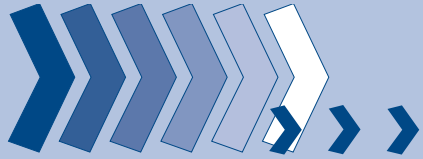
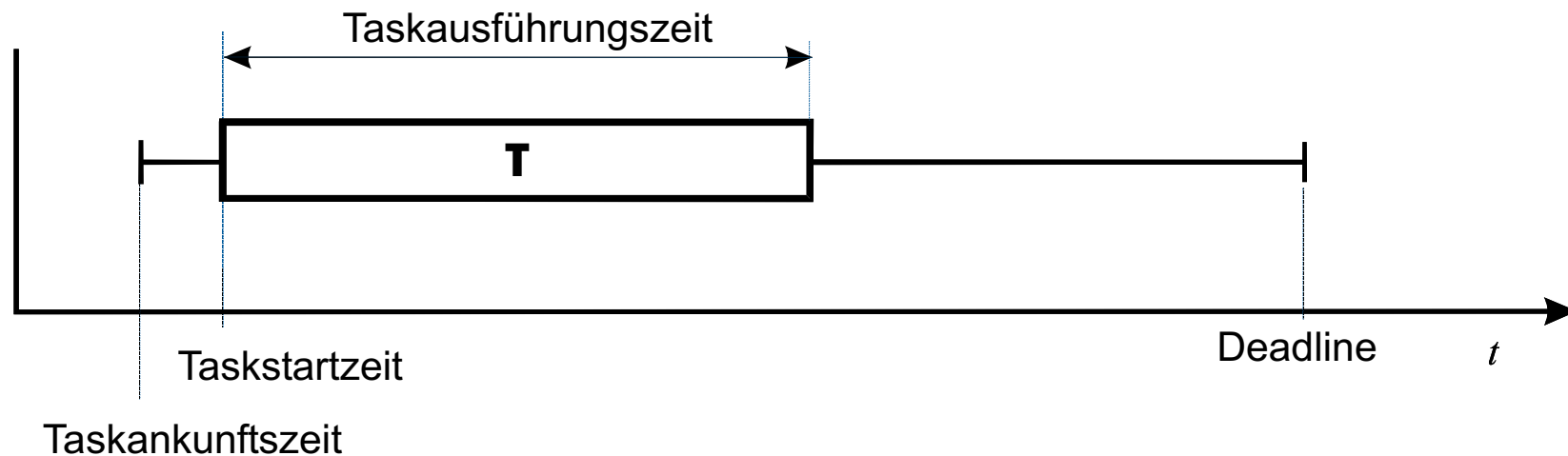


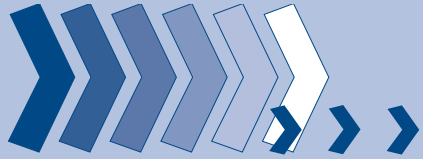
Prioritätsbasiertes Scheduling



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





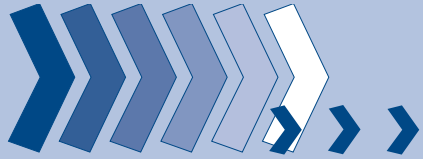
Allgemeine Notation von Schedulingproblemen nach LAWLER:

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

Bedeutung:

