EMES: Eigenschaften mobiler und eingebetteter Systeme

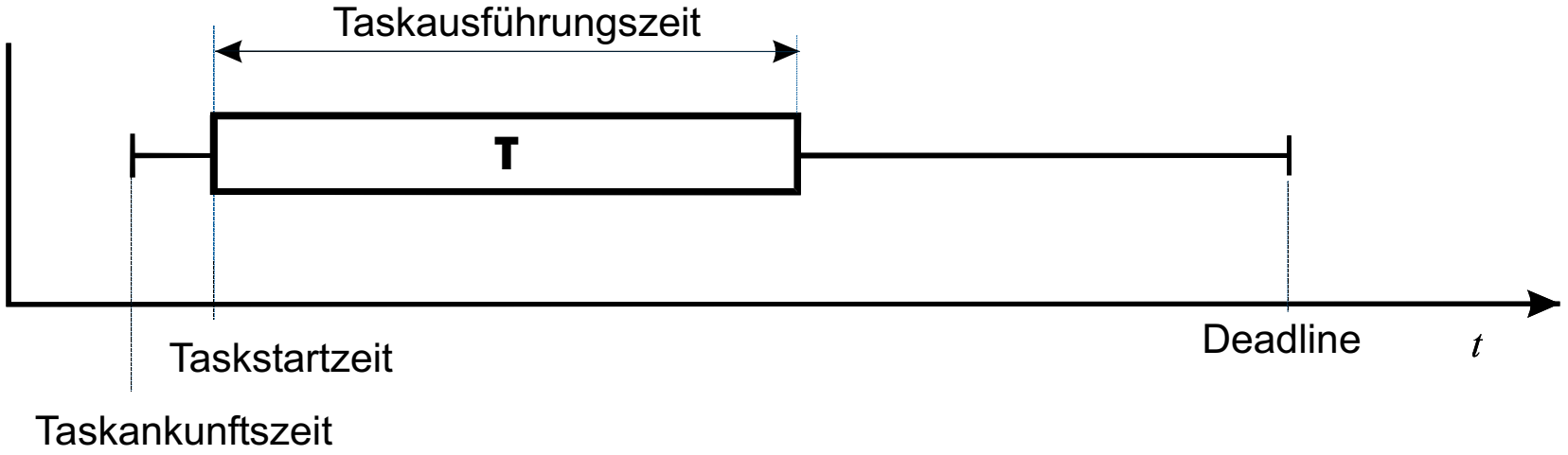
Prioritätsbasiertes Echtzeit-Scheduling

Dr. Felix Salfner, Dr. Siegmar Sommer

Wintersemester 2010/2011

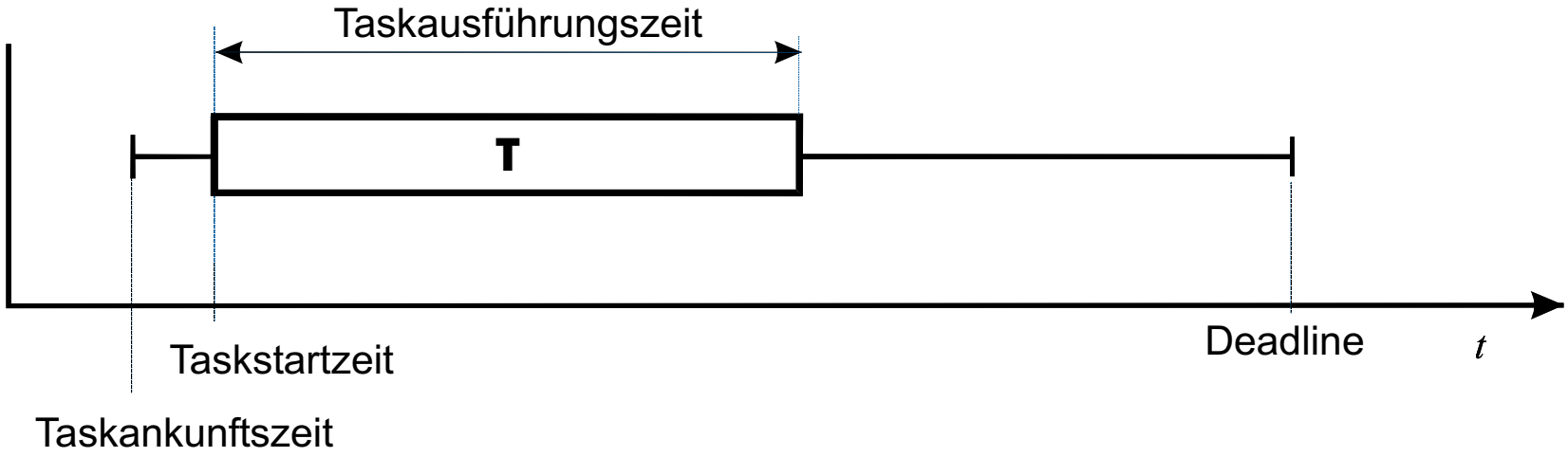
Wiederholung: Parameter einer Task

- Taskankunft r_i
- Taskstartzeit
- Ausführungszeit e_i
- Deadline D_i
- Periode P_i
- Maß für "Wichtigkeit" (Priorität)



Wiederholung: Taskmodell

- In der Regel ist nicht die wirkliche Ausführungszeit bekannt, sondern nur die längstmögliche Ausführungszeit (WCET)
- Je nach Betriebssystem können Tasks während ihrer Abarbeitung unterbrochen werden



Schedulingprobleme

Allgemeine Notation von Schedulingproblemen nach Lawler:

$$\alpha|\beta|\gamma$$

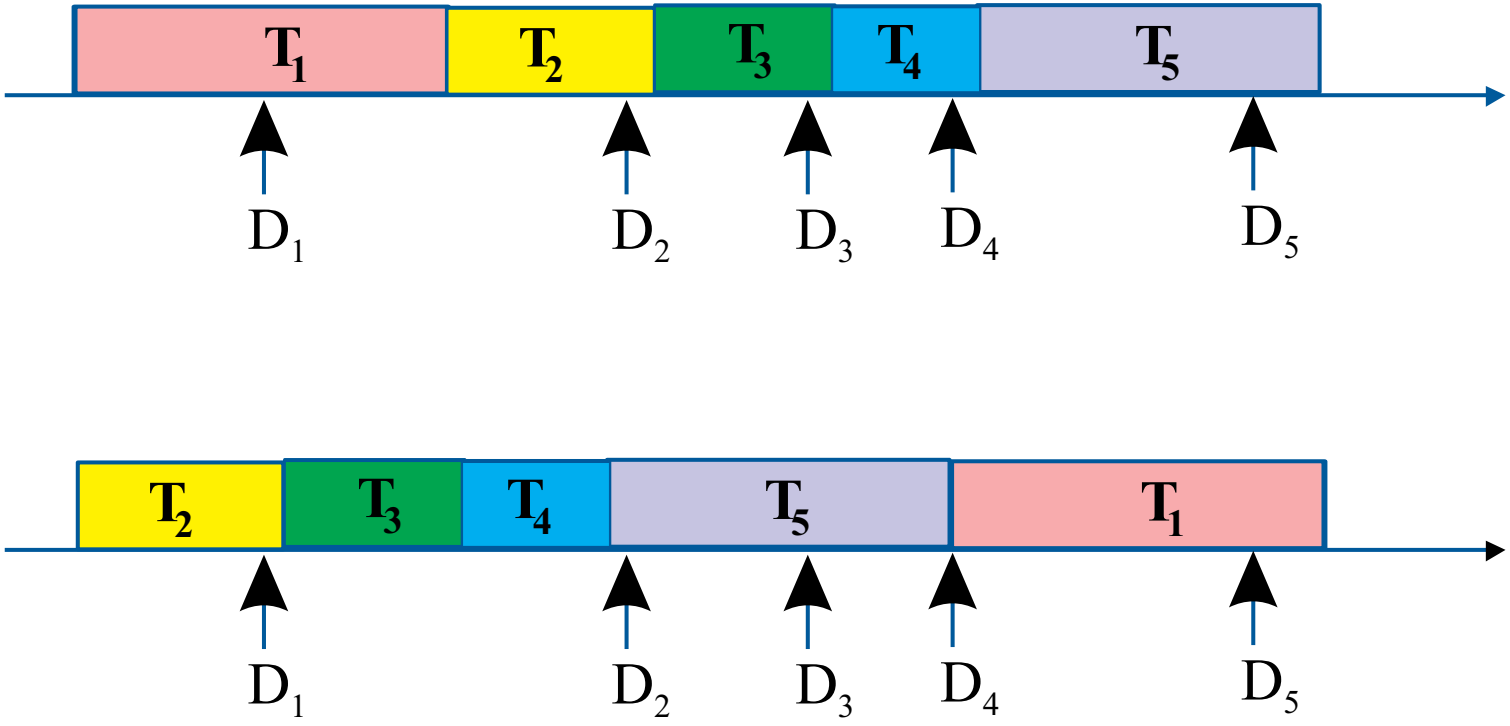
Bedeutung:

- α: Ausführungsumgebung, Ressourcen (z.B. Singleprozessor)
- β: Taskcharakteristiken (unterbrechbar/nicht unterbrechbar, unabhängig/abhängig, deadlinebehaftet, etc.)
- γ: Optimalitätskriterium (z.B. Minimierung der Verspätungen, Antwortzeit, Einhaltung der Deadlines, etc.)

Fast alle Probleme sind $\mathbb{NP}$!

Verschiedene Optimalitätsmetriken

Geringste maximale Start-Verzögerung vs. maximale Anzahl eingehaltener Deadlines



Problem des Echtzeitschedulings

- Gegeben sei eine Menge von Tasks
- Gesucht wird ein Ausführungsplan (Schedule), der es ermöglicht, daß alle Tasks immer ihre Deadline einhalten

  $\longrightarrow$ Tasks sind *ausführbar*.

- Wieder: Allgemein ist Problem $\mathbb{NP}$
- Unter bestimmten Einschränkungen (z.B. alle Task-Eigenschaften bekannt) gibt es jedoch einfache Algorithmen, die das Problem lösen

Scheduling für periodische Tasks

- Modell:
  - Standardzyklus: Sensor lesen – Wert verarbeiten – Effektor steuern
  - Mehrere Standardzyklen sind durch einen Prozessor zu bedienen
- Annahmen (Standardannahmen):
  - Jede Task ist ohne Kosten stets unterbrechbar
  - Tasks sind voneinander unabhängig, alle Ressourcen sind hinreichend vorhanden
  - Alle Tasks treten periodisch auf, Deadline = Periodenlänge
  - Konstante Periodenlänge und Ausführungszeit

Prioritätsbasierte Verfahren

Klassische Verfahren für Einprozessorsysteme, die Prioritäten nutzen, sind u.a.:

- Rate Monotonic Scheduling (RMS) (Liu und Layland)
- Earliest Deadline First (EDF) (Liu und Layland)
- Least Slack-Time First (LST, auch Minimum Laxity First, MLF) (Leung, Whitehead und Mok)

Alle prioritätsbasierten Verfahren haben gemeinsam, daß jeder Task eine Priorität zugewiesen wird.

Bei den genannten Verfahren wird immer die Task mit der augenblicklich höchsten Priorität ausgeführt.

Begriffe: Last

- Tasks erzeugen auf einen Prozessor eine gewisse Last (*load*, *utilization*)
- Lastbegriff ist schwierig im Nicht-Echtzeit-Bereich
- Bei genannten Annahmen einfache Definition:

$$\text{Last einer Task:} \quad U_i = \frac{e_i}{D_i}$$

$$\text{Gesamtlast:} \quad U = \sum_i U_i = \sum_i \frac{e_i}{D_i}$$

e_i: Ausführungszeit, D_i: Zeit bis zur Deadline

- Unter Standardannahmen: $D_i = P_i$: Periodenlänge
- $U = 1$ bedeutet, daß der Prozessor niemals unbeschäftigt (idle) ist.
- Lasten $U > 1$ sind nicht ausführbar.

Begriffe: Antwortzeit und kritischer Zeitpunkt

- Die *Antwortzeit* ist die Zeit zwischen der Taskankunft ($\neq$ Periodenbeginn) und der Beendigung der Ausführung (ACHTUNG - Unterschied zu Nicht-Echtzeitsystemen !).

- Ein *kritischer Zeitpunkt* ist die Ankunftszeit einer Task, bei der sie die längste Antwortzeit hat.

- Ein *kritisches Intervall* ist die Zeit zwischen dem kritischen Zeitpunkt einer Task und der Beendigung der entsprechenden Taskausführung.

Begriffe: Vollständige Ausnutzung

Eine Menge von Tasks $\mathcal{T}$ nutzt den Prozessor unter einem Scheduling-verfahren $\mathcal{S}$ vollständig aus (*fully utilize*), wenn jede Vergrößerung der Rechenzeit irgendeiner Task dazu führt, daß $\mathcal{T}$ nicht mehr ausführbar ist.

- Eine volle Ausnutzung bedeutet *nicht*, daß der Prozessor niemals unbeschäftigt (idle) ist
- Ausnutzung $\neq$ Last
- Häufig wird auch von einer *schwer schedulbaren* (*difficult to schedule*) Taskmenge gesprochen.

Variable und feste Prioritäten

Es werden zwei Klassen von prioritätenbasierten Schedulingverfahren unterschieden:

- Verfahren mit festen Prioritäten (*fixed priority scheduling*)
- Verfahren mit variablen Prioritäten (*variable priority scheduling*)

Verfahren mit festen Prioritäten sind leichter zu implementieren.

Rate Monotonic Scheduling (RMS)

- Offline-Verfahren
- RMS arbeitet mit festen Prioritäten
- Task erhält eine Priorität reziprok zu ihrer Periode (Deadline)
- Eine laufbereite Tasks mit höherer Priorität unterbricht stets eine Task niederer Priorität
- Optimaler Algorithmus für feste Prioritäten

Literatur zu RMS

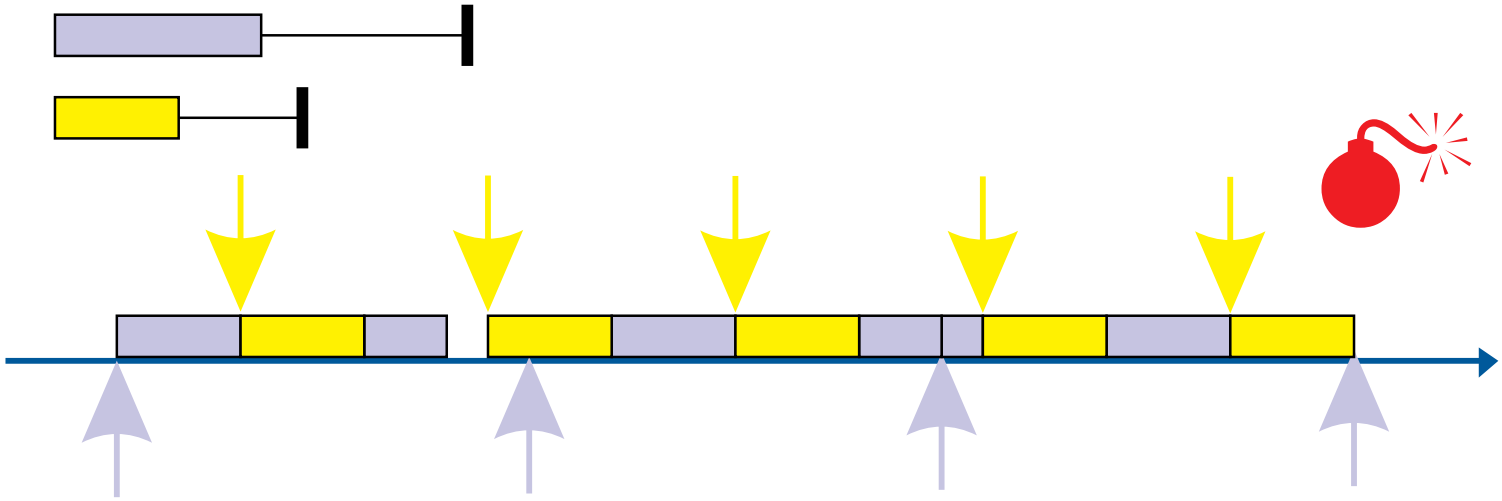
- Jane W. S. Lui: Real-Time Systems, Abschnitte 6.4 und 6.7
- C.M. Krishna und K.G. Shin: Real-Time Systems, Abschnitt 3.2.1
- C.L. Lui und J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

Beispiel für RMS-Schedule

- Task 1: $P_1 = D_1 = 2$, $e_1 = 0.5$, $r_1 = 0$
- Task 2: $P_2 = D_2 = 6$, $e_2 = 2.0$, $r_2 = 1$
- Task 3: $P_3 = D_3 = 10$, $e_3 = 1.75$, $r_3 = 3$
- Priorität von Task 1 am höchsten

Beispiel für RMS-Schedule

- Task 1: $P_1 = D_1 = 6$, $e_1 = 3$
- Task 2: $P_2 = D_2 = 10$, $e_2 = 5$
- Task 1 hat höhere Priorität und wird bei $T = 3$ bereit
- nicht ausführbar mit RMS (aber mit EDF)



Kritischer Zeitpunkt bei RMS

Theorem 1. *Der kritische Moment für eine Task tritt dann ein, wenn sie zeitgleich mit allen höherpriorisierten Tasks angefordert wird.*

Beweis.

- Sei $prio_1 > prio_2$ für T_1, T_2; T_2 kommt zum Zeitpunkt r_2 an
- T_1 kommt zu $r_1 = r_2, r_1 + t_1, r_1 + 2t_1, \ldots$ an
- Die Antwortzeit von T_2 erhöht sich um $\lceil \frac{e_2}{t_1 - e_1} \rceil * e_1$
  (Ausgefüllte Slack-Slots $*$ Ausführungszeit von e_1)
- Verschieben von r_2 verringert den Faktor oder lässt ihn konstant

□

- Beispiel: $e_1 = 3, e_2 = 10, t_1 = 7 \rightarrow$ Verzögerung um 9

Folgerung

- Worst Case eines Schedule: Alle Tasks mit dem kritischen Moment als Ankunftszeit $\longrightarrow$ nur maximale Antwortzeiten
- Reicht für Betrachtung der Ausführbarkeit eines Schedule
- Liu u. Leyland Beispiel: $e_1 = 1, e_2 = 1, P_1 = 2, P_2 = 5$
  - Egal welcher Task die höhere Priorität hat, der Schedule ist gültig

Optimalität von RMS

Theorem 2. *Unter den Standardannahmen kann RMS für die Taskmenge $\mathcal{T}$ genau dann einen ausführbaren Schedule erzeugen, wenn irgendein anderes* Fixed-priority-*Verfahren einen ausführbaren Schedule für $\mathcal{T}$ findet.*

Insgesamt ist RMS **nicht** optimal, da es Taskmengen geben kann, die mit RMS nicht ausführbar sind, wohl aber mit anderen Schedulingverfahren $\mathcal{M}$.

Aber: $\mathcal{M}$ kann kein Fixed-Priority-Verfahren sein.

Beweis von Theorem 2

- $\mathcal{M}$ sei Fixed-Priority-Verfahren, das ein Taskmenge $\mathcal{T}$ ausführbar plant; der entstehende Schedule sei $\mathcal{S}$
- Überführung von $\mathcal{S}$
- In $\mathcal{S}$ gelte $p_j = p_i + 1$ bei $P_i > P_j$ (p_x bezeichne Prioritäten, wobei eine kleinere Zahl „wichtiger“ bedeutet)
- Austausch von p_i und p_j läßt den Schedule ausführbar
- RMS Schema ist von beliebigen S durch mehrfaches paarweises Vertauschen ableitbar

□

Schedulingkriterium für RMS

- Untere Schranke der oberen Grenze

Theorem 3. *Eine Menge $\mathcal{T}$ von n Tasks ist mit RMS unter den Standardbedingungen immer dann ausführbar, wenn*

$$U \leq n(2^{\frac{1}{n}} - 1)$$

- Hinreichend, aber nicht notwendig
- Beweis siehe Paper

n	1	2	5	10	50	100
U	1,0	0,828	0,743	0,718	0,698	0,695

Grenzwert für Lastschranke

$$
\begin{aligned}
U_\infty &= \lim_{n\to\infty} n\left(2^{\frac{1}{n}} - 1\right) \\
&= \lim_{n\to\infty} \frac{2^{\frac{1}{n}} - 1}{\frac{1}{n}} \\
&= \lim_{n\to\infty} \frac{\frac{d}{dn}\left(2^{\frac{1}{n}} - 1\right)}{\frac{d}{dn}\left(\frac{1}{n}\right)} \\
&= \lim_{n\to\infty} \frac{2^{\frac{1}{n}} \cdot \ln 2 \cdot -\frac{1}{n^2}}{-\frac{1}{n^2}} \\
&= \ln 2 \\
U_\infty &\approx 0,693
\end{aligned}
$$

Analyse der Zeitforderungen

- Theorem 3 ist nur eine Schranke
- Tatsächliche Grenze liegt meist viel besser
- Statistischer Mittelwert liegt etwa bei 0,88
- Genauere Schedulingtests mit *Analyse der Zeitanforderungen (Time-Demand Analysis)*

Analyse der Zeitforderungen (II)

- Time-Demand Analysis (TDA) wurde von Lehoczky, Sha und Ding 1989 vorgestellt
- TDA kann als Test für alle Fixed-Priority-Schedulingverfahren angewandt werden, wenn die Deadline nicht größer als die Periode ist.
- Für viele wichtige Fälle liefert TDA hinreichendes und notwendiges Kriterium
- Basiert wieder auf Analyse vom kritischen Zeitpunkt

Schedulingbedingung nach TDA

- Die Zeitanforderungsfunktion (*time demand function*) einer Task T_i wird mit $w_i(t)$ bezeichnet.
- $w_i(t)$ ist wie folgt definiert:

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{P_k} \right\rceil \cdot e_k$$

- Tasks sind nach Prioritäten geordnet, T_1 hat die höchste Priorität.
- Bestandteile: Eigene Ausführungszeit, Häufigkeit und Länge der Unterbrechung durch Tasks mit höherer Priorität
- Mit steigendem t wächst auch $w_i(t)$
- Um Deadline D_i einzuhalten, muss ein t existieren, so dass $w_i(t) \leq D_i$

TDA Theorem

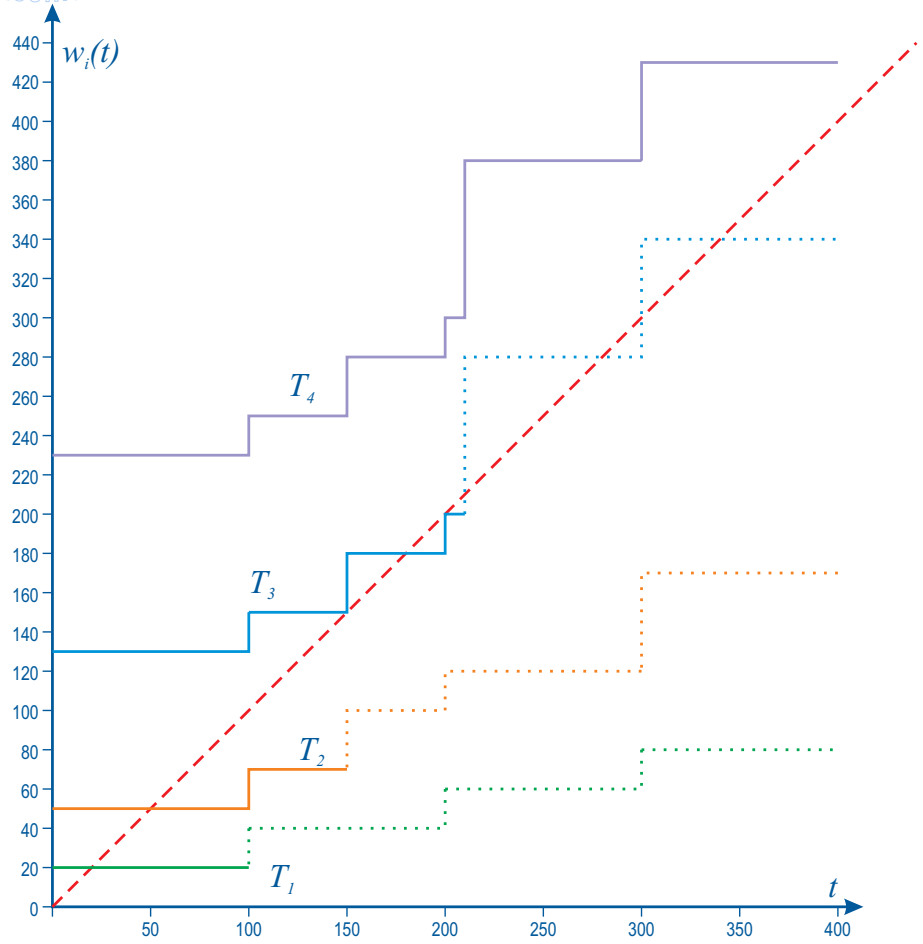
Theorem 4. *Eine Menge $\mathcal{T}$ von unterbrechbaren, periodischen und unabhängigen Tasks ist mit einem Fixed-Priority-Verfahren $\mathcal{S}$ ausführbar, wenn*

$$\forall i : (\exists t, 0 < t \leq P_i : w_i(t) \leq t)$$

- w_i einer Taskmenge sind rekursiv berechenbar
- $w_i(0) = 0$ und $w_{i+1}(t) = e_{i+1} + (\left\lceil \frac{t}{P_i} \right\rceil - 1) + w_i(t)$
- Vereinfachung erlaubt Aussage, wieviel Zeit von allen Instanzen einer periodischen Task bis t verbraucht wird

Beispiel für TDA

	e_i	P_i
T_1	20	100
T_2	30	150
T_3	80	210
T_4	100	400



- TDA-Funktion muss bei Deadline unterhalb der t-Funktion liegen
- Betrachtung nur da nötig, wo höherpriorisierte Tasks anstehen

Variationen von RMS

Betrachten Änderungen der Grundbedingungen:

- RMS für einfach periodische Taskmengen
- Deadline Monotonic Scheduling (DMS)

RMS für einfach periodische Taskmengen

Eine Menge von Tasks $\mathcal{T}$ nennt man *einfach periodisch*, wenn für jedes Paar von Tasks T_i und T_j mit $P_i < P_j$ gilt, daß P_j ein ganzzahliges Vielfache von P_i ist.

Theorem 5. *Unter den Standardannahmen kann RMS für jede einfach periodische Taskmenge $\mathcal{T}$, genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für $\mathcal{T}$ existiert.*

RMS für einfach periodische Taskmengen nennt man auch $harmonisches\ RMS$.

RMS-Schedulingbedingung für einfach periodische Tasksets

Theorem 6. [Schedulingbedingung] *Eine Menge $\mathcal{T}$ von einfach periodischen, unabhängigen und unterbrechbaren Tasks T_i, in der für alle T_i gilt: $D_i \leq P_i$, ist genau dann mit RMS ausführbar, wenn*

$$\sum_{i=0}^{n} \frac{e_i}{P_i} \leq 1$$

Deadline Monotonic Scheduling (DMS)

- DMS ist eine Verallgemeinerung von RMS
- Es gilt nicht mehr $D_i = P_i$
- Statische Zuweisung von Prioritäten entsprechend ihrer relativen Deadline
- Die Tests zur Ausführbarkeit sind komplexer als bei RMS.
- Für $D_i = P_i$ gilt DMS = RMS

Optimalität von DMS

Bei beliebigen Deadlines gilt: DMS ist RMS überlegen

- Es gibt Taskmengen, die mit DMS, aber nicht mit RMS ausführbar sind.
  Verdeutlichung: Die Ausführbarkeit kann getestet werden, indem die Periode gleich der Deadline gesetzt wird, das führt aber zu einer zu hohen Lastabschätzung.

- Es gibt keine Taskmengen, die mit RMS, aber nicht mit DMS ausführbar sind

Theorem 7. *DMS ist unter den Fixed-Scheduling-Verfahren mit beliebigen Deadlines optimal.*

Earliest Deadline First (EDF)

- EDF arbeitet mit variablen Prioritäten
- Jede Task erhält eine Priorität entsprechend ihrer Deadline: Eine Task mit einer kürzeren Deadline hat stets eine höhere Priorität als eine Task mit einer längeren Deadline.
- Eine laufbereite Task mit höherer Priorität unterbricht stets eine Task niederer Priorität

Literatur zu EDF

- Jane W. S. Lui: Real-Time Systems, Abschnitte 6.3
- C.M. Krishna und K.G. Shin: Real-Time Systems, Abschnitt 3.2.2
- C.L. Lui und J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

Optimalität von EDF

Theorem 8. *Unter den Standardannahmen kann EDF für die Taskmenge $\mathcal{T}$ genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für $\mathcal{T}$ existiert.*

Anders gesagt: Wenn irgendein Schedulingverfahren $\mathcal{S}$ unter den Standardbedingungen für $\mathcal{T}$ einen ausführbaren Schedule erzeugen kann, dann kann EDF das auch.

Schedulingkriterium für EDF

Problem: Gibt es ein Kriterium, welches feststellt, ob eine Taskmenge $\mathcal{T}$ unter EDF ausführbar ist, ohne daß das Verfahren vollzogen wird?

Hinreichende und notwendige Schedulingbedingung für EDF:

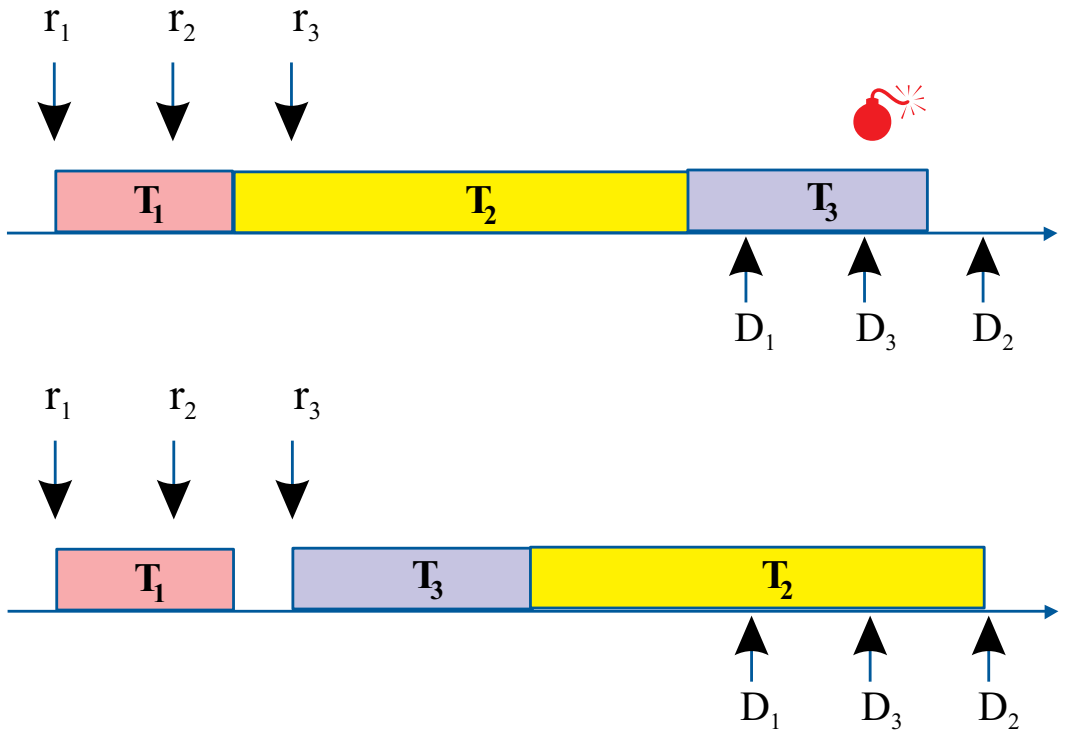
Theorem 9. *Ein Menge $\mathcal{T}$ von n Tasks ist mit EDF unter den Standardannahmen genau dann ausführbar, wenn*

$$U_{\mathcal{T}} = \sum_{i=1}^{n} \frac{e_i}{P_i} \leq 1$$

Mit anderen Worten: EDF ist immer erfolgreich, wenn die Last der Taskmenge kleiner/gleich 1 ist.

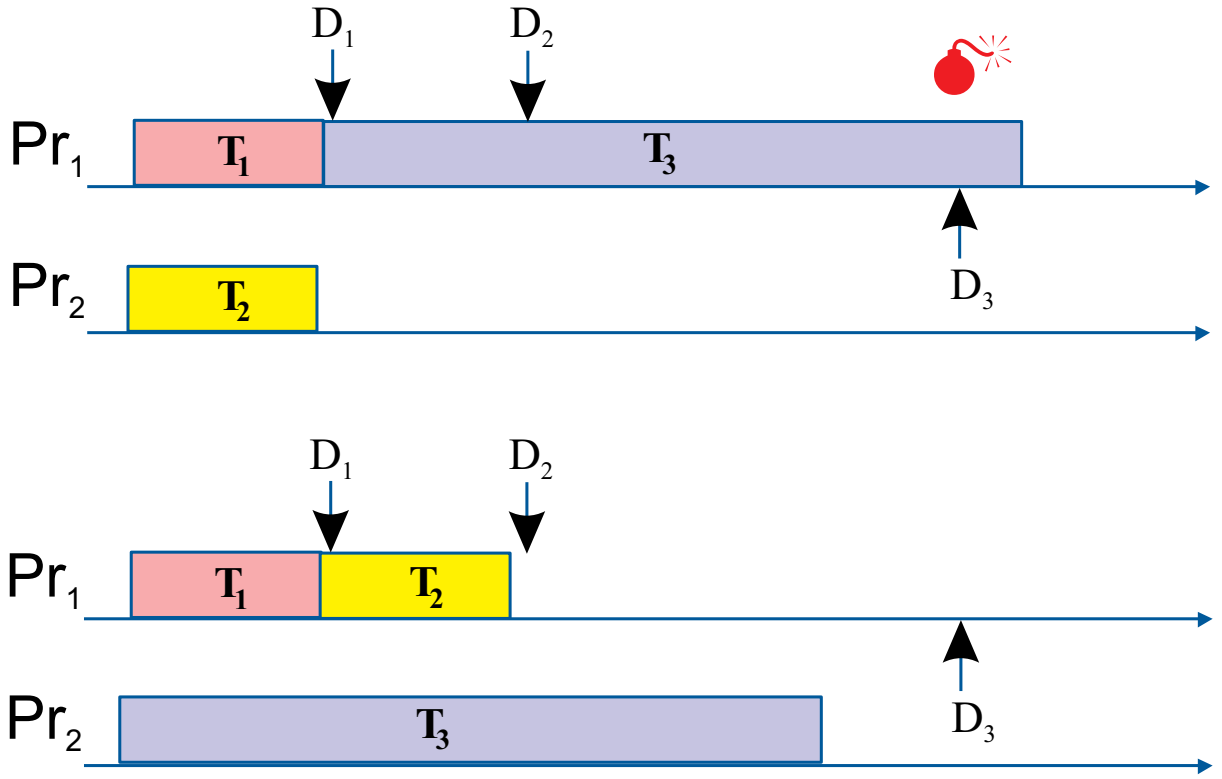
Notwendigkeit der Standardbedingungen (I)

- Optimalität von EDF gilt nur für die Standardbedingungen
- Beispiel: Nichtunterbrechbare Tasks



Notwendigkeit der Standardbedingungen (II)

EDF ist ebenfalls nicht optimal bei mehreren Prozessoren:



Allgemeines EDF

EDF bleibt optimal, wenn die Bedingung $D_i = P_i$ weggelassen wird.

Theorem 10. *Gilt für eine Taskmenge $\mathcal{T}$ die Ungleichung*

$$U_{\mathcal{T}} = \sum_{i=1}^{n} \frac{e_i}{\min\left(D_i, P_i\right)} \leq 1$$

ist $\mathcal{T}$ mit dem EDF-Algorithmus ausführbar.

- Wenn $\forall i : D_i \geq P_i$, dann ist dies eine notwendige und hinreichende Bedingung.
- Wenn wenigstens eine Task T_k existiert mit $D_k < P_k$, dann nur hinreichende Bedingung.

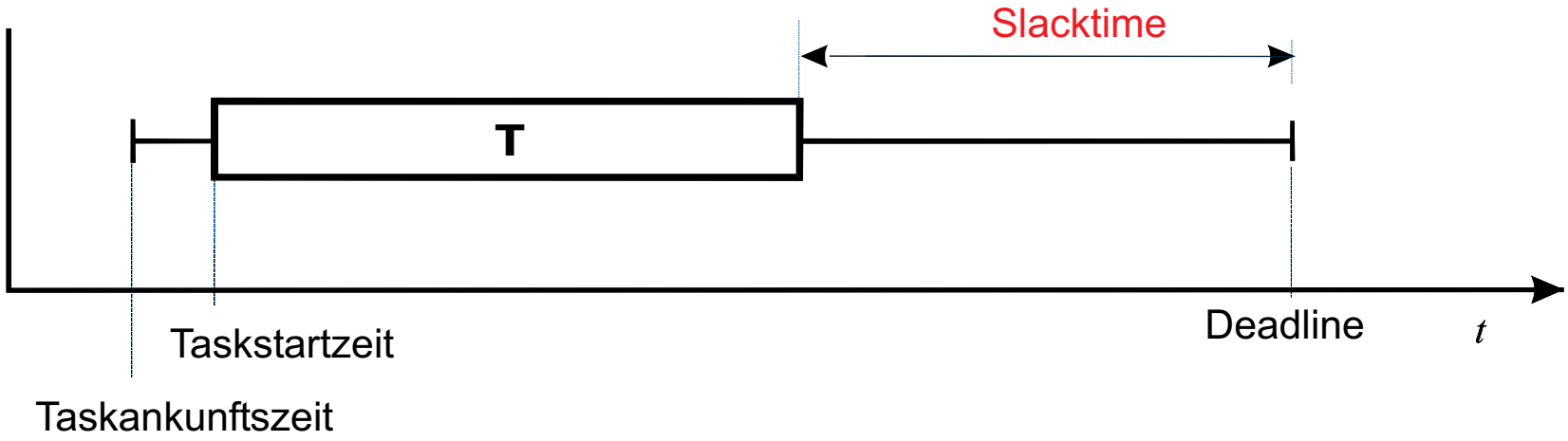
Least Slacktime First (LSF)

- LSF arbeitet mit variablen Prioritäten
- Jede Task erhält eine Priorität entsprechend der Differenz zwischen Deadline und (restlicher) Ausführungszeit (*Slacktime*).

  Kürzere Slacktime = höhere Priorität.
- Eine laufbereite Tasks mit höherer Priorität unterbricht stets eine Task niederer Priorität.

Slacktime

- Slacktime, auch *Laxity*



Optimalität von LSF

Theorem 11. *Unter den Standardannahmen kann LSF für die Taskmenge $\mathcal{T}$ genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für $\mathcal{T}$ existiert.*

Anders gesagt: Wenn irgendein Schedulingverfahren $\mathcal{S}$ unter den Standardbedingungen für $\mathcal{T}$ einen ausführbaren Schedule erzeugen kann, dann kann LSF das auch.

Beweis: Ähnlich dem für EDF.

Praktische Probleme

Die Standardbedingungen sind in der Praxis nicht (immer) einzuhalten.

Probleme ergeben sich durch

- Endliche Zeiten für Context-Wechsel
- Beschränkte Anzahl von Prioritätsleveln
- Diskrete Timer-Ticks für Scheduling