

Embedded Systems / Eingebettete Systeme

Studiengang Informatik
Campus Minden

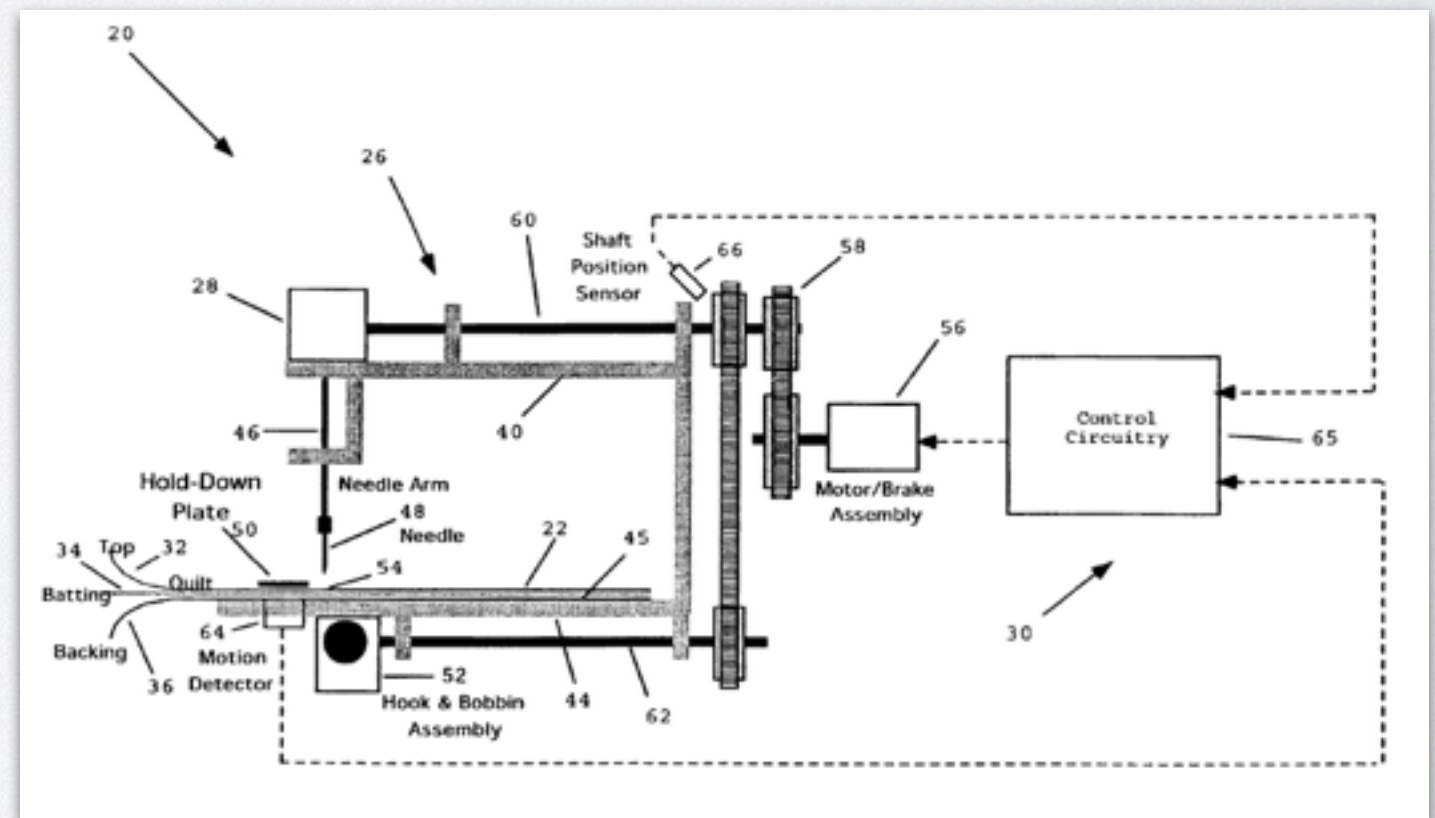
Matthias König



FH Bielefeld
University of
Applied Sciences

Beispiel einer Anwendung: Nähmaschine

- Sensoren zur Messung
 - der Geschwindigkeit des Nähstoffes
 - Position der Nähnadel
- Aktor Motor/Bremse



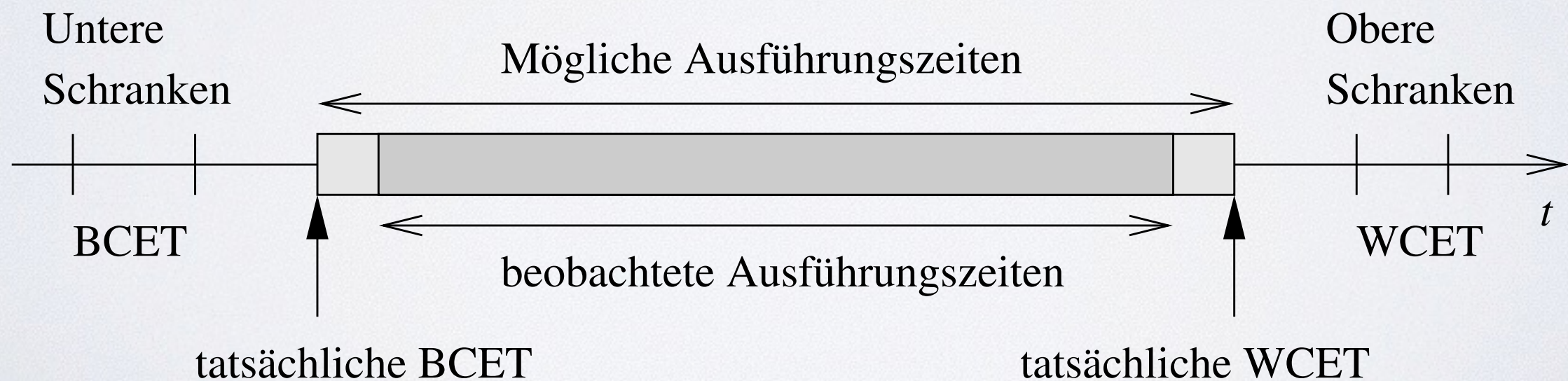
SCHEDULING

Scheduling und Echtzeit

- Scheduling
 - Zuordnung von Jobs/Tasks zu Ressourcen/Prozessoren
- Systeme mit Echtzeit-Anforderungen
 - Reaktion in vorgegebener Zeitspanne (gleich mehr)
 - Notwendig spezielle Scheduling-Verfahren
- Einschätzung der Eignung eines Scheduling-Verfahrens anhand längster Ausführungszeit (worst case Betrachtung)
- Praktischer Einsatz in RTOS (nächste Woche)

Worst Case Execution Time WCET

- Längst mögliche Ausführungszeit / WCET
- Obere Schranken als Ausgangsbasis für Scheduling-Verfahren



Schedulingalgorithmen

- Schedulingalgorithmen von Echtzeitbetriebssystemen
- Unterscheidung nach
 - harte und weiche Deadlines
 - harte Deadlines weiterhin nach
 - periodisch/aperiodisch
 - präemptiv/nicht-präemptiv
 - statisch/dynamisch

Scheduling: Soft/Hard Deadlines

- Weiche Zeitbedingungen
 - Verfehlen einer Zeitbedingung ist kein Fehler, statistisch sollte die Zeitbedingung eingehalten werden.
- Harte Zeitbedingungen
 - Verfehlen einer Zeitbedingung ist ein **Fehler**, Folgen können **dramatisch** sein.

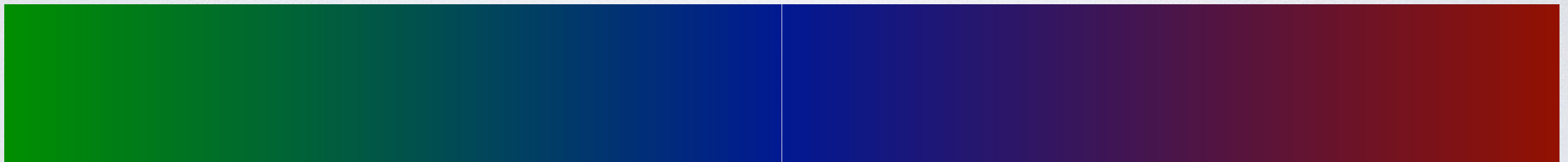
Echtzeit und Anwendungsgebiete

Beispiele: Anforderungen an Echtzeit (Real-time)

Keine

Weiche (Soft)

Harte (Hard)



Simulationen

Desktopanwendungen

Video-/Audiostreaming

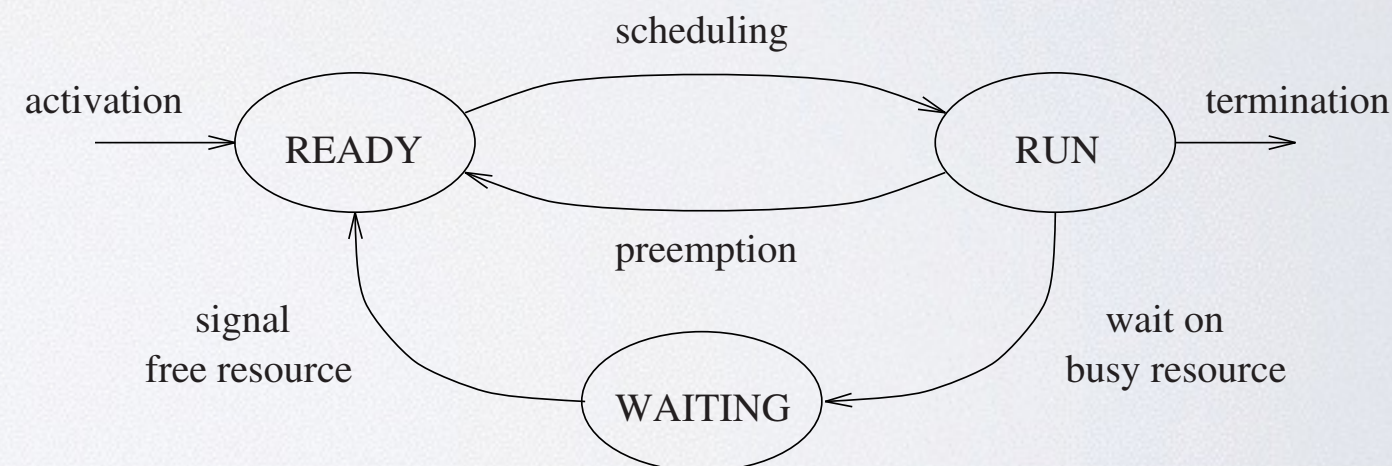
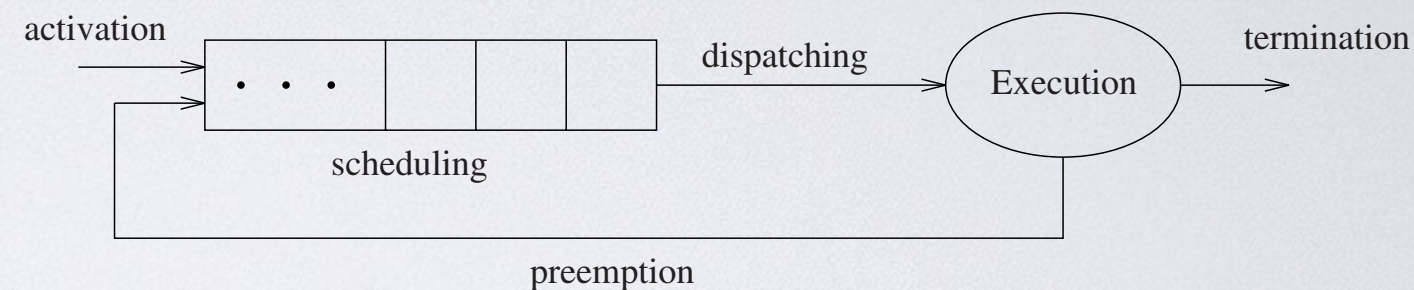
Telekommunikation

Fabrikautomatisierung

Motorsteuerung

Tasks und Zustände

- Zustände eines Tasks
 - *Ready*, bereit zur Ausführung
 - *Waiting/Blocked*, blockiert durch nicht verfügbare Ressource
 - *Run*, in Ausführung
- Verwaltung bereiter Tasks in Queue von Scheduler
- Präemption/Unterbrechung eines Tasks für wichtigeren



Scheduling und Overhead

- Scheduling in der Praxis mit Rechenzeit/Overhead für
 - Umschalten zwischen Prozessen(Context Switch)
 - Verwaltung der Tasks-Queue
 - ggf. Auswertung von Prioritäten der Tasks
 - Hardware-Schaltzeiten

wird im Folgenden im Wesentlichen nicht betrachtet.

Harte Zeitbedingungen

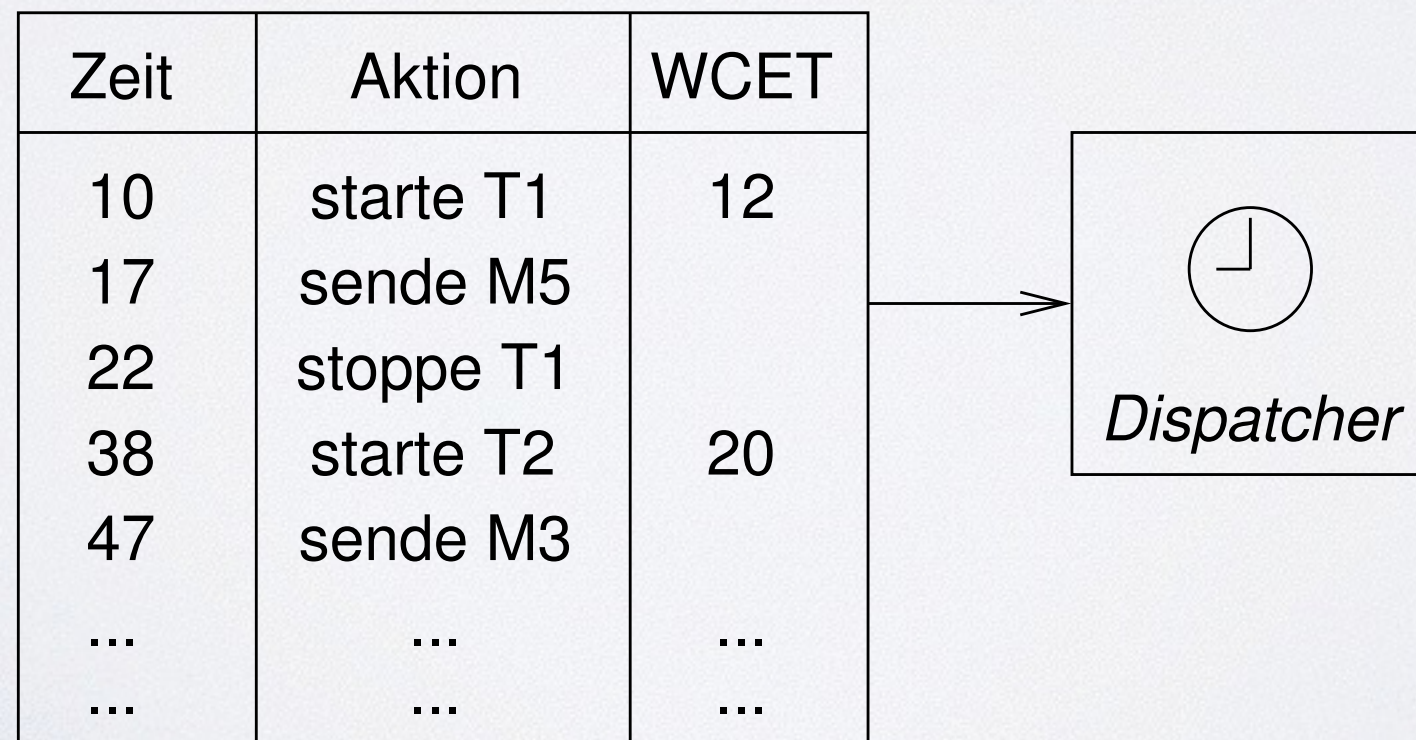
- Tasks sind
 - **periodisch**, wenn sie alle p Zeiteinheiten ausgeführt werden,
 - **aperiodisch**, ansonsten.
- Scheduler sind
 - **nicht-präemptiv**, wenn sie die Ausführung eines Tasks abwarten,
 - **präemptiv**, wenn sie die Ausführung eines Tasks unterbrechen.

Harte Zeitbedingungen

- Ein Scheduler ist
 - **dynamisch**, wenn die Entscheidung zur Laufzeit getroffen wird (auch on-line Scheduler genannt).
 - **statisch**, wenn die Entscheidung zur Entwurfszeit (vor der Laufzeit) festgelegt werden (off-line Scheduler).
- Statische Scheduler “kennen” Abhängigkeiten von Tasks.

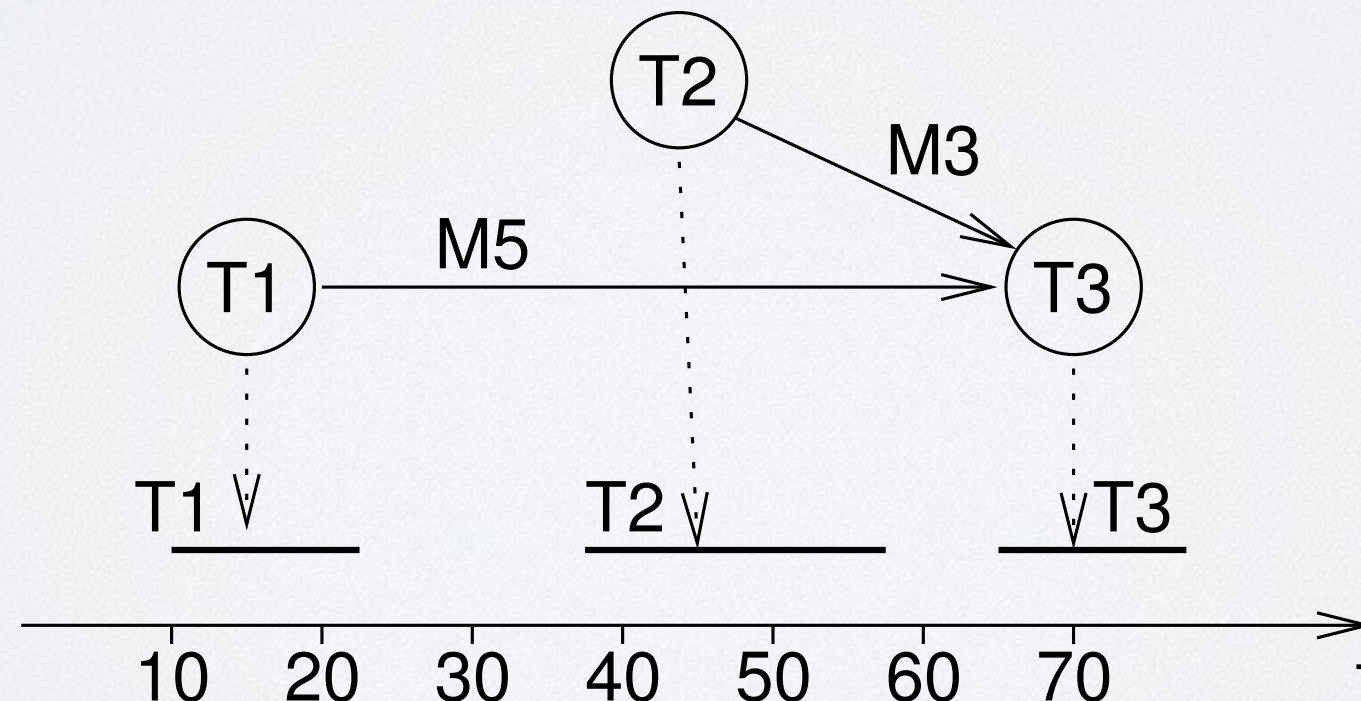
Statische Scheduler

- Normalerweise sind Startzeiten in Tabellen abgelegt.
- Ein Dispatcher startet Tasks zu den angegebenen Zeiten.
- Ablaufplanung erfolgt a priori (ggf. mittels entsprechenden Programmen).
- Leichte Überprüfbarkeit von Zeitbedingungen.



Abhängige Tasks

- Bei abhängigen Tasks, Nutzung eines Tasks-Graphen
- Bei statischem Scheduling, Tabelle für Dispatcher für Startzeiten der Tasks T_i und Nachrichten M_i



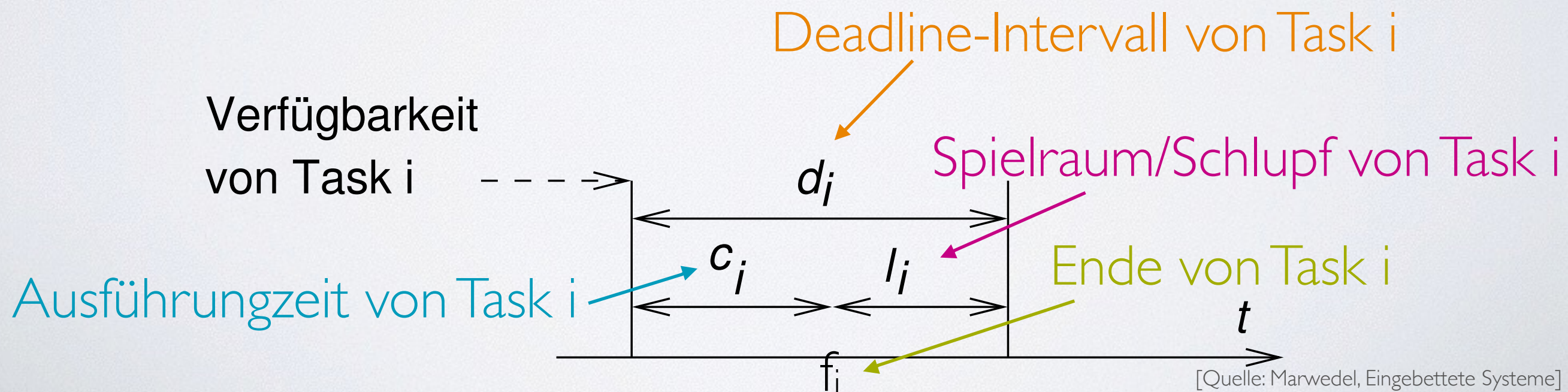
Aperiodisches Scheduling

- **Maximale Verspätung:**

- Maximale Differenz zwischen Ausführungsende und Deadline über alle Tasks; negativ, wenn vor Deadline fertig.

- **Deadline-Intervall:**

- Zeit zwischen Verfügbarkeit und Deadline



Round Robin mit Zeitschlitz

- Sequentielle Abarbeitung der Tasks (gemäß FIFO)
 - Einreihen in Queue (*Waiting*)
 - Erster Task aus Queue zur Ausführung (Running)
 - Dauer der Ausführung: eine Zeitscheibe (Timeslice/Quantum)
 - Keine Unterbrechung/Präemption
 - Danach an das Ende der Queue (*Waiting*)
 - Nächster Task

Scheduling: Earliest Due Date

- Annahme: alle Tasks gleichzeitig verfügbar
- Ausführung von Tasks in der Reihenfolge nicht-abnehmender Deadlines (früheste Deadline zuerst)
- Optimal für Minimierung der maximalen Verspätung
- Implementierung als statisches Scheduling möglich
- Komplexität für Sortieren $O(n \log(n))$

Scheduling: Earliest Due Date

Beispiel

	T_1	T_2	T_3	T_4
c_i	1	2	1	3
d_i	2	5	3	6

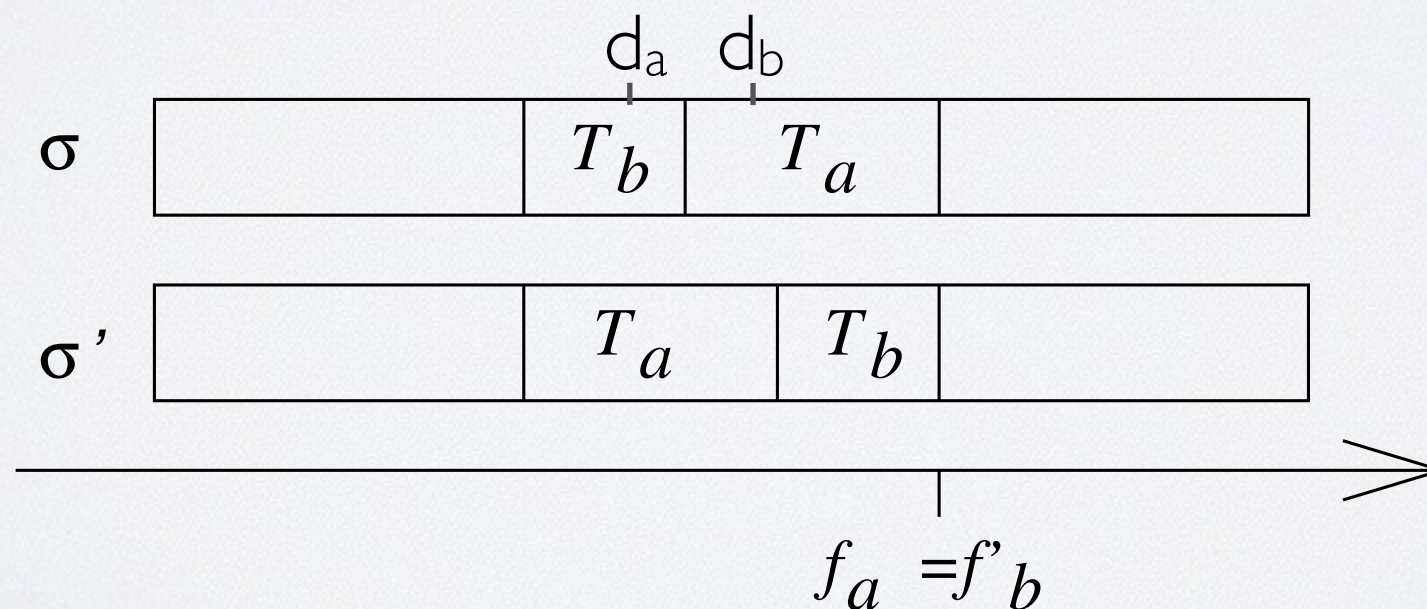
EDD Reihenfolge: T_1 T_3 T_2 T_4

Taskende: $f_1 = 1$, $f_3 = 2$, $f_2 = 4$, $f_4 = 7$

Maximale Verspätung: 1 (T_4)

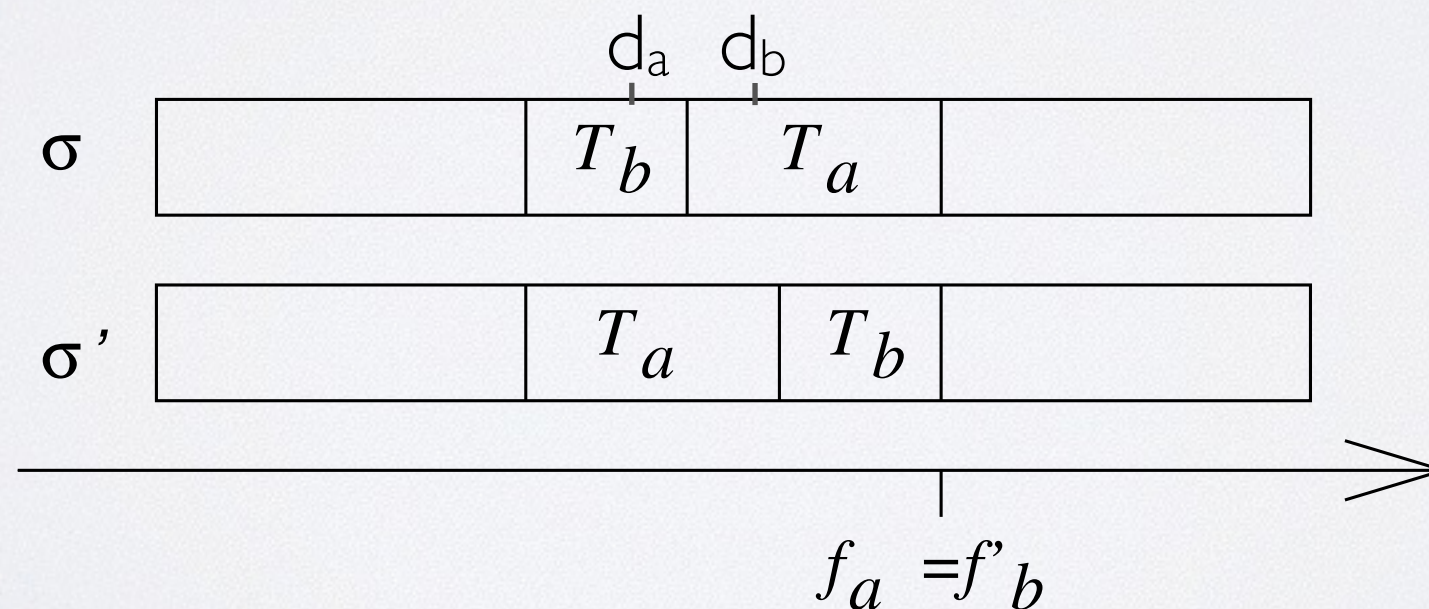
Scheduling: Earliest Due Date

- Beweis der Optimalität: sei σ ein Schedule von Algorithmus A.
- Falls A nicht EDD, dann gibt es T_a, T_b in σ , so dass T_b vor T_a und $d_a < d_b$.
- Sei σ' aus σ nach Vertauschung der Reihenfolge T_a und T_b .
- Max. Verspätung für T_a, T_b in σ ist $f_a - d_a$.



Scheduling: Earliest Due Date

- Max. Verspätung für T_a, T_b in σ' ist $\max\{f_a' - d_a, f_b' - d_b\}$
- Es ist $f_a' - d_a < f_a - d_a$ und $f_b' - d_b = f_a - d_b < f_a - d_a$.
- Also kann für ein EDD-Schedule die maximale Verspätung nur kleiner werden. Daher ist EDD optimal.



Earliest Deadline First EDF

- Annahme: unterschiedliche Ankunftszeit von Tasks
- Unterbrechen von Tasks (preemption) verbessert maximale Verspätung.
- Zu jedem Zeitpunkt wird Task mit der frühesten absoluten Deadline ausgeführt, ggf. Unterbrechen des aktuellen Task.
- Optimal für Minimierung der maximalen Verspätung
- Warteschlange von ausführbaren Tasks nach Deadlines
- Dynamisches Scheduling

Earliest Deadline First Algorithmus

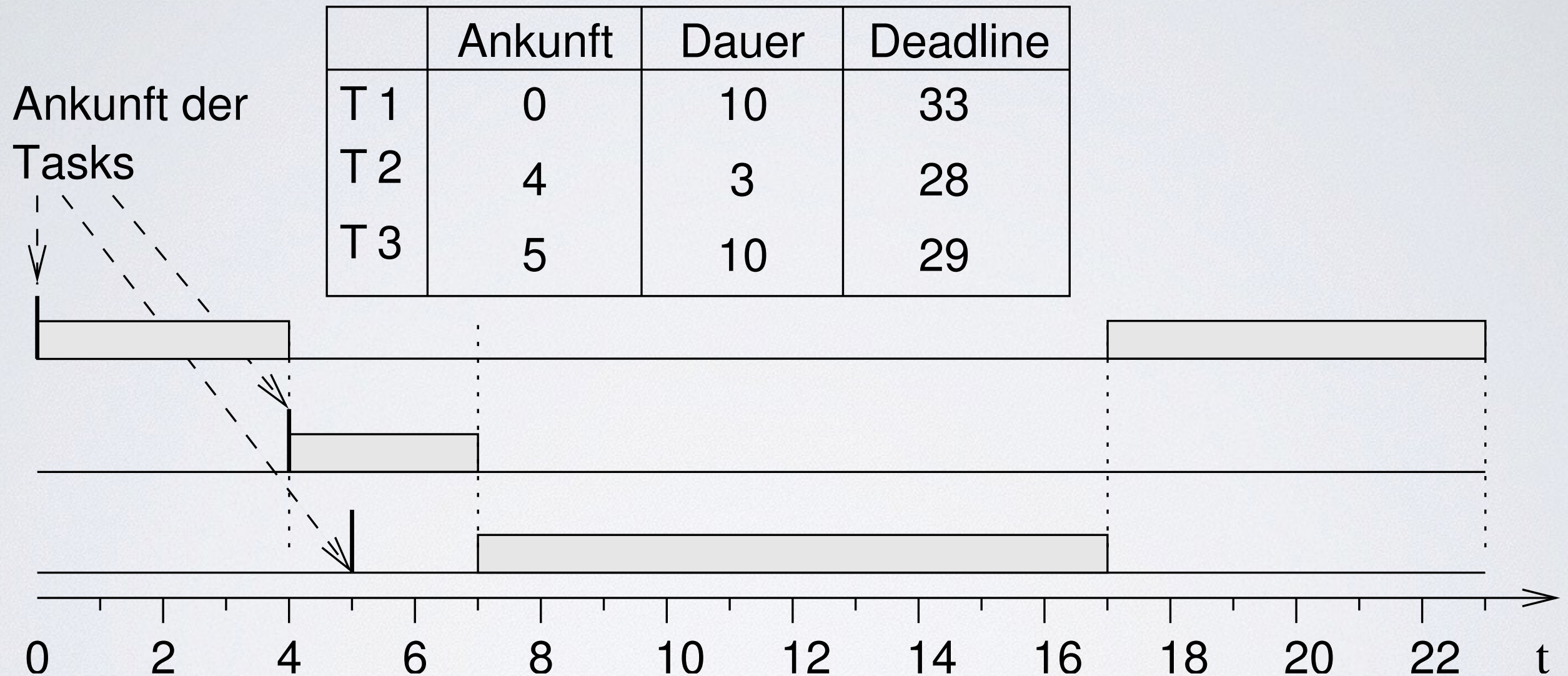
```
/* linked list, sorted by deadline */
Activation_record *processes;
/* data structure for sorting processes */
Deadline_tree *deadlines;

void expired_deadline(Activation_record *expired){
    remove(expired); /* remove from the deadline-sorted list */
    add(expired,expired->deadline); /* add at new deadline */
}

Void EDF(int current) { /* current = currently executing process */
    int i;
    /* turn off current process (may be turned back on) */
    processes->state = READY_STATE;
    /* find process to start executing */
    for (alink = processes; alink != NULL; alink = alink->next_deadline)
        if (processes->state == READY_STATE) {
            /* make this the running process */
            processes->state == EXECUTING_STATE;
            break;
        }
    }
}
```


Earliest Deadline First EDF

Beispiel

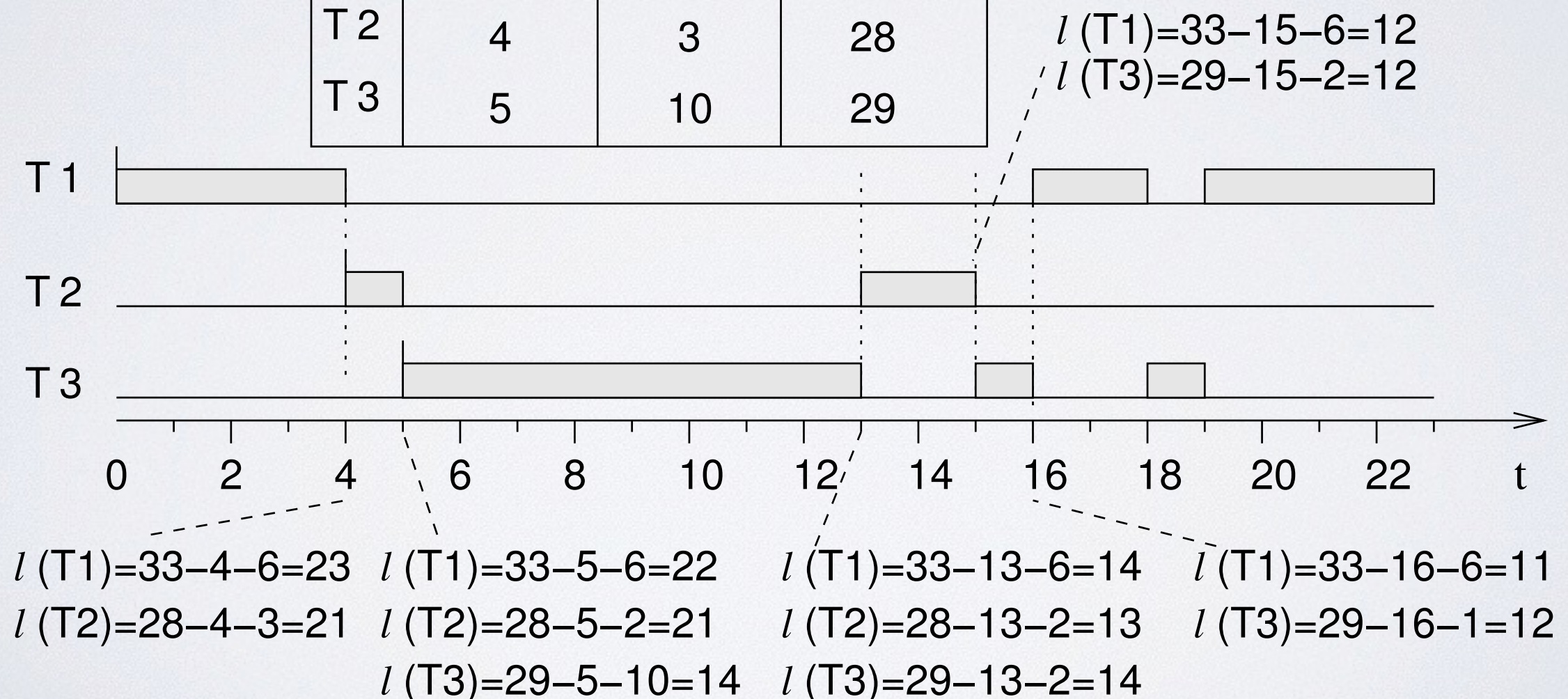


Least-Laxity (Geringster Schlupf)

- Prioritäten der Tasks sind monoton fallende Funktion ihres Schlupfs. Schlupf verändert sich dynamisch.

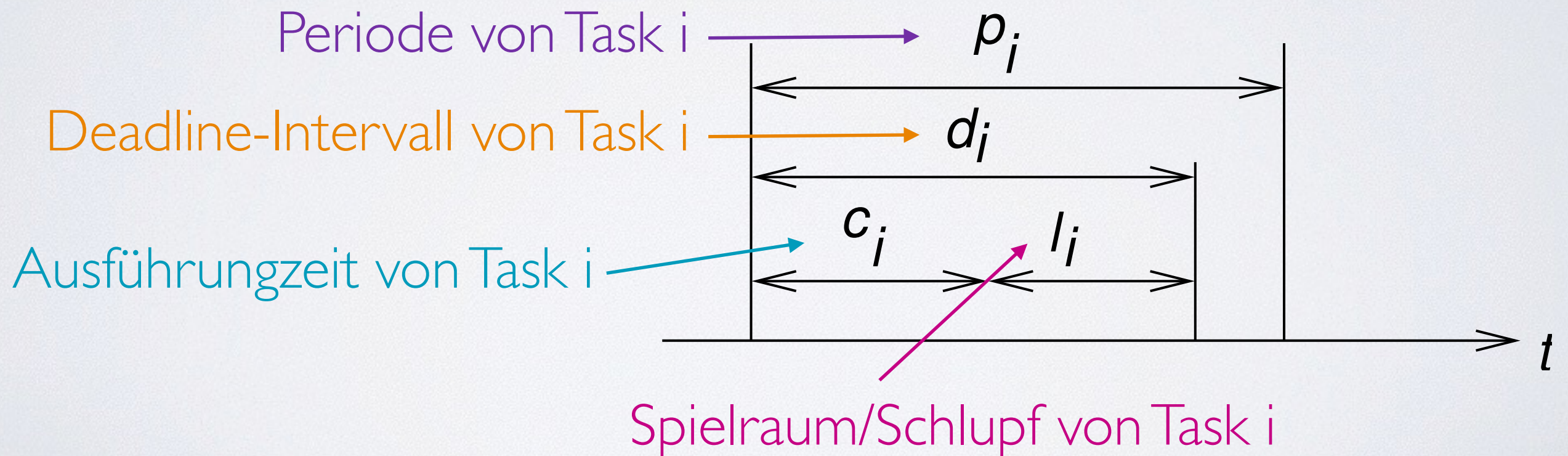
- Präemptiv.

	Ankunft	Dauer	Deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29



Periodisches Scheduling

- Jede Ausführung eines Tasks heißt Job.
- Annahme: Die Ausführungszeit aller Jobs eines Task ist gleich.
- Durchschnittliche Prozessauslastung für n Prozesse: $\mu = \sum_{i=1}^n c_i / p_i$



Rate Monotonic Scheduling

- Annahmen:

- unabhängige Tasks

- $d_i = p_i$

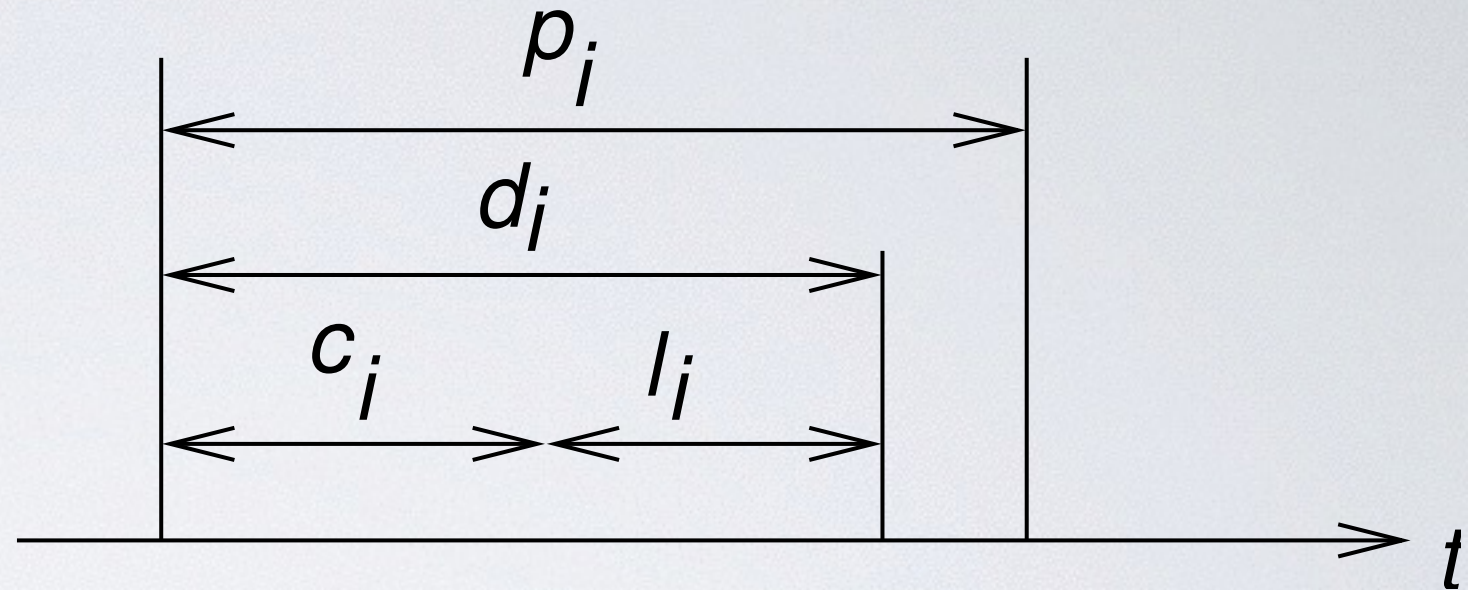
- c_i konstant und bekannt

- bei n Tasks Einhaltung von Deadlines für Auslastung

$$\mu = \sum_{i=1}^n c_i / p_i \leq n(2^{1/n} - 1)$$

- Priorität ist monoton fallende Funktion ihrer Periode, also haben Tasks mit kurzer Periode hohe Priorität.

- Prioritäten statisch.



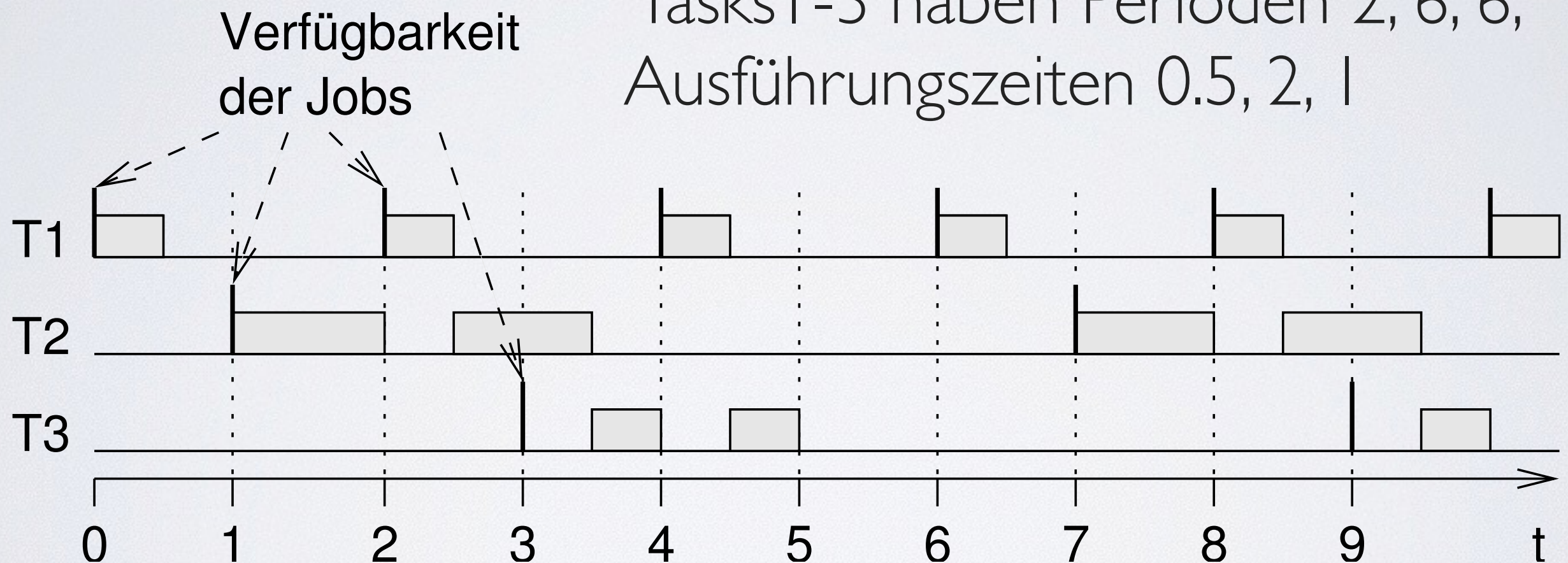
Rate Monotonic Scheduling - Algorithmus

```
/* processes[] is an array of process activation records,
   stored in order of priority, with processes[0] being
   the highest-priority process */
Activation_record processes[NPROCESSES];

void RMA(int current) { /* current = currently executing
   process */
    int i;
    /* turn off current process (may be turned back on) */
    processes[current].state = READY_STATE;
    /* find process to start executing */
    for (i = 0; i < NPROCESSES; i++)
        if (processes[i].state == READY_STATE) {
            /* make this the running process */
            processes[i].state == EXECUTING_STATE;
            break;
        }
}
```


Rate Monotonic Scheduling - Beispiel

Tasks 1-3 haben Perioden 2, 6, 6,
Ausführungszeiten 0.5, 2, 1

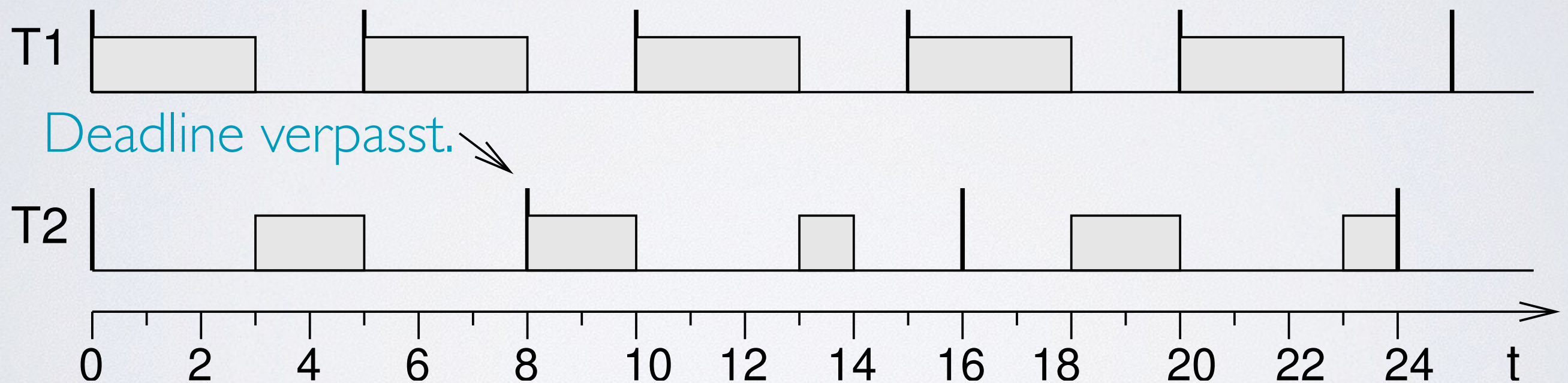


Rate Monotonic Scheduling - Beispiel

Task 1: Periode 5, Ausführungszeit 3

Task 2: Periode 8, Ausführungszeit 3

Annahme: Bedingung $\mu \leq n(2^{1/n} - 1)$ nicht gegeben

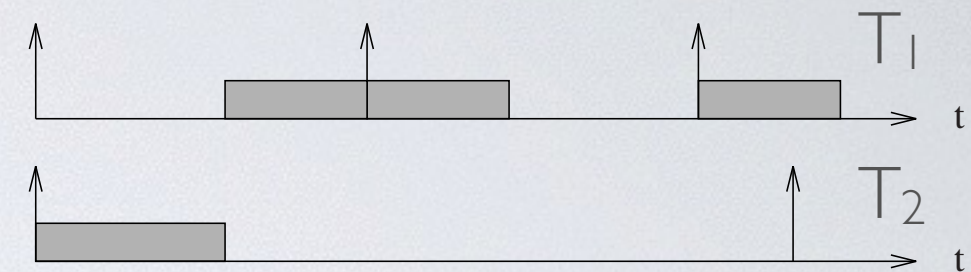


Rate Monotonic Analysis / RMA

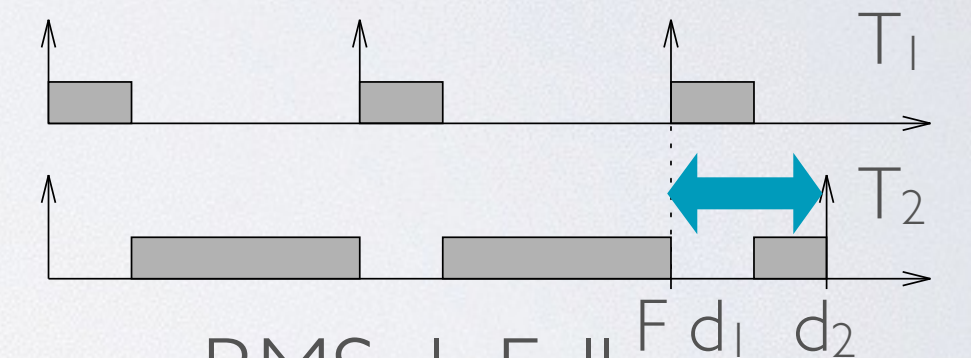
- Analyse des Rate Monotonic Schedulings
- Betrachtung des Falls mit der schlechtesten Antwortzeit
 - Alle Jobs gleichzeitig bereit
 - Kritische Antwortzeit beim niedrig periodisierten Task
- Hier: nur illustrative Betrachtung für zwei Tasks

RMA: Optimalität von RMS

- Tasks T_1 und T_2 mit $d_1 < d_2$
- Wenn beliebiges Scheduling (kein RMS) durchführbar mit $c_1 + c_2 \leq d_1$
- dann auch RM Scheduling ($F = \lfloor d_2/d_1 \rfloor$)

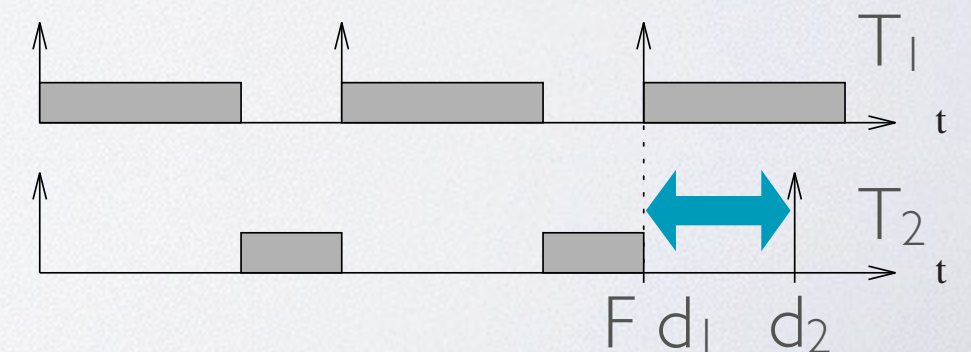


Kein RMS



RMS, 1. Fall

2. Fall, $c_1 \geq d_2 - F d_1$ durchführbar mit $F c_1 + c_2 \leq F d_1$



RMS, 2. Fall

RMA: Obere Schranke Prozessorauslastung

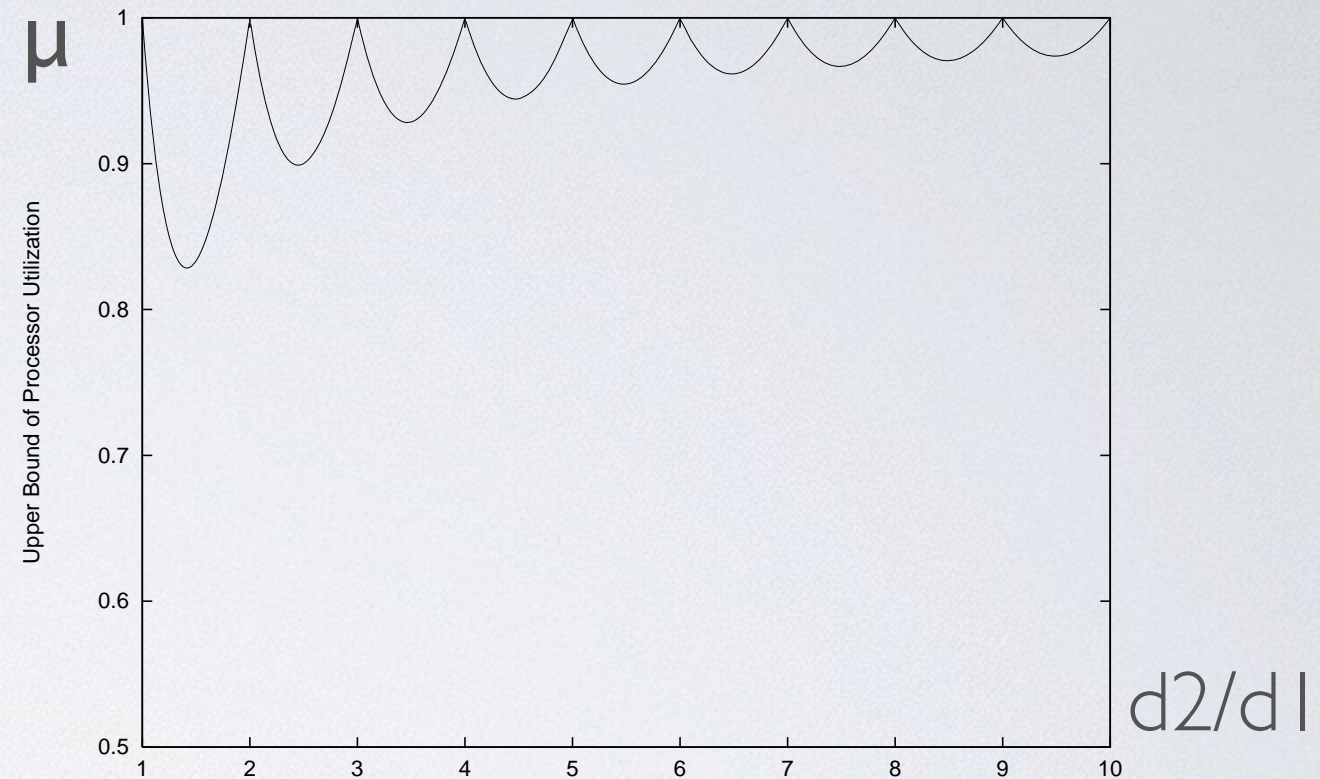
- Obere Schranke der Prozessorauslastung

$$\mu = \sum_{i=1}^n c_i / p_i \leq n(2^{1/n} - 1)$$

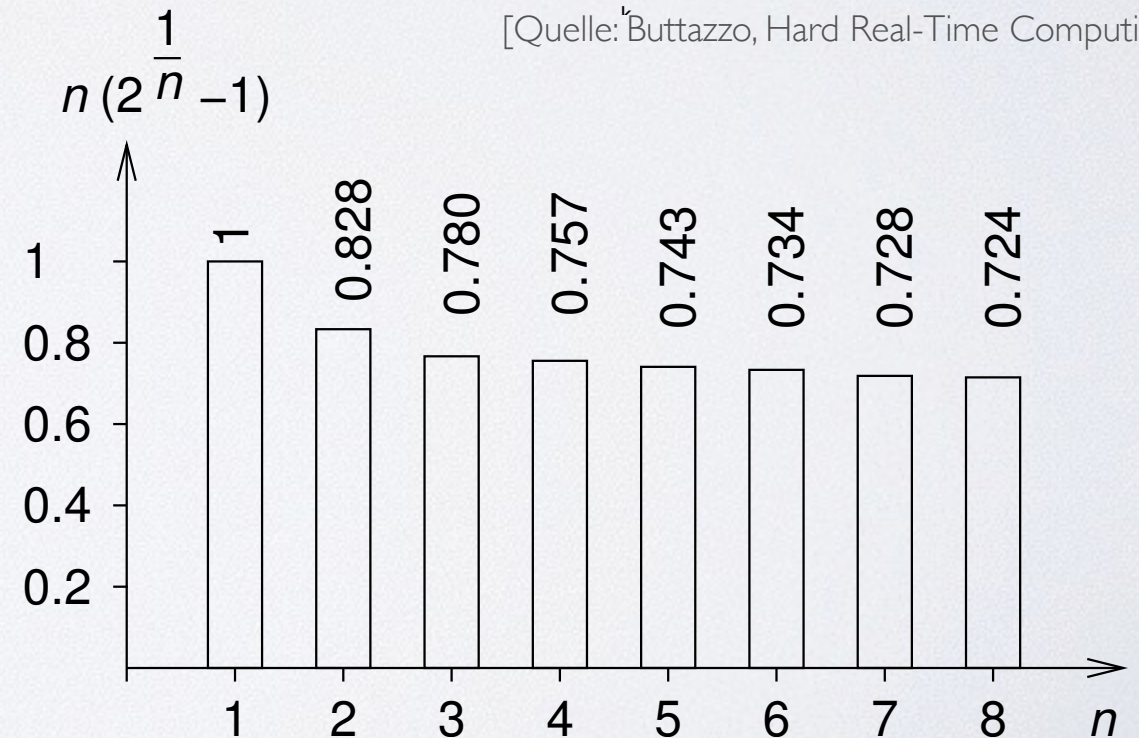
- Schritte zur Herleitung für zwei Tasks ($\mu = c_1/d_1 + c_2/d_2$)
 - Einsetzen für c_2 (zwei Fälle) und Bestimmung des Minimums (monoton abfallende Funktion): μ minimal für $c_1 = d_2 - d_1$ F.
 - Einsetzen von $c_1 = d_2 - d_1$ F in μ , Vereinfachung mit $G = d_2/d_1 - F$, Bestimmung von F (=1) für Minimum von μ .
 - Bestimmung des Minimums von μ durch Ableitung nach G
 - Ergebnis: $\mu = 2(2^{1/2} - 1) \approx 0,83$.

RMA: Obere Schranke Prozessorauslastung

- Wenn Periode aller Tasks ein ganzzahliges Vielfaches des jeweils höheren Task, dann $\mu \leq 1$
- Ansonsten $\mu \leq n(2^{1/n} - 1)$



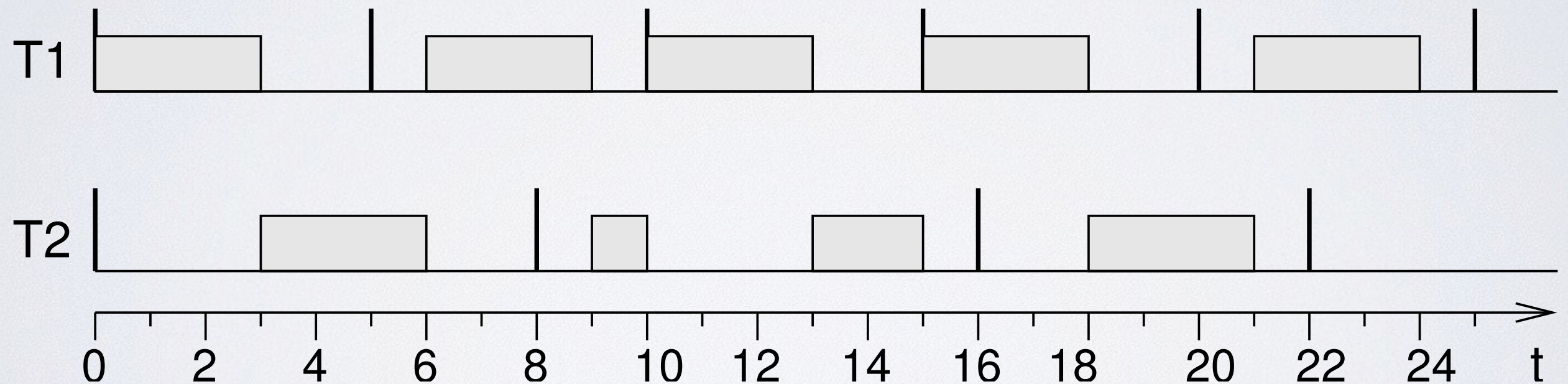
[Quelle: Buttazzo, Hard Real-Time Computing Systems]



[Quelle: Marwedel, Eingebettete Systeme]

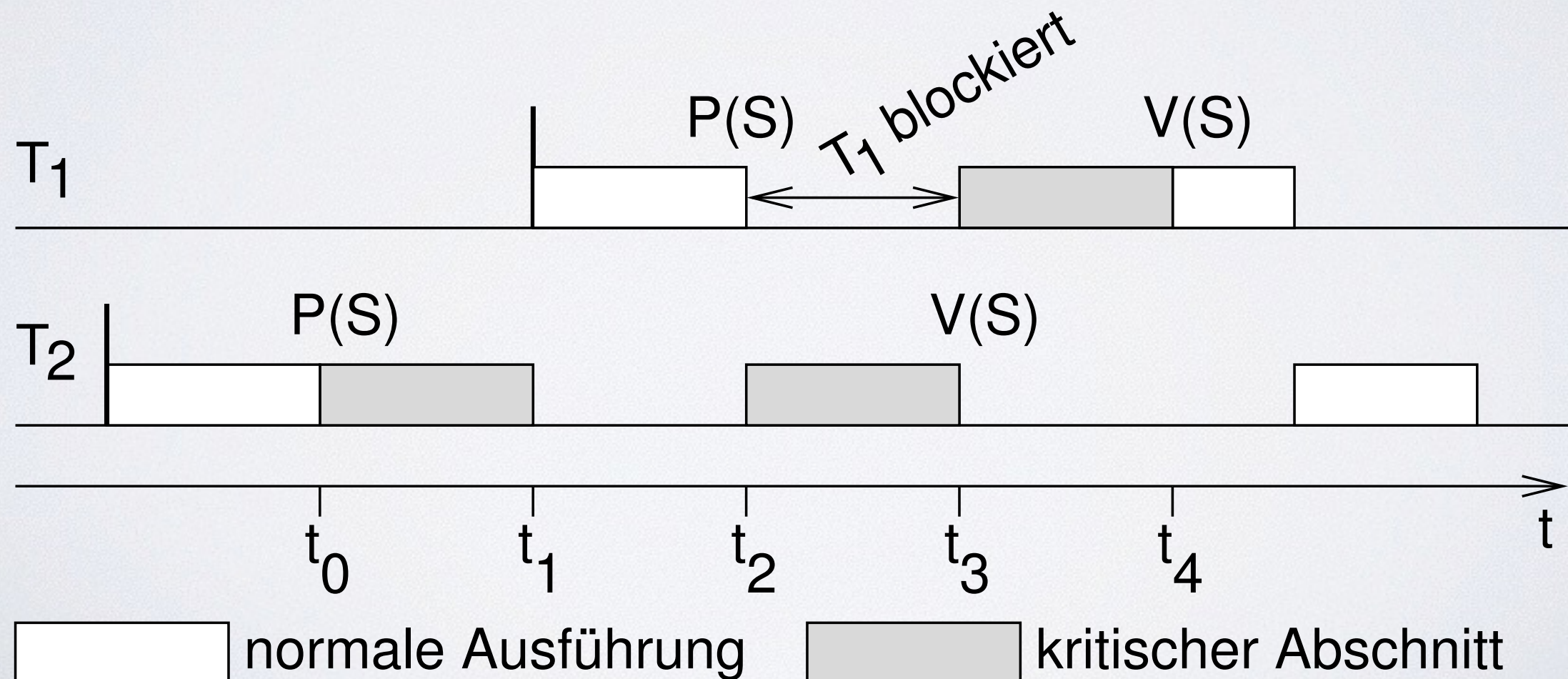
Earliest Deadline First bei periodischen Tasks

- Dynamische Prioritäten bei periodischen Tasks



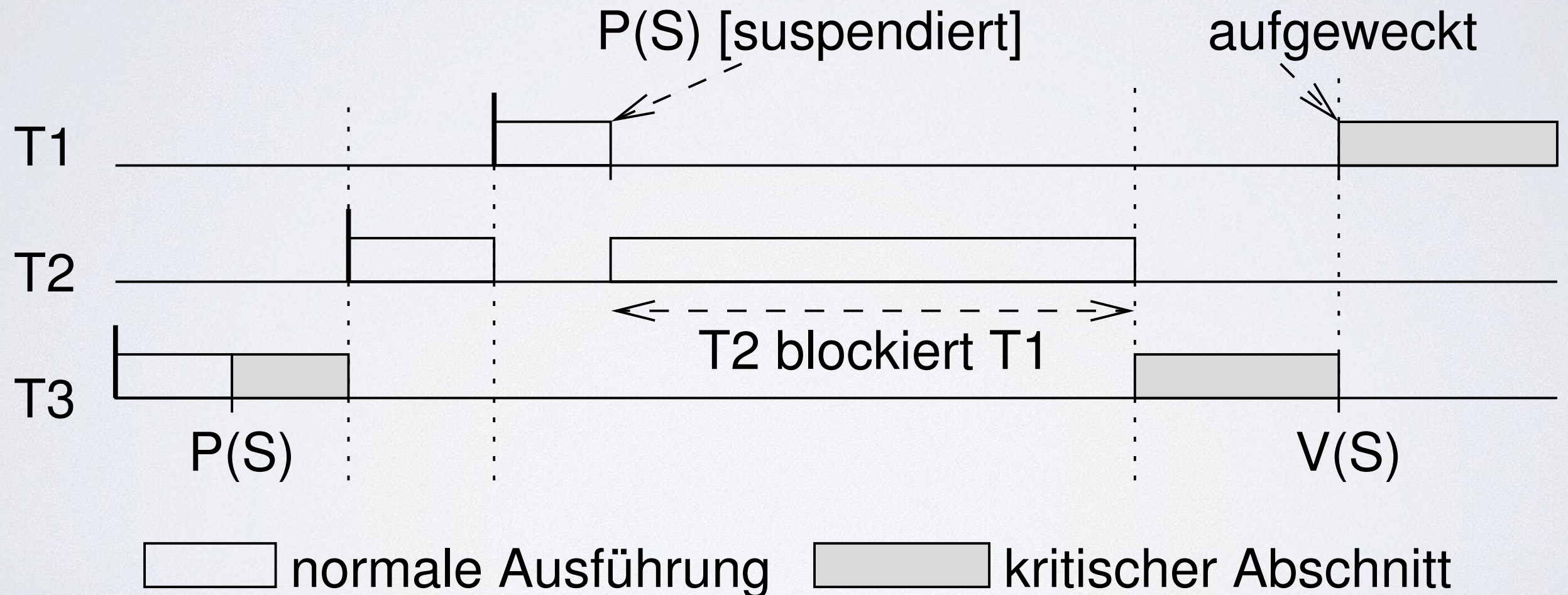
Ressourcen-Zugriffs-Kontrolle

- Kritische Abschnitte: Programmteile mit exklusiven Zugriff
- Mutex (Mutual exclusion) für Zugriffsregelung
 - Anforderung einer Ressource: $P(S)$, Freigabe einer Ressource: $V(S)$
- Gefahr einer Prioritätsumkehr bei kritischen Abschnitten



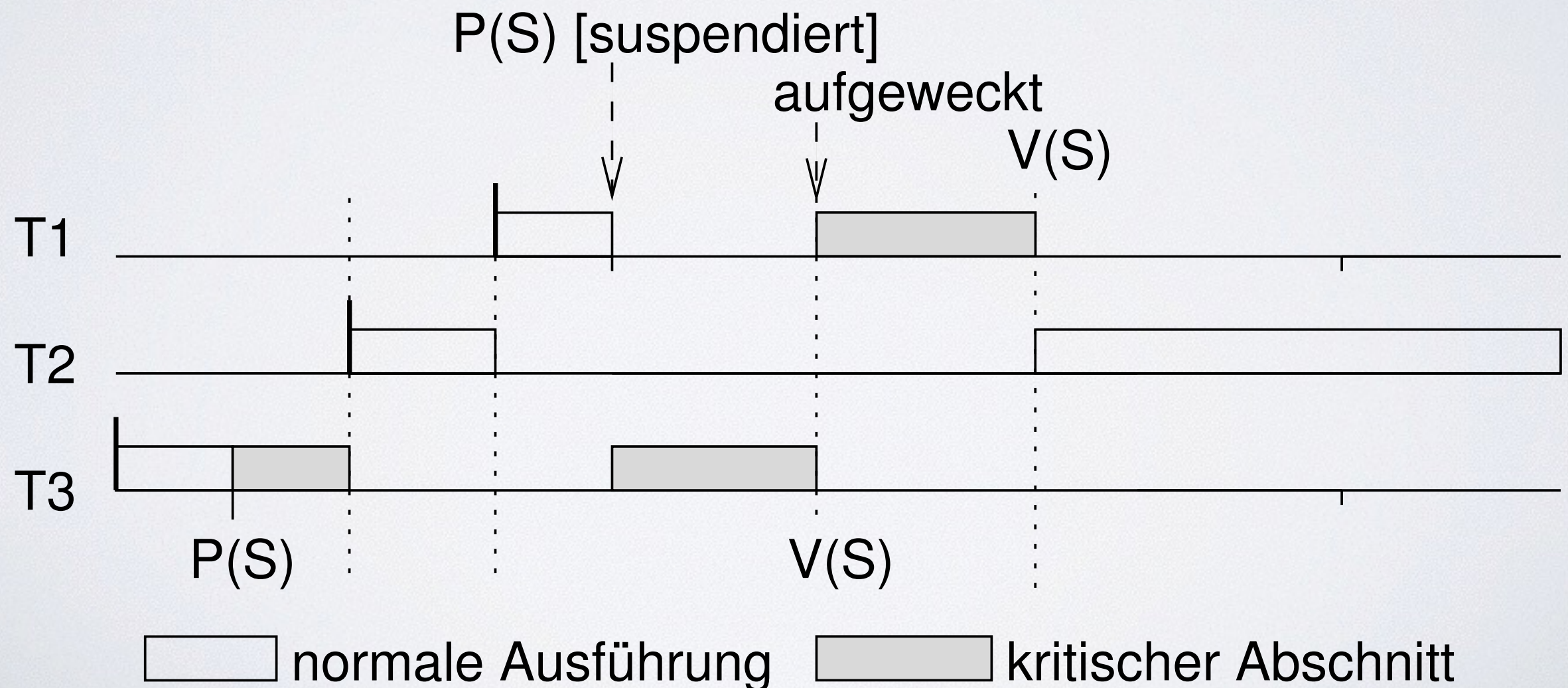
Ressourcen-Zugriffs-Kontrolle

- Prioritäten: $T1 > T2 > T3$



Ressourcen-Zugriffs-Kontrolle

- Prioritätsvererbung:
 - $P(S)$: Hochsetzen auf (nur höhere) Priorität von blockierten Tasks
 - $V(S)$: Herabsetzen auf (nur höhere) Priorität von blockierten Tasks oder auf eigene Priorität, Fortsetzung mit Task der höchsten Priorität



Zusammenfassung

- Scheduling und Echtzeit
- WCET
- Harte und weiche Echtzeit
- Diverse Scheduling-Verfahren
- Ressourcen-Zugriffs-Kontrolle

Beispiel einer Anwendung: Elektronische Waage

- Messung einer Verformung
bspw. einer Feder über
 - Dehnungsmessstreifen
oder
 - Kapazitätsänderung eines
Kondensators
- Digitalanzeige



Beispiel einer elektronischen Waage

[Quelle: Wikipedia: [http://de.wikipedia.org/w/index.php?title=Datei:Tischwaage_01_KM\].jpg&filetimestamp=20070421004816](http://de.wikipedia.org/w/index.php?title=Datei:Tischwaage_01_KM].jpg&filetimestamp=20070421004816)]

Literatur / Quellen

- Buttazzo, *Hard Real-Time Computing Systems*, Springer-Verlag 2011
- Körner, US Patent 6883446, *Quilting method and apparatus*, 11.02.2004
- Marwedel, *Eingebettete Systeme*, Springer-Verlag, 2008
- Wikipedia, *Waage*, <http://de.wikipedia.org/wiki/Waage>
- Wolf, *Computers as Components*, Morgan Kaufmann, 2012
- **Stand aller Internetquellen: 23.02.2014**