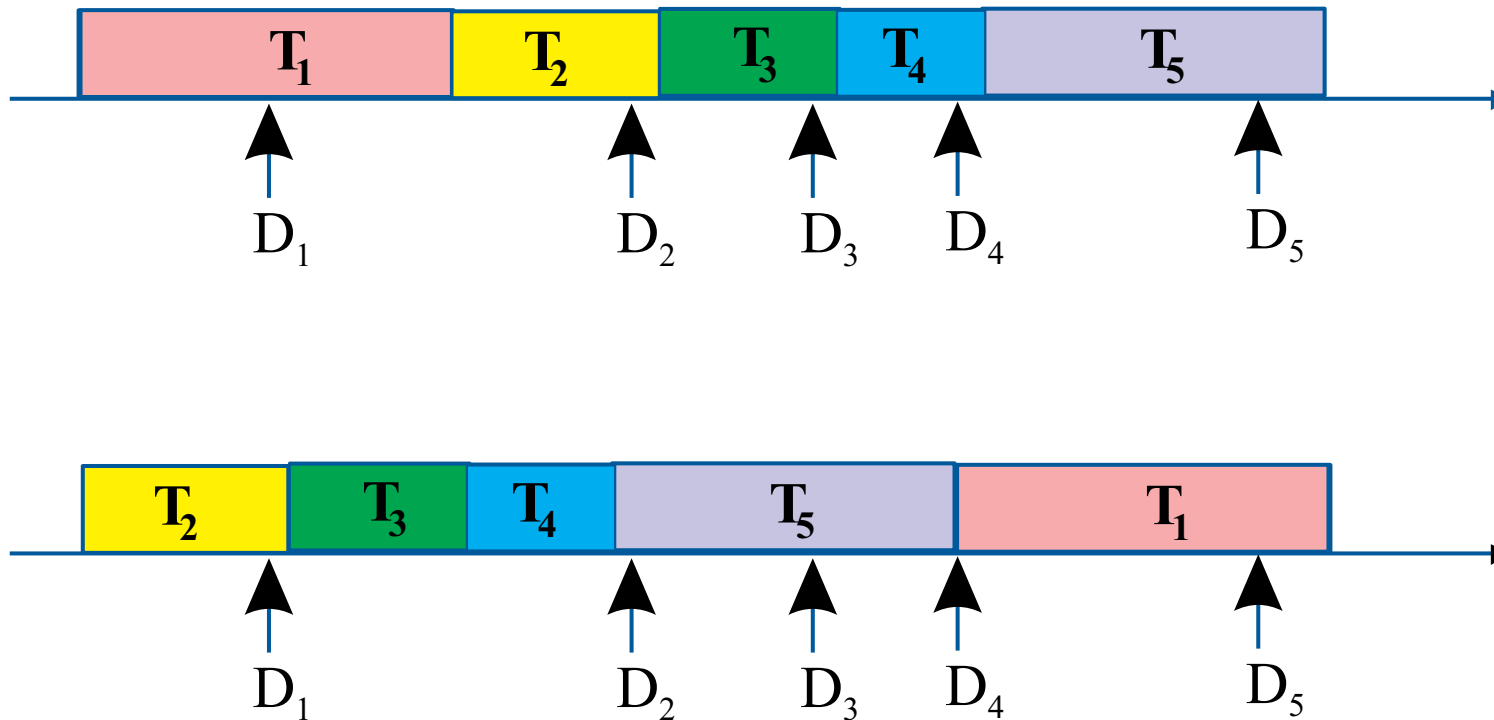
- α : Ausführungsumgebung, Ressourcen (z.B. Singleprozessor)
- β : Taskcharakteristiken (unterbrechbar/nicht unterbrachbar, unabhängig/abhängig, deadlinebehaftet, etc.)
- γ : Optimalitätskriterium (z.B. Minimierung der Verspätungen, Antwortzeit, Einhaltung der Deadlines, etc.)

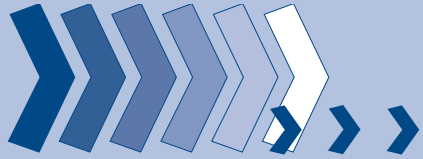
Pech: Fast alle Probleme sind NP



Verschiedene Optimalitätsmetriken

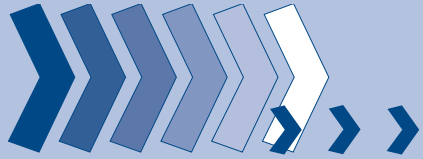
Geringste maximale Verzögerung vs. maximale Anzahl eingehender Deadlines





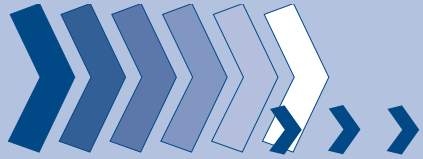
Problem des Echtzeitschedulings

- Gegeben sei eine Menge von Tasks
- Gesucht wird ein Ausführungsplan (Schedule), der es ermöglicht, daß alle Tasks immer ihre Deadline einhalten
→ Tasks sind *ausführbar*.
- Wieder: allgemein ist Problem NP
- Unter bestimmten Einschränkungen 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 stets unterbrechbar, und die Unterbrechungskosten sind vernachlässigbar
 - Tasks sind voneinander unabhängig, alle Ressourcen sind hinreichend vorhanden
 - Alle Tasks treten periodisch auf
 - Deadline = Periodenlänge



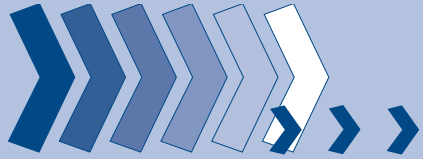
Prioritätsbasierte Verfahren

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, WHITE-HEAD 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.



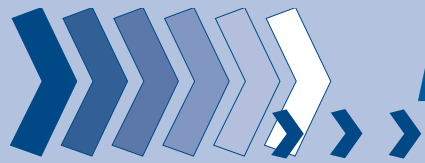
- Tasks erzeugen auf einen Prozessor eine gewisse Last (*load, utilization*)
- Lastbegriff ist schwierig in Nicht-Echtzeit-Bereich
- Bei genannten Annahmen einfache Definition:

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

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

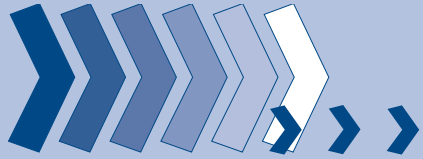
e_i : Ausführungszeit, D_i : Zeit bis zur Deadline

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



Begriffe: Antwortzeit und kritischer Zeitpunkt

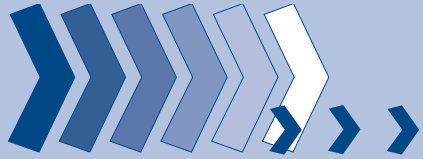
- Die *Antwortzeit* ist die Zeit zwischen der Taskankunft und der Beendigung der Ausführung.
- Ein *kritischer Zeitpunkt* ist die Ankunftszeit einer Task, bei der sie die längste Antwortzeit hat.
- Ein *kritisches Intervall* ist die Zeit zwischen dem kritischen Zeitpunkt einer Task und der Beendigung der entsprechenden Taskausführung.



Begriffe: Vollständige Ausnutzung

Eine Menge von Tasks \mathcal{T} nutzt den Prozessor unter einem Schedulingverfahren \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.

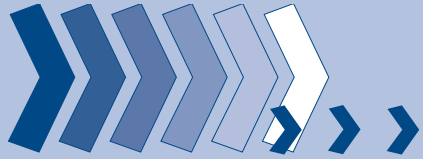


Variable und feste Prioritäten

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.

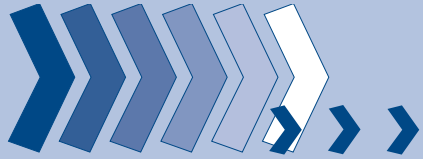


Earliest Deadline First (EDF)

- 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 niedrigerer Priorität



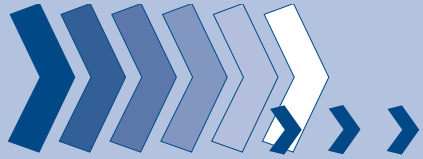
- Jane W. S. Lui: Real-Time Systems, Abschnitte 6.3
- C.M. Krishna und K.G. Shin: Real-Time Systems, Abschnitt 3.2.2
- C.L. Lui und J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, siehe Semesterapparat EMES 2001



Optimalität von EDF

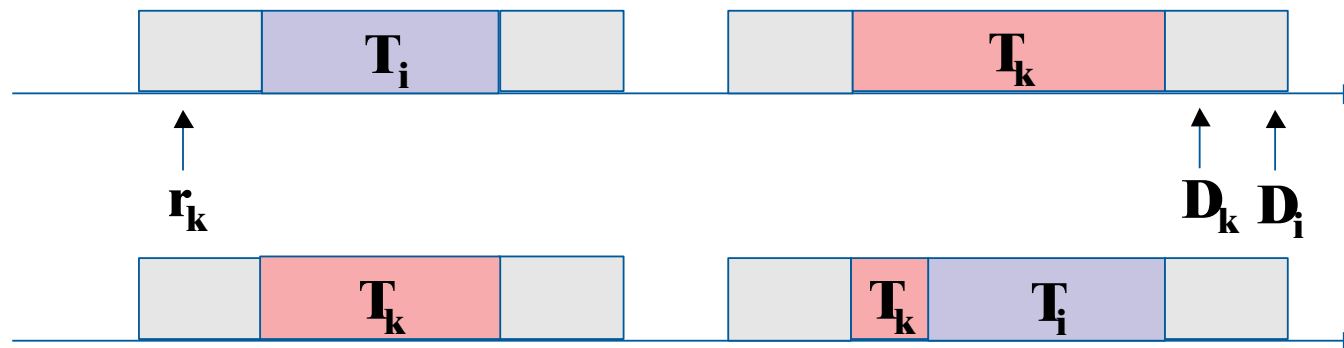
Theorem 0.1. *Unter den Standardannahmen kann EDF für die Taskmenge \mathcal{T} genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für \mathcal{T} existiert.*

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



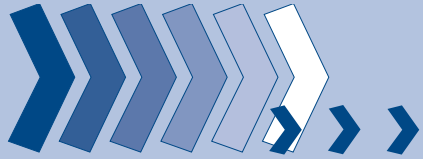
Überführung eines Nicht-EDF-Schedules in einen EDF-Schedule

- Austausch der Tasks innerhalb ihrer bisherigen Timeslots (nur Zeitanteile nach der Taskankunft von T_k)



- Wiederholen für alle Tasks.
- Verschieben der Idle-Zeiten





Schedulingkriterium für EDF

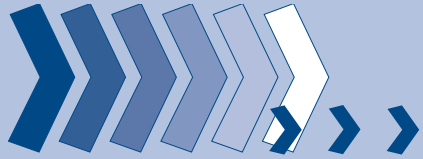
Problem: Gibt es ein Kriterium, welches feststellt, ob eine Taskmenge \mathcal{T} unter EDF ausführbar ist, ohne daß das Verfahren vollzogen wird?

Hinreichende und notwendige Schedulingbedingung für EDF:

Theorem 0.2. *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.



Beweis für Theorem 0.2

- Notwendigkeit: offensichtlich
- Hinlänglichkeit: Annahme des Gegenteils: Taskset \mathcal{T}' sei nicht ausführbar und habe $U \leq 1$

Benutzen folgendes Lemma:

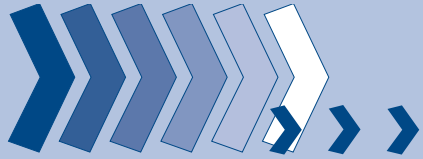
Lemma 0.2.1. *Vor einem Überlauf gibt es bei EDF-Scheduling keine Idle-Zeiten.*

Es sei $T_{i,k}$ die k -te Instanz von T_i und die erste Task, die ihre Deadline verpaßt, und zwar zum Zeitpunkt t . Die Ankunftszeit sei $r_{i,k}$

Fallunterscheidung

1. Fall: Keine Task, die im Intervall $[0, t]$ ausgeführt wird, hat eine Deadline *nach* t .

Da es zu einem Überlauf kommt, ist der Prozessor nie idle (Lemma 0.2.1). Folglich ist die von Task T_i im Intervall $[0, t]$ benötigte Zeit größer als die Länge des Intervalls.



Beweis für Theorem 0.2 (Fort.)

Eine beliebige in $[0, t]$ ausgeführte Task T_j benötigt $\left\lfloor \frac{t}{P_j} \right\rfloor \cdot e_j$ Zeit.

$$\left\lfloor \frac{t}{P_1} \right\rfloor \cdot e_1 + \left\lfloor \frac{t}{P_2} \right\rfloor \cdot e_2 + \cdots + \left\lfloor \frac{t}{P_n} \right\rfloor \cdot e_n > t$$

\Rightarrow

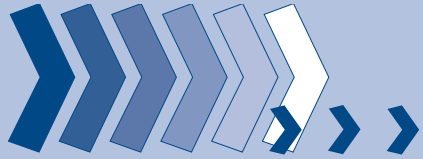
$$\left(\frac{t}{P_1} \right) \cdot e_1 + \left(\frac{t}{P_2} \right) \cdot e_2 + \cdots + \left(\frac{t}{P_n} \right) \cdot e_n > t$$

\Rightarrow

$$\sum_{i=1}^n \frac{e_i}{P_i} = U_{ges} > 1$$

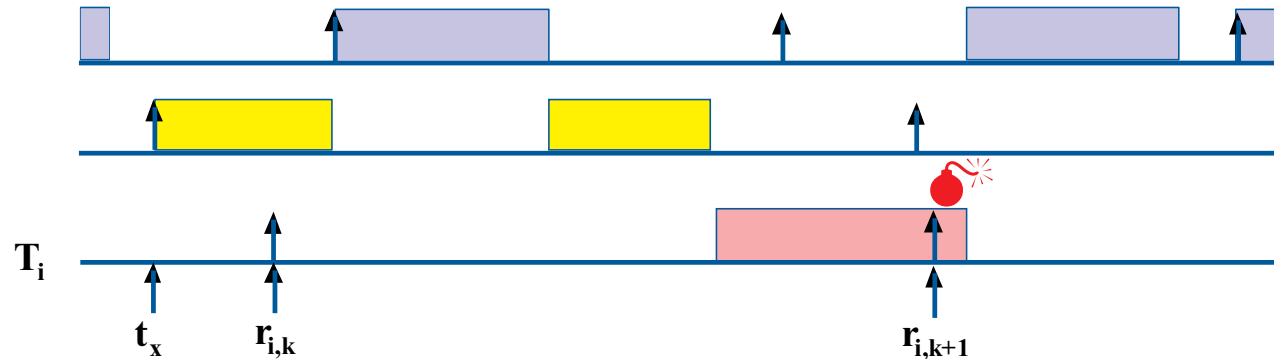
Dies steht im Widerspruch zur Annahme.





Beweis für Theorem 0.2 (Fort.)

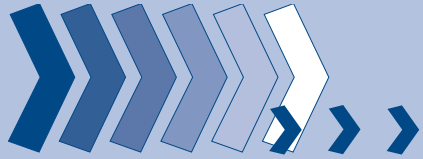
Fall 2: Mindestens eine Task im Intervall $[0, t]$ hat eine Deadline *nach* t .



Es sei t_x der letzte Zeitpunkt vor t , zu dem eine Task mit einer Deadline *nach* t ausgeführt wird.

Wenn zu t_x eine Task mit einer Deadline *nach* t ausgeführt wird, heißt das, daß es zu t_x keine Task mit einer Deadline *vor* t gibt.

Folglich kommen alle Tasks, die im Intervall $[t_x, t]$ ausgeführt werden, erst nach t_x an.



Beweis für Theorem 0.2 (Fort.)

Da $T_{i,k}$ seine Deadline verpaßt, und der Prozessor laut 0.2.1 vorher nicht idle ist, muß die Gesamtforderung nach Rechenzeit im Intervall $[t_x, t]$ größer sein als die Intervalldauer.

$$t - t_x < \sum_{T_j \in \mathcal{T}_c} \left\lceil \frac{t - t_x}{P_j} \right\rceil \cdot e_j$$

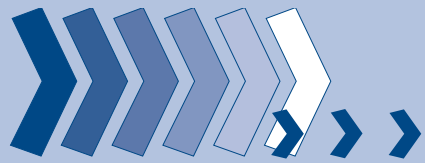
$$t - t_x < \sum_{T_j \in \mathcal{T}_c} \frac{t - t_x}{P_j} \cdot e_j$$

$$1 < \sum_{T_j \in \mathcal{T}_c} \frac{e_j}{P_j}$$

$$1 < U_{\mathcal{T}_c} < U \Rightarrow 1 < U$$

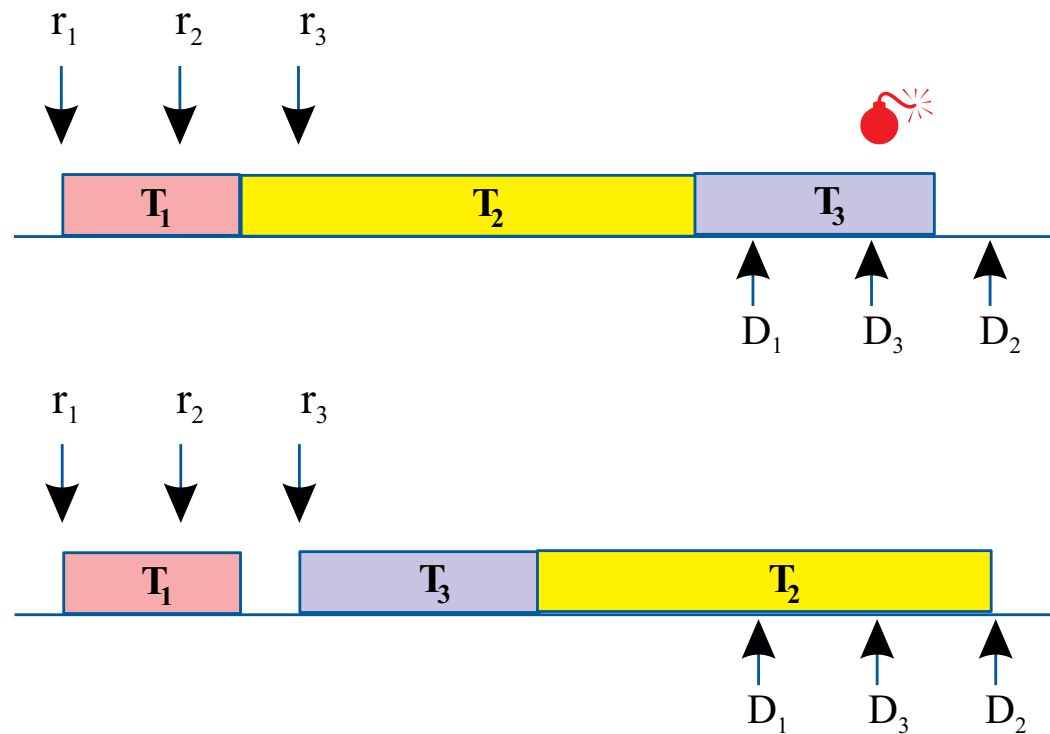
Dies ist ein Widerspruch, Theorem 0.2 ist somit bewiesen.

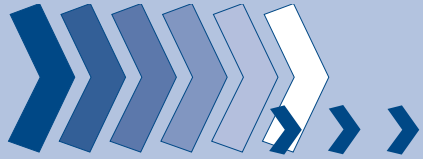




Notwendigkeit der Standardbedingungen (I)

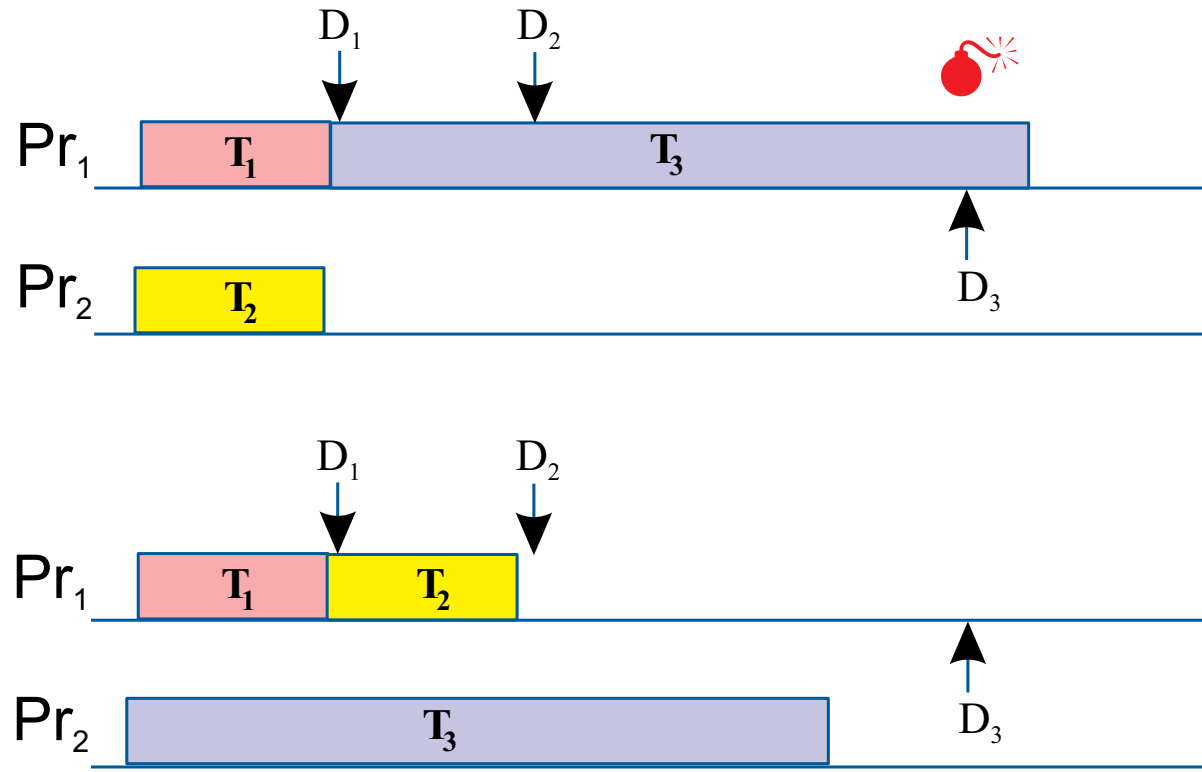
- Optimalität von EDF gilt nur für die Standardbedingungen
- Beispiel: Nichtunterbrechbare Tasks

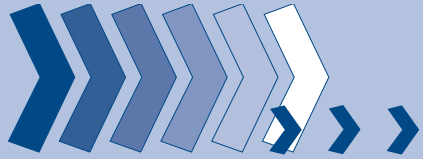




Notwendigkeit der Standardbedingