vielen (voneinander unabhängigen) Programmstellen zu platzieren
  - Implementierungsvarianten sind die Folge, die auch die Frage der Kombinierbarkeit nach sich ziehen
- stellt bei weitem nicht die einzige Eigenschaft dieser Art dar
  - Verlässlichkeit, Rechtzeitigkeit, Dienstgüte, . . . , Architektur

With the increasing demand for special purpose system software on small computers, the need for simple and reliable production of this software becomes obvious. (Löhr, SOSP-6, 1977)

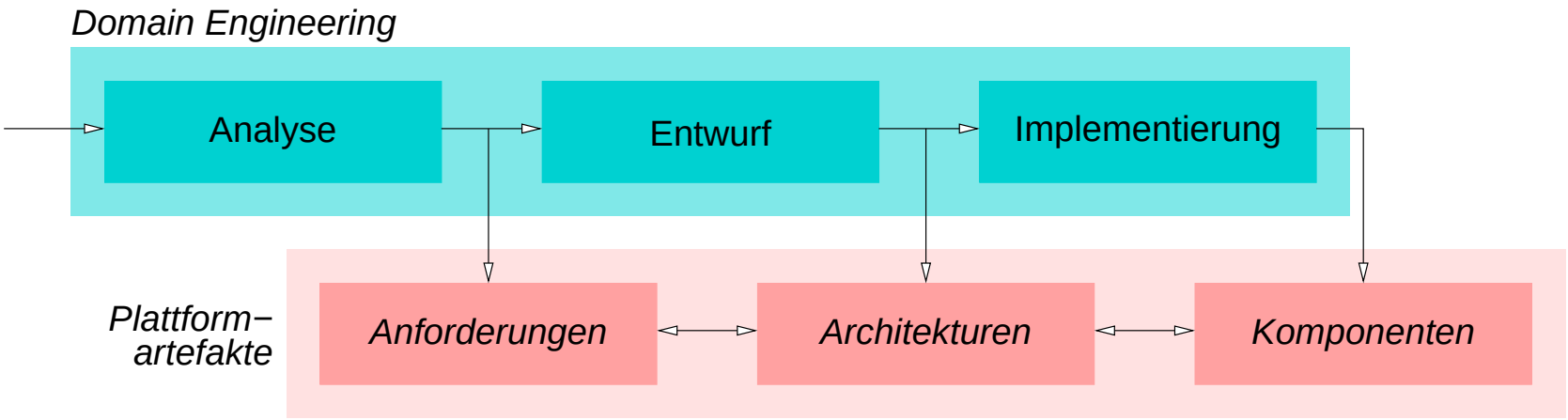
Softwareproduktlinie $\mapsto$ Betriebssystemproduktlinie

Anforderungsspezifische Fertigung von Systemvarianten



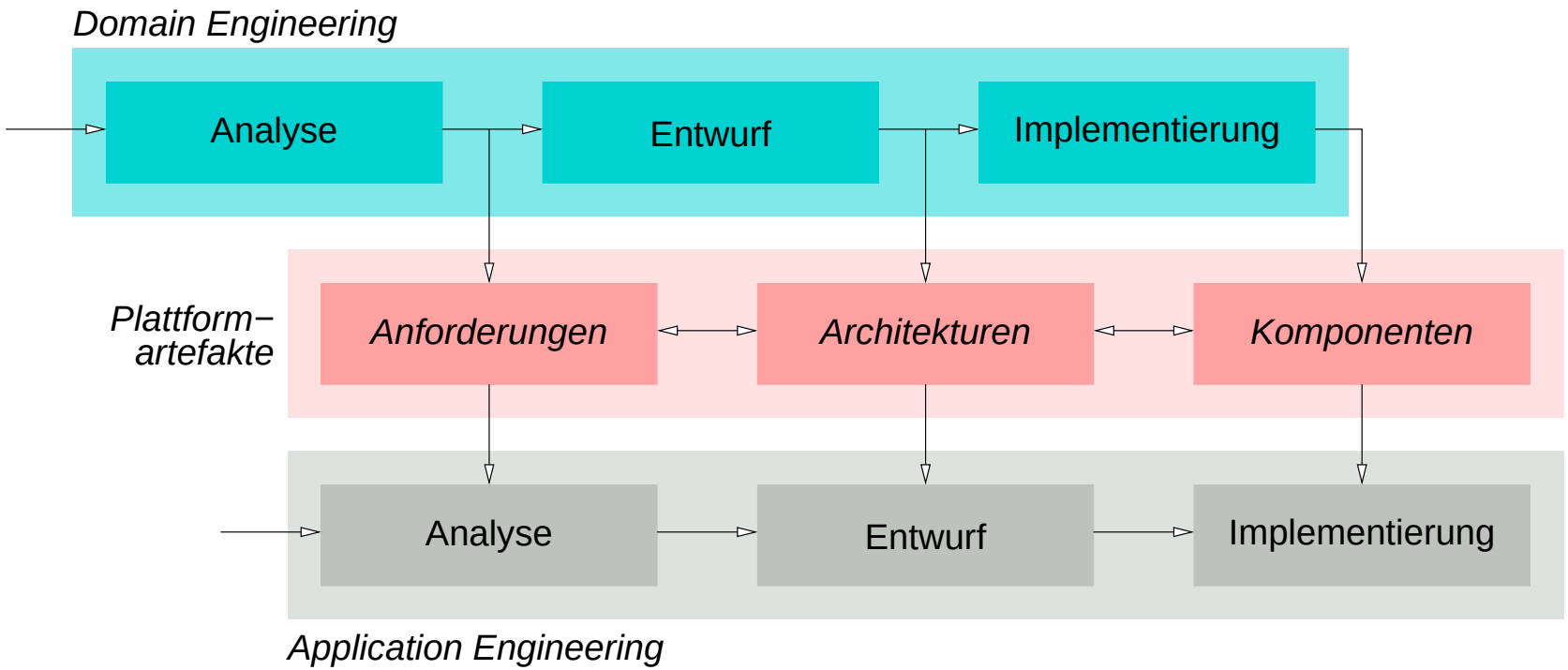
Organisierte Wiederverwendung

Referenzprozess zur Entwicklung von Softwareproduktlinien



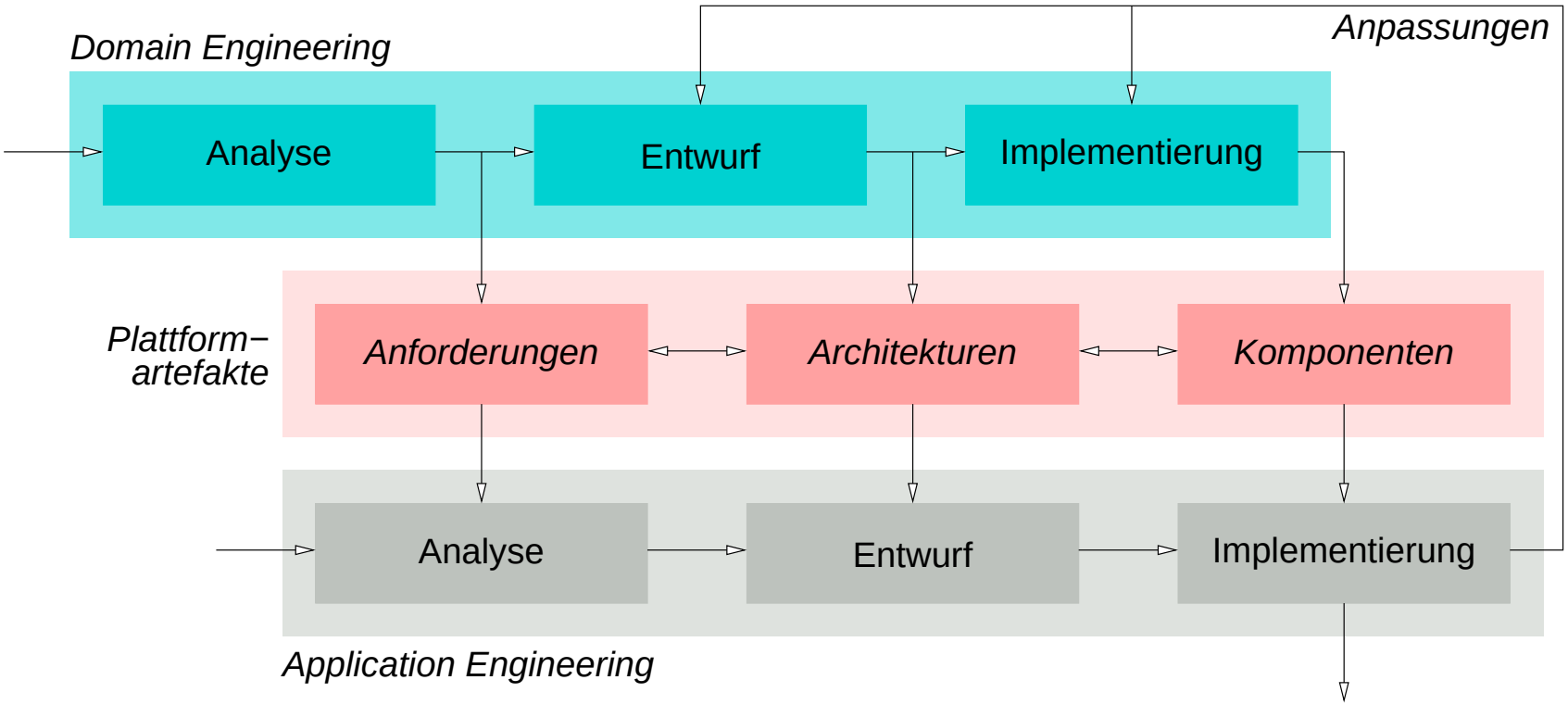
Organisierte Wiederverwendung

Referenzprozess zur Entwicklung von Softwareproduktlinien



Organisierte Wiederverwendung

Referenzprozess zur Entwicklung von Softwareproduktlinien



Domänenanalyse

Konzentration auf das Wesentliche. . .

-abgrenzung (engl. *domain scoping*)

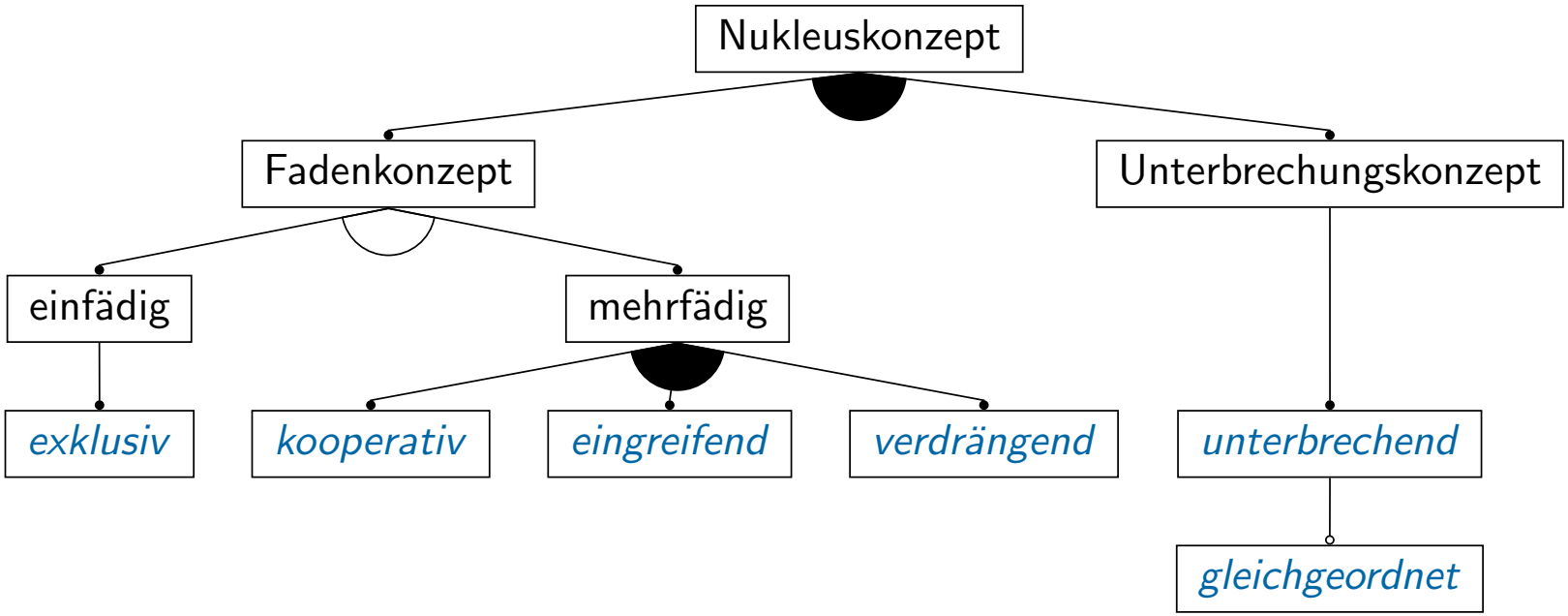
- Auswahl und Aufbereitung von Wissen
- an „erfolgversprechende Wiederverwendbarkeit“ orientieren

-modellierung (engl. *domain modeling*)

- Auswertung der Wissensbasis, Taxonomie(n) aufbauen
- ein Domänenmodell als Ergebnis liefern
  - Definition, Lexikon, Konzeptmodelle, Merkmalmodelle

Merkmalmodell $\rightsquigarrow$ Merkmaldiagramm

Gemeinsame und variable Merkmale der Nukleusinstanzen von PURE



Merkmaltypen

pure::variants

verbindlich	vorhanden bei Auswahl des übergeordneten Merkmals
optional	kann vorhanden sein
alternativ	genau eins von sich gegenseitig ausschließenden Merkmalen
oder	nicht-leere Menge beliebiger Merkmalkombinationen

Querschneidender Belang $\mapsto$ Aspekt

Aspektorientierte Programmierung (AOP)

> *Aspect-Oriented Programming is Quantification and Obliviousness.*
> *(Filman & Friedman, OOPSLA, 2000)*

Quantifizierung

- ein Aspekt wirkt auf viele Komponente

Vergesslichkeit

- die Komponenten müssen für die Einwirkung durch Aspekte nicht vorbereitet werden

Querschneidender Belang $\mapsto$ Aspekt

Aspektorientierte Programmierung (AOP)

> *Aspect-Oriented Programming is Quantification and Obliviousness.*
> *(Filman & Friedman, OOPSLA, 2000)*

Quantifizierung

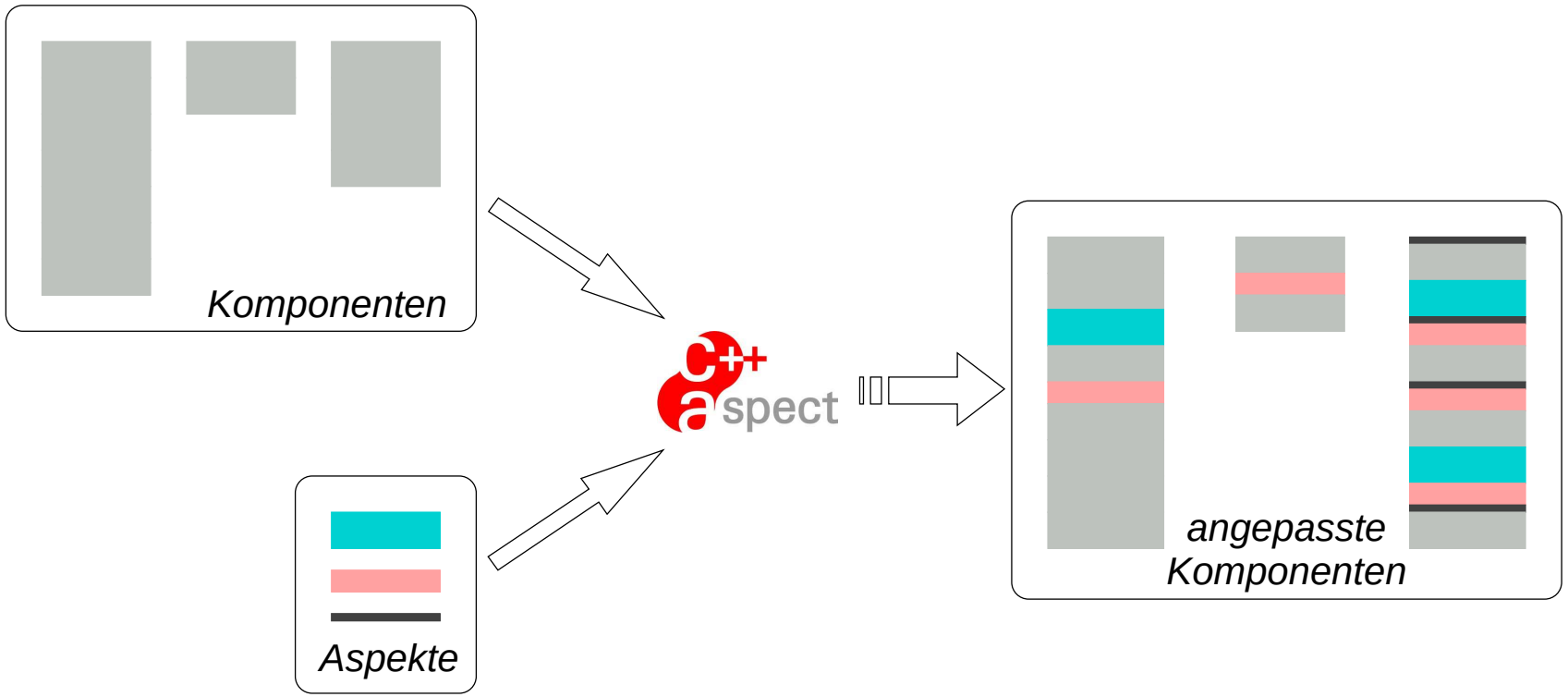
- ein Aspekt wirkt auf viele Komponente

Vergesslichkeit ist umstritten...

- die Komponenten müssen für die Einwirkung durch Aspekte nicht vorbereitet werden

Sprach- und Werkzeugunterstützung

Aspektprogramme interpretiert von einem Aspektweber

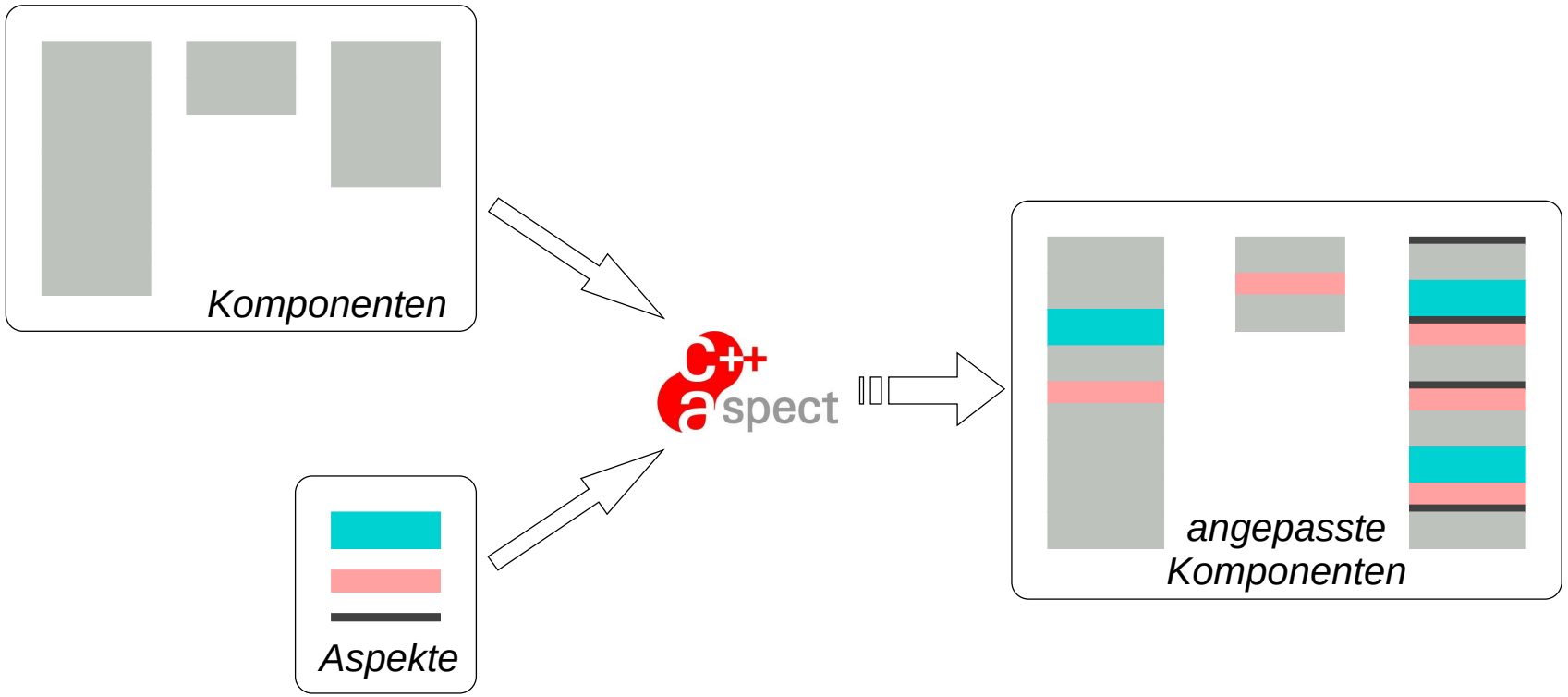


AOP

- erlaubt die Modularisierung querschneidender Belange

Sprach- und Werkzeugunterstützung

Aspektprogramme interpretiert von einem Aspektweber



AOP

- erlaubt die Modularisierung querschneidender Belange
- sollte jedoch sehr mit Bedacht verwendet werden...

Zusammenfassung

Betriebssysteme sind (je nach Zweck) komplexe Softwaresysteme

- mit harten Anforderungen an Entwurf und Implementierung. . .

Zusammenfassung

Betriebssysteme sind (je nach Zweck) komplexe Softwaresysteme

- mit harten Anforderungen an Entwurf und Implementierung. . .

Softwaretechnik tut Betriebssystemen (nach wie vor) gut

- organisierte Wiederverwendung in Form einer Produktlinie
- Modularisierung querschneidender Belange
- Merkmaldiagramme zur automatisierten Konfigurierung
- Nacharbeitung vorgefertigter Komponenten durch AOP