

Embedded Systems / Eingebettete Systeme

BSc-Studiengang Informatik
Campus Minden

Matthias König



FH Bielefeld
University of
Applied Sciences

Beispiel einer Anwendung: Funkmelder

- Mobiler Funkmeldeempfänger (Pager, Pieper...)
- Verschiedene Systeme je nach Anwendung hinsichtlich:
 - Funkfrequenz
 - Reichweite
 - Alarmierung

[Quelle: Wikipedia, Funkmeldeempfänger]



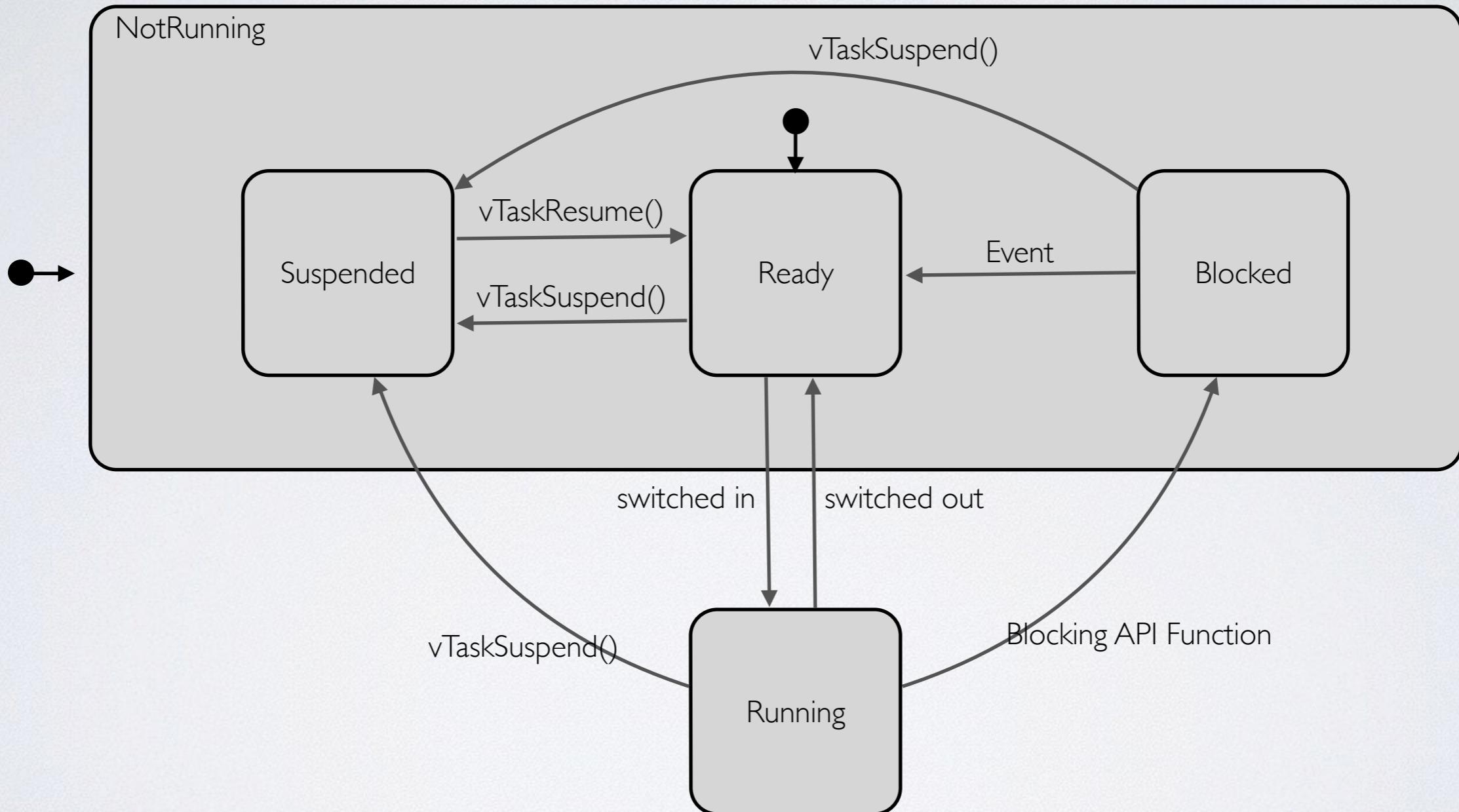
Beispiel eines Gästerufsystems

Wiederholung

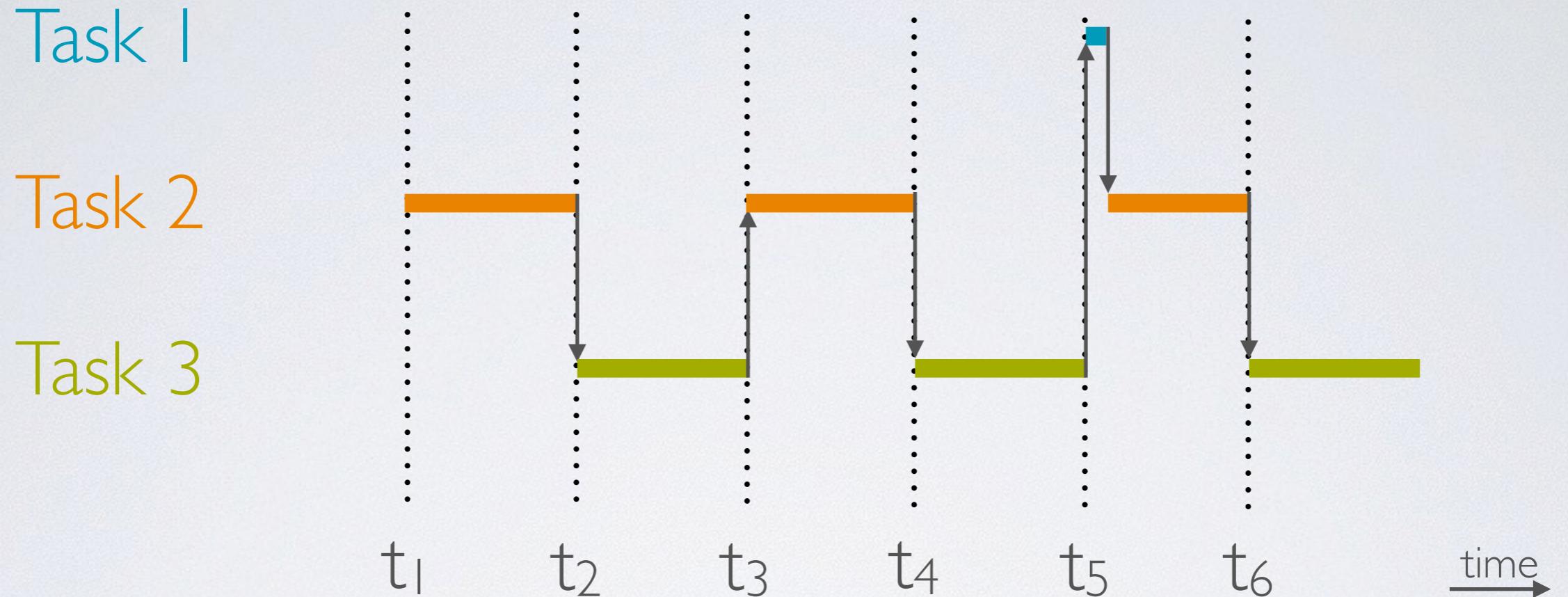
Beispiel RTOS: FreeRTOS

- Präemptives und kooperatives Scheduling
- Flexibles Management von Taskprioritäten
- Queues
- Semaphoren und Mutexes
- Tick und Idle Hook Funktionen
- Stack Overflow checking

FreeRTOS: Task-Zustände



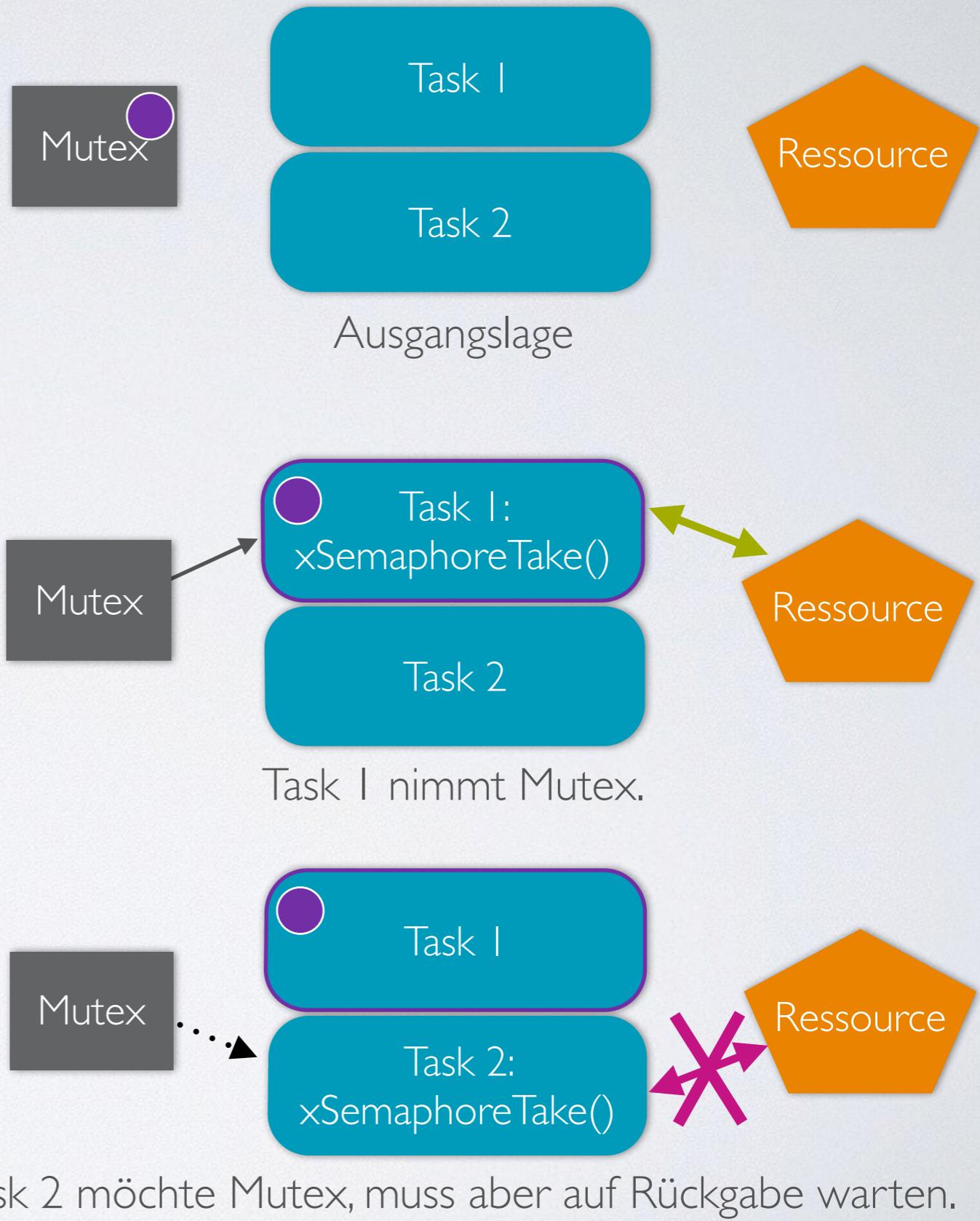
FreeRTOS: Scheduling (Example 4)



- Task 1 mit höherer Priorität als Task 2 und Task 3
- Task 1 wird periodisch aufgerufen; Hier Time Slice = 1 ms.
- Achtung: Kernel Overhead Prozess ist nicht dargestellt.

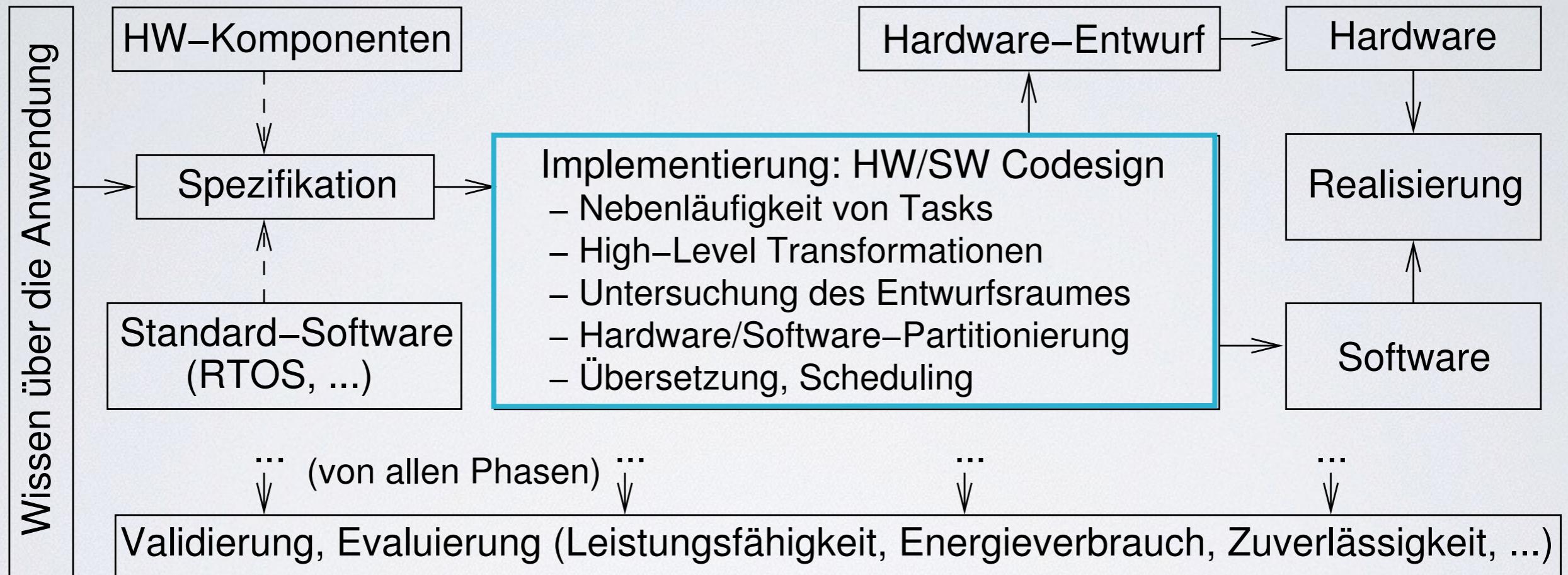
FreeRTOS: Mutexes

- Mutex (Mutual Exclusion)
- Spezielle binäre Semaphore
- Nehmen und Rückgeben von Token
- Token regelt Zugriff auf gemeinsame Ressource.
- Mit Prioritätsvererbung



Implementierung

Hardware/Software Codedesign

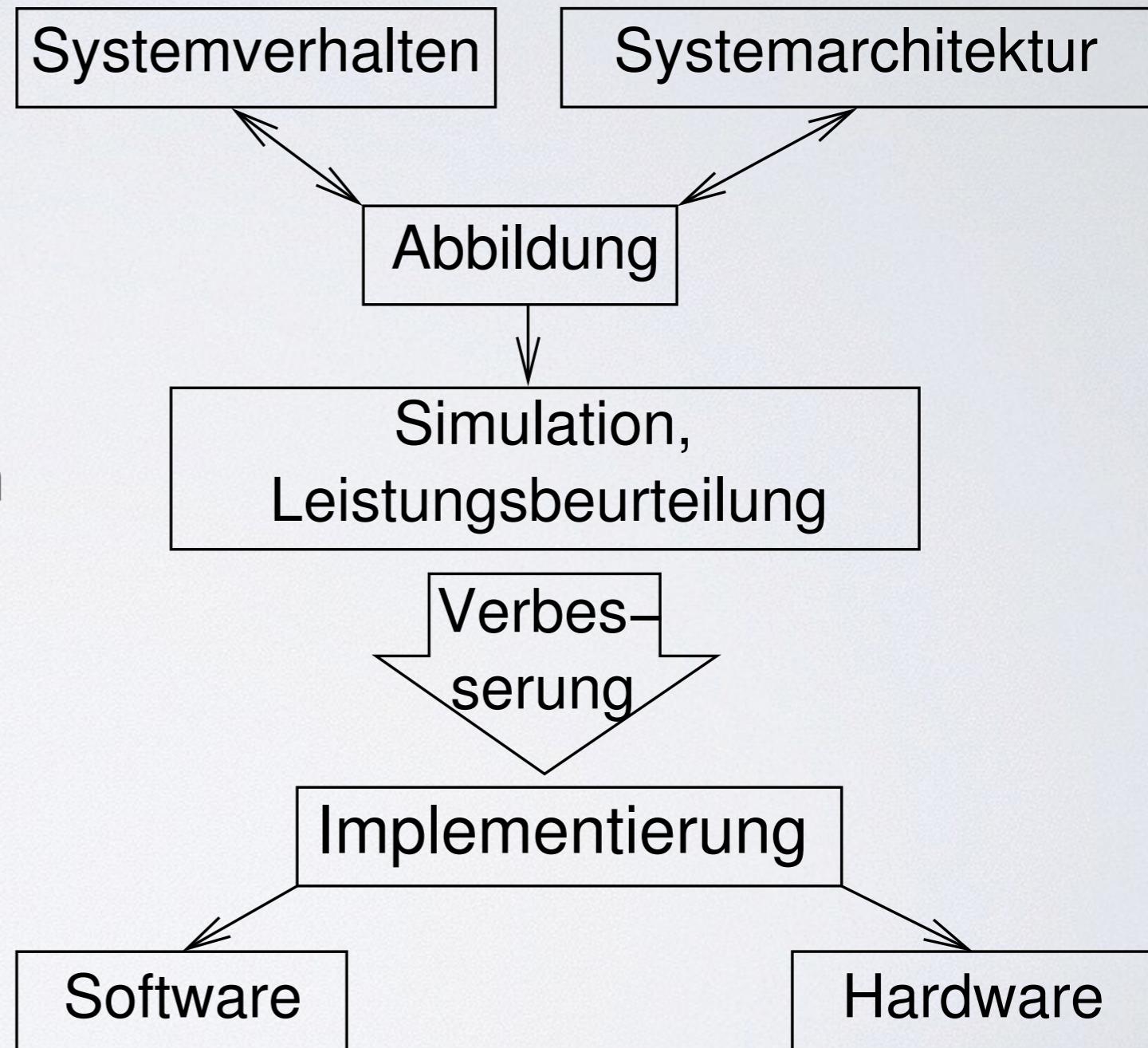


Entwurf Eingebetteter Systeme (nach Marwedel)

[Quelle: Marwedel, Eingebettete Systeme]

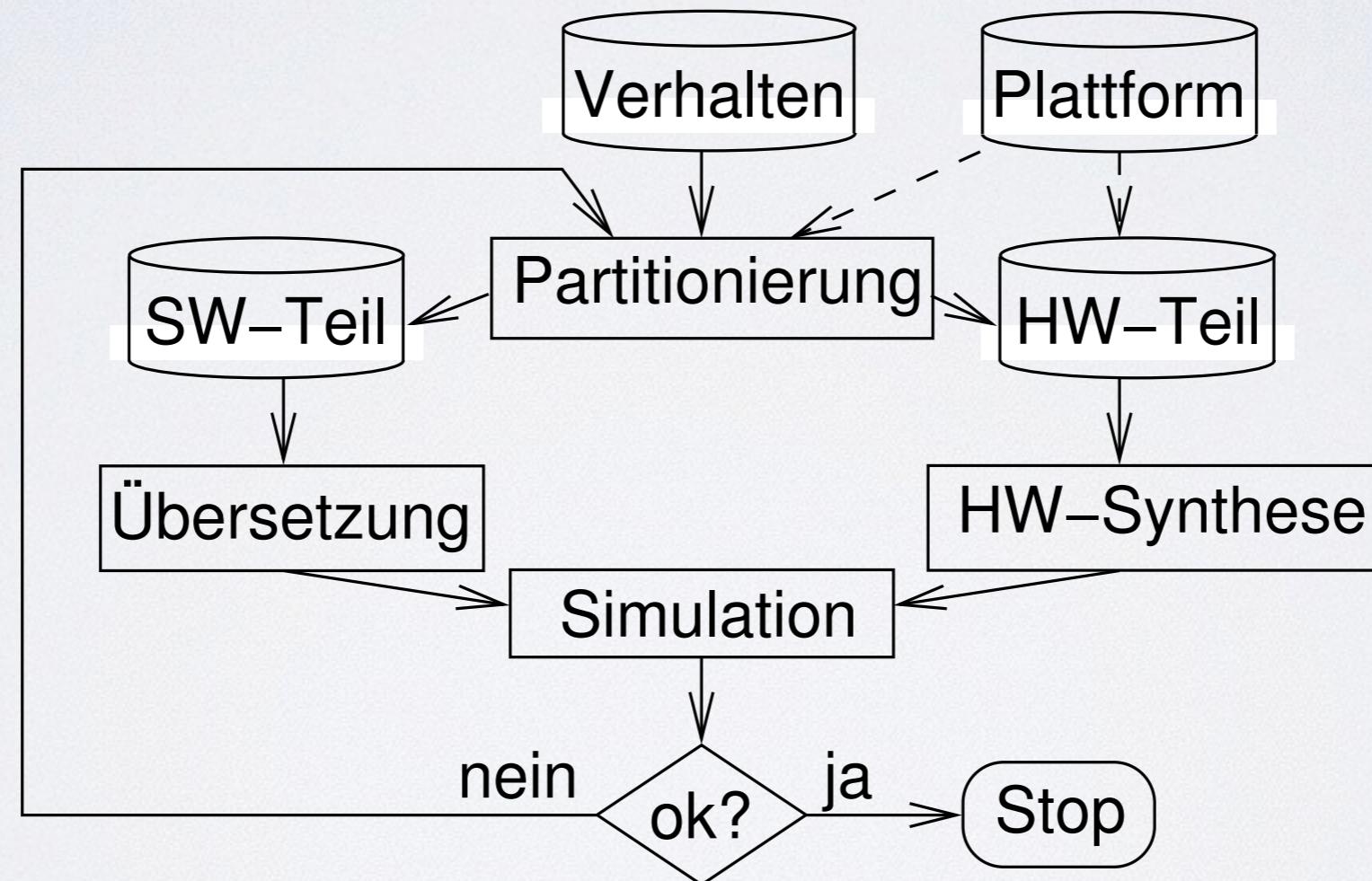
Iterative Verbesserung

- Nach Spezifikation und Beurteilung folgt
 - Implementierung und
 - Optimierung auf Plattform
- Verbesserung auf
 - Task-Ebene
 - Quellcode-Ebene



Hardware-/Software-Partitionierung

- Entscheidung, was in Software und was in Hardware implementiert wird.



Implementierung: Basics

- Im Praktikum bereits kennengelernt:
 - Arbeiten mit Hardware Abstraction Layer
 - Interrupts
 - Interrupt Service Routine sollte kurz sein, da Programmablauf unterbrochen wird.
 - Polling
 - “Richtige” Einstellung der Abfrage-Frequenz
 - Achtung: Prozessor Overhead

Implementierung: Basics

- Im Praktikum bereits kennengelernt:
 - Timer und Watchdog
 - Implementierung von Zustandsautomaten / State Machines:
 - Verschachtelung Zustand/Ereignis (switch-case statement)
 - Zustand Entwurfsmuster (State Pattern)
 - (es gibt noch weitere Ansätze)

Wdh. aus Praktikum: State Machine

- Switch-Case Statement in C:

```
switch (state) {  
  
    case STATE0 : { switch (event) {  
  
        case EVENT0 :  
  
            doSomething();  
  
            state = newState;  
  
            break;  
  
        case EVENT1:  
  
            ...} break;  
  
    case STATE1 : ...
```

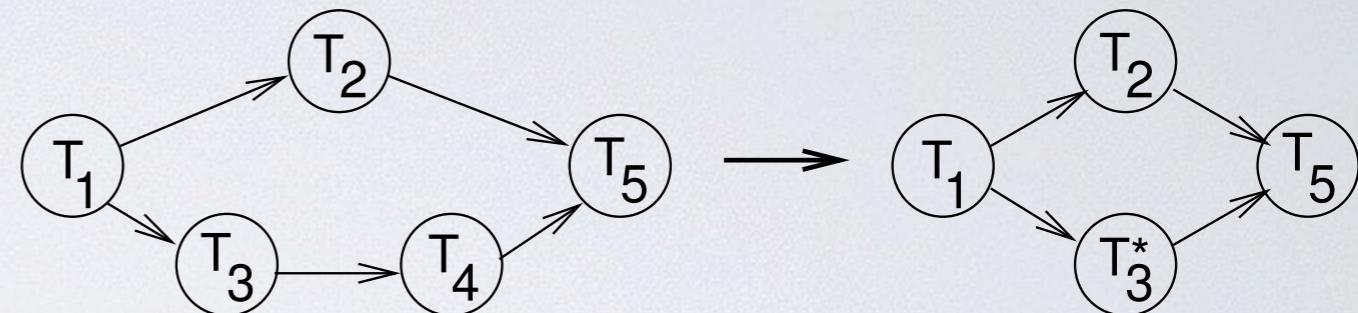
Wdh. aus Praktikum: State Pattern

Ein möglicher Ansatz in C++:

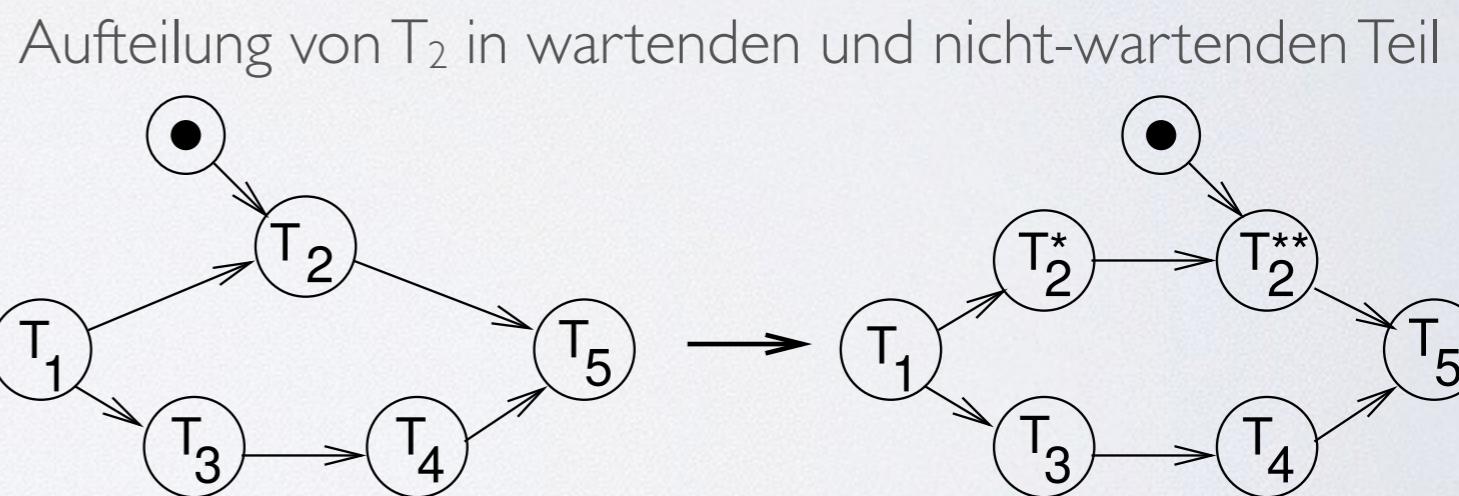
```
struct State  
{ virtual State* processEvent(Event e) = 0; };  
  
class State0 : public State  
{ State* processEvent(Event e) { ... } };  
  
...  
  
currentState = currentState->processEvent(e1);
```

Organisation der Nebenläufigkeit auf Task-Ebene

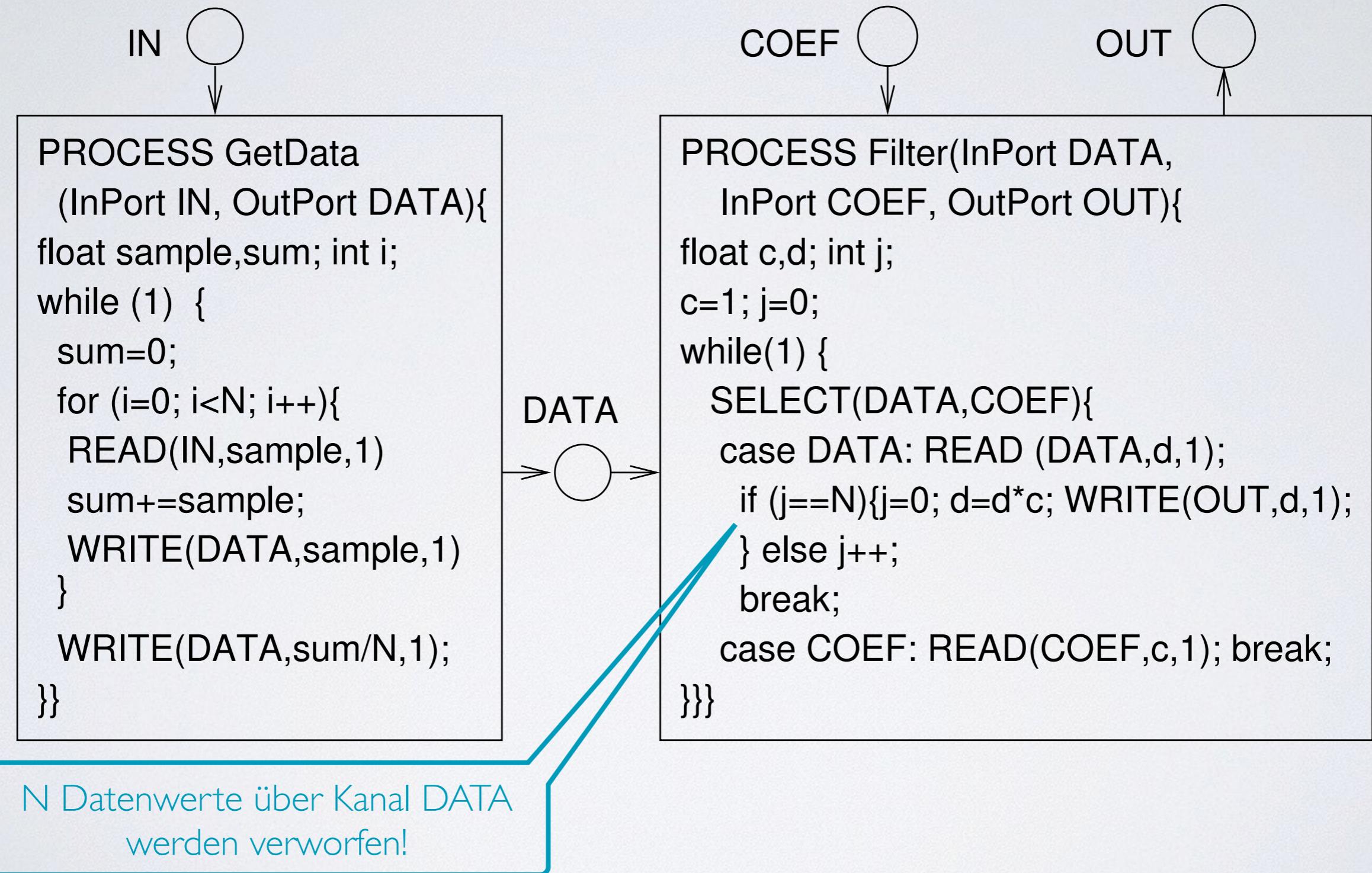
- Verschmelzen von Tasks
 - Verringern des Aufwands von Kontextwechseln



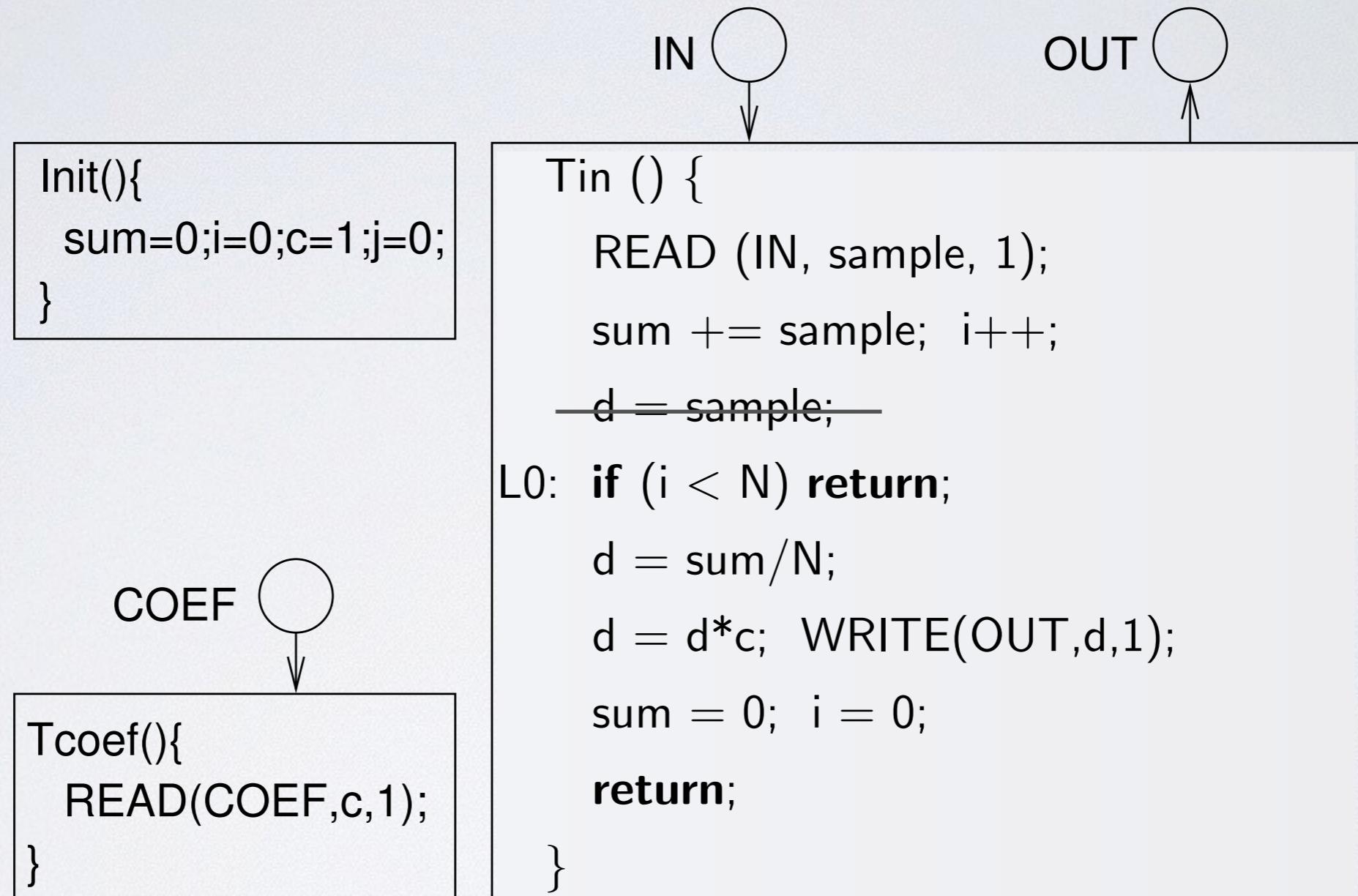
- Aufteilen von Tasks
 - Verbesserung der Ressourcennutzung, z.B. des allokierten Speichers



Beispiel: Neuaufteilung von Tasks - vorher



Beispiel: Neuauflistung von Tasks - nachher



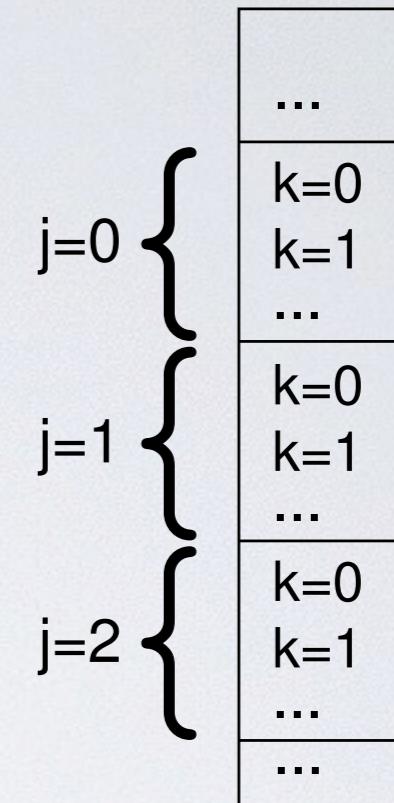
High-Level-Optimierung des Quellcodes

- Schleifen:
 - Einfache Transformationen
 - Kachelweise Verarbeitung
 - Aufteilung
 - Falten
- Mathematik
 - Festkomma-Darstellung
 - Mathematische Näherungen
- Look-up Tabellen

Einfache Schleifentransformationen

```
for (k=0; k<=m; k++)      for (j=0; j<=n; j++)  
  for (j=0; j<=n; j++) ⇒  for (k=0; k<=m; k++)  
    p[j][k] = ...          p[j][k] = ...
```

Bei Row-Major-Anordung in C,
letzter Index in inneren Schleife



- Vertauschen (loop permutation)
 - Abarbeitung von Feldern hinsichtlich ihrer Speicherung
 - Bessere Cache-Nutzung, Zugriff auf folgende Adressen

Einfache Schleifentransformationen

```
for(j=0; j<=n; j++)      for (j=0; j<=n; j++)  
    p[j]= ... ;           {p[j]= ... ;  
for (j=0; j<=n; j++) ⇔   p[j]= p[j] + ... }  
    p[j]= p[j] + ...
```

- Verschmelzen / Aufspalten (loop fusion and fission)
 - Verschmelzen für bessere Cache-Nutzung und Parallelität
 - Aufspalten, falls Hardware-Schleifen-Instruktion möglich

Einfache Schleifentransformationen

```
for (j=0; j<=n; j++)      for (j=0; j<=n; j+=2)  
    p[j]= ... ;           ⇒  {p[j]= ... ;  
                           p[j+1]= ... }
```

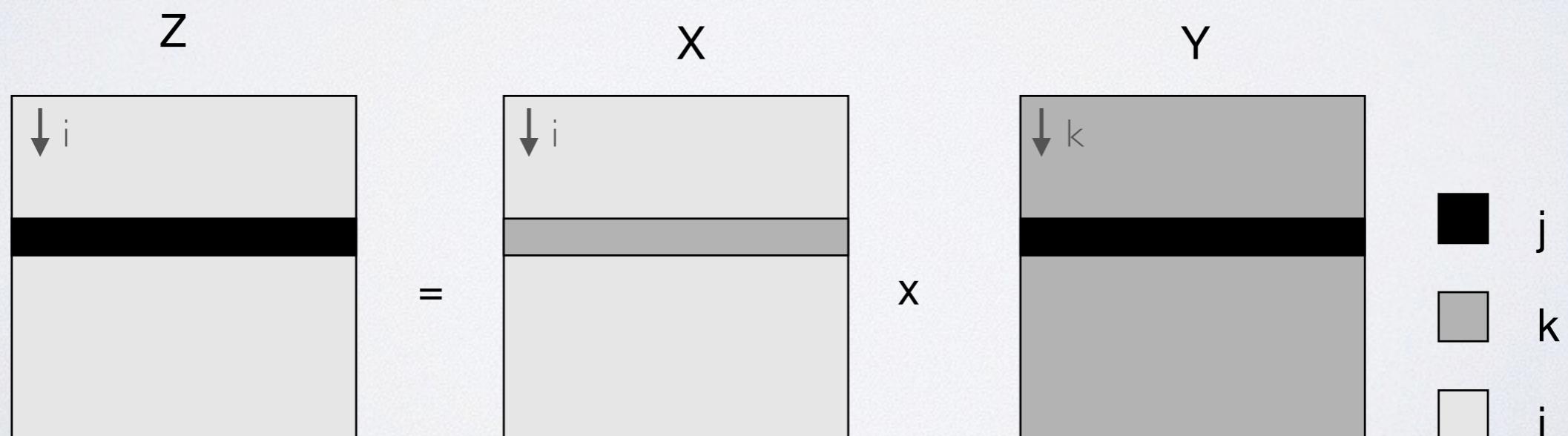
- Abrollen (loop unrolling)
 - Erzeugen mehrerer Instanzen der Anweisungen innerhalb der Schleife
 - Damit Verringerung der Sprungbefehle, Vergrößerung des Codes
 - (s.a. Ergebnisse des Codeerzeugung des Compiles aus dem 2. Praktikum)

Kachelweise Verarbeitung von Schleifen

- Ausgangssituation:

Bei kleinem Cache/großem N → Vielzahl von Speicherzugriffen

```
for (i=1; i<=N; i++)  
  for(k=1; k<=N; k++){  
    r=X[i,k]; /* einem Register zugeordnet */  
    for (j=1; j<=N; j++)  
      Z[i,j] += r* Y[k,j]  
  }
```



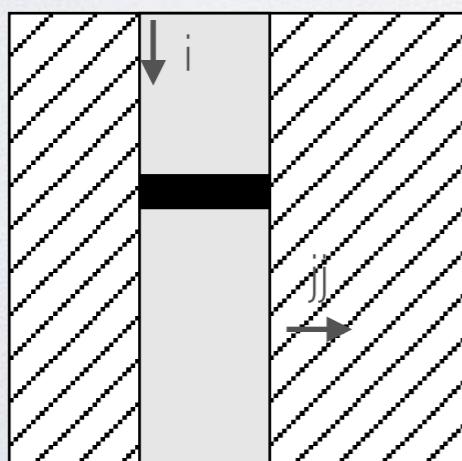
Kachelweise Verarbeitung von Schleifen

- Kachelung mit Blocking-Faktor B, dass $B \times B$ in Cache passt.

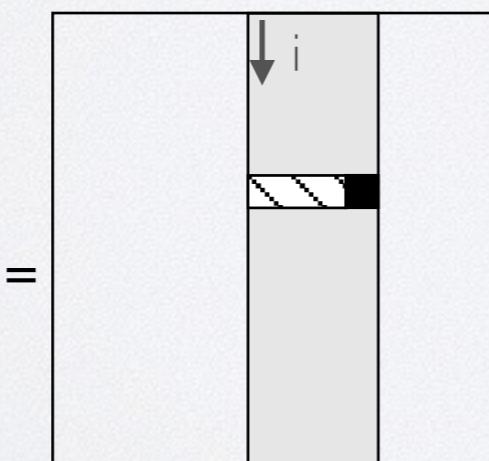
```
for(kk=1; kk<= N; kk+=B)          für Block
    for (jj=1; jj<= N; jj+=B)
        for (i=1; i<= N; i++)
            for (k=kk; k<= min(kk+B-1,N); k++){
                r=X[i][k]; /* einem Register zugeordnet */
                for (j=jj; j<= min(jj+B-1, N); j++)
                    Z[i][j] += r* Y[k][j]
```

}

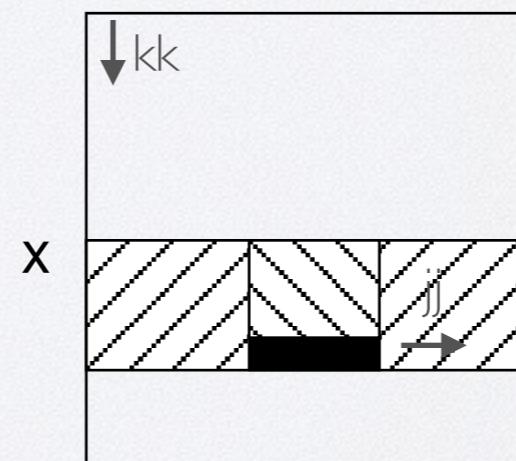
Z



X



Y

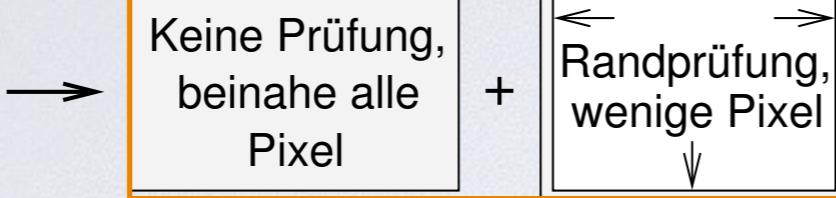


Schleife

■	j	innerste
□	k	
□	i	
□	jj	
□	kk	äußerste

Aufteilung von Schleifen (loop splitting)

Viele If-Befehle
für Randpixel-
Behandlung



```
for (z=0; z<20; z++)  
    for (x=0; x<36; x++) {x1=4*x;  
        for (y=0; y<49; y++) {y1=4*y;  
            for (k=0; k<9; k++) {x2=x1+k-4;  
                for (l=0; l<9; ) {y2=y1+l-4;  
                    for (i=0; i<4; i++) {x3=x1+i; x4=x2+i;  
                        for (j=0; j<4;j++) {y3=y1+j; y4=y2+j;  
                            if (x3<0 || 35<x3||y3<0||48<y3)  
                                then_block_1; else else_block_1;  
                            if (x4<0|| 35<x4||y4<0||48<y4)  
                                then_block_2; else else_block_2;  
                }}}}}}}
```

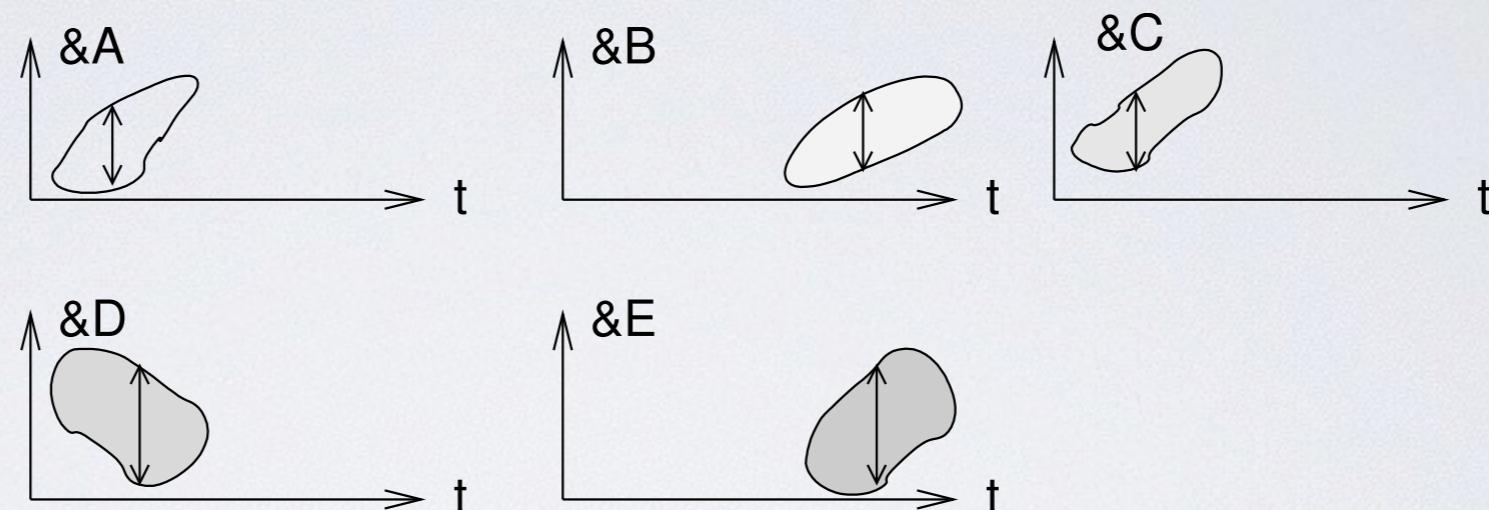
→

```
for (z=0; z<20; z++)  
    for (x=0; x<36; x++) {x1=4*x;  
        for (y=0; y<49; y++)  
            if (x>=10||y>=14) // splitting-if  
                for (; y<49; y++)  
                    for (k=0; k<9; k++)  
                        for (l=0; l<9;l++ )  
                            for (i=0; i<4; i++)  
                                for (j=0; j<4;j++) {  
                                    then_block_1; then_block_2}  
            else {y1=4*y;  
                for (k=0; k<9; k++) {x2=x1+k-4;  
                    for (l=0; l<9; ) {y2=y1+l-4;  
                        for (i=0; i<4; i++) {x3=x1+i; x4=x2+i;  
                            for (j=0; j<4;j++) {y3=y1+j; y4=y2+j;  
                                if (0 || 35<x3 ||0|| 48<y3)  
                                    then_block_1; else else_block_1;  
                                if (x4<0|| 35<x4||y4<0||48<y4)  
                                    then_block_2; else else_block_2;  
                }}}}}}}
```

Falten von Feldern

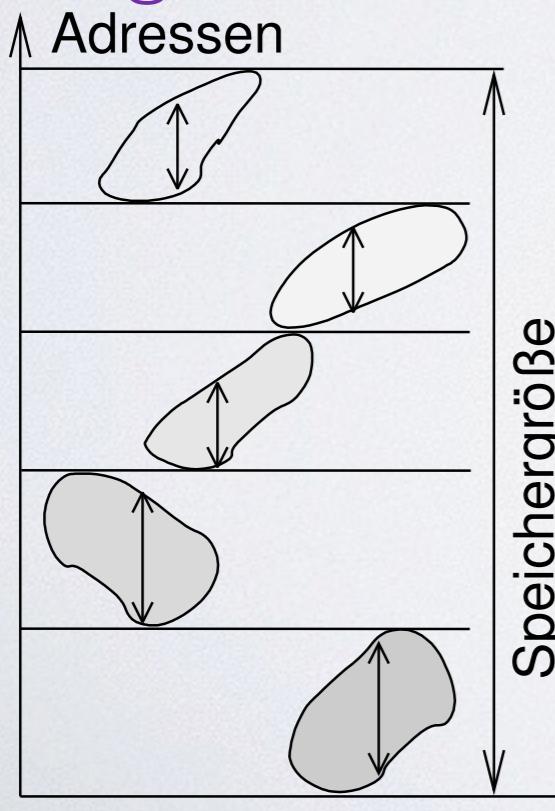
- Verringerung des benötigten Speicherplatzes

Adresszugriff
über Zeit

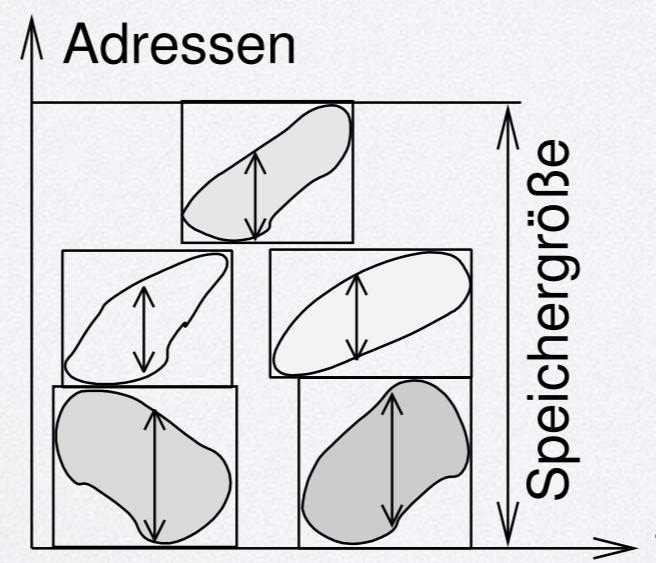


Ungefaltet

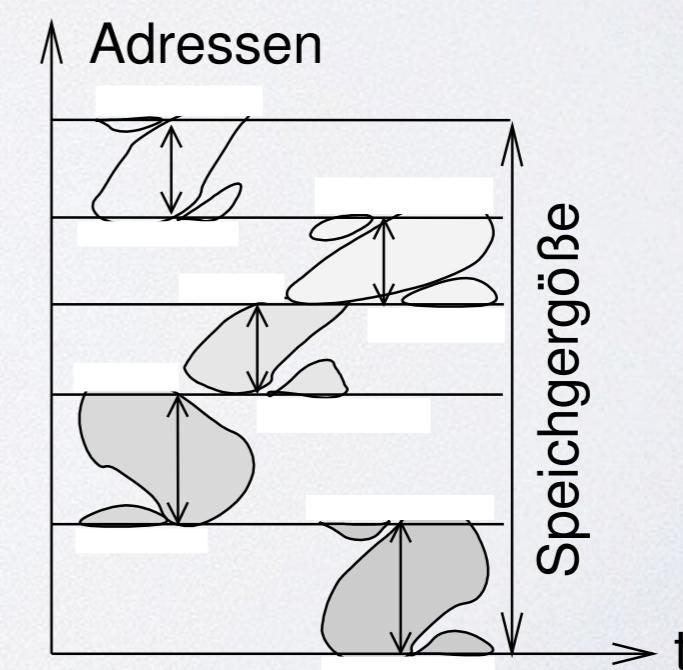
Adressen



Inter-array
folding



Intra-array
folding



Umwandlung von Fließkomma nach Festkomma

- Berechnungen mit Ganzzahlen/Festkomma sind schneller als mit Fließkomma.
- Reihenfolge bei Berechnungen beeinflusst die Genauigkeit.
- Beispiel: Festkommarechnung mit Einheit 0,1:

$a = 125; b = 25; c = 20$ entspricht $a = 12,5; b = 2,5; c = 2,0$

$(a*b)/c = 156$ entspricht 15,6 (Fehler 0,025)

$(a/c)*b = 150$ entspricht 15,0 (Fehler 0,625)

Mathematische Näherungen

- Einige mathematische Funktionen (z.B. sinus) sind teuer.
- Genauigkeit durch Näherung mit Taylorreihen mit wenigen Termen ist ggf. ausreichend.
- Beispiel:

$$\sin(x) \approx x - x^2/6 + x^5/120$$

Look-up Tabellen

- Einträge in einer Tabelle anstelle von Befehlsausführungen
- Insbesondere sinnvoll:
Vorberechnung für teure mathematische Funktionen
- Ablegung der Vorberechnung in Tabelle
- Beispiel: Tabelle für Sinus mit 8 Festpunktwerten mit Offset 16

tab: [16, 27, 32, 27, 16, 5, 0, 5] (Intervall [0..π))
[0, 11, 16, 11, 0, -11, -16, -11] (ohne Offset)

$\sin(x) \triangleq (\text{tab}[(8x)/(2\pi)] - 16) / 16$

Tipps: Richtlinien / Coding Rules

- Insbesondere für Embedded Systeme sinnvoll.
- Beispiele:
 - Kommentare
 - Klammern und Layout von mathematischen Ausdrücken
 - Keine impliziten Typkonvertierungen
 - Keine magischen Zahlen (magic numbers)
 - Einheitliche Namensgebung bei Variablen...

Beispiel aus MISRA C++

Rule 0-1-7	(Required)	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
-------------------	-------------------	--

Rationale

In C++ it is possible to call a function without *using* the return value, which may be an error. The return value of a function shall always be *used*.

Overloaded operators are excluded, as they should behave in the same way as built-in operators.

Exception

The return value of a function may be discarded by use of a (void) cast.

Example

```
uint16_t func ( uint16_t para1 )
{
    return para1;
}

void discarded ( uint16_t para2 )
{
    func ( para2 );           // value discarded - Non-compliant
    (void)func ( para2 );     // Compliant
}
```

See also

Rule 5-2-4

Beispiel aus MISRA C

Rule 2.2	There shall be no <i>dead code</i>
	[IEC 61508-7 Section C.5.10], [ISO 26262-6 Section 9.4.5], [DO-178C Section 6.4.4.3.c]
Category	Required
Analysis	Undecidable, System
Applies to	C90, C99

Amplification

Any operation that is executed but whose removal would not affect program behaviour constitutes *dead code*. Operations that are introduced by language extensions are assumed always to have an effect on program behaviour.

Note: The behaviour of an embedded system is often determined not just by the nature of its actions, but also by the time at which they occur.

Note: *unreachable code* is not *dead code* as it cannot be executed.

Rationale

The presence of *dead code* may be indicative of an error in the program's logic. Since *dead code* may be removed by a compiler, its presence may cause confusion.

Exception

A cast to *void* is assumed to indicate a value that is intentionally not being *used*. The cast is therefore not *dead code* itself. It is treated as using its operand which is therefore also not *dead code*.

Beispiel aus MISRA C

Example

In this example, it is assumed that the object pointed to by p is used in other functions.

```
extern volatile uint16_t v;

extern char *p;

void f ( void )
{
    uint16_t x;

    ( void ) v;          /* Compliant      - v is accessed for its side effect
                           *                      and the cast to void is permitted
                           *                      by exception
                           */
    ( int32_t ) v;       /* Non-compliant - the cast operator is dead
                           */
    v >> 3;             /* Non-compliant - the >> operator is dead
                           */
    x = 3;               /* Non-compliant - the = operator is dead
                           *
                           *                      - x is not subsequently read
                           */
    *p++;
    ( *p )++;
}
```

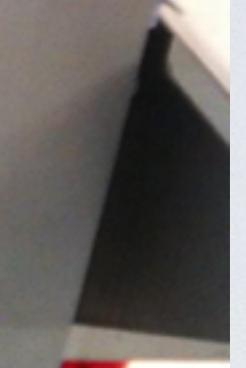
Compiler

- Compiler bieten Optimierungsmöglichkeiten für
 - Laufzeit / Geschwindigkeit
 - Codegröße
 - Energieverbrauch
 - Ausnutzen des Speicherhierarchie
 - Ändern der Befehlsreihenfolge

Zusammenfassung

- Hardware-/Software-Partitionierung
- Implementierung: einige Basics für Embedded Systems
- Transformationen von Tasksaufteilung
- High-Level-Optimierung von Quellcode
- Mathematische Näherungen
- Coding Rules

Made in Germany



Beispiel einer Anwendung: Spargelschälmaschine

- Eigenschaften:
 - Optische Anzeige
 - Betriebsstundenzähler
 - Elektronische Überwachung des Schälvorgangs

[Quelle: Hepro GmbH]



Beispiel einer Spargelschälmaschine

Literatur / Quellen

- Hapro GmbH, URL: <http://www.hepro-gmbh.de>
- Peter **Marwedel**, Eingebettete Systeme, Springer-Verlag, 2008
- Chris **Tapp**, MISRA C++, MISRA-C++:2008 Launch, London 5. Juni 2008, URL: http://www.phaedsys.org/standards/misra/misradata/MISRA_CppLaunchPresentation.pdf
- MISRA C:2012, MIRA Limited 2013
- Wikipedia, Funkmeldeempfänger, URL: <http://de.wikipedia.org/wiki/Funkmeldeempfänger>
- **Stand aller Internetquellen: 18.06.2014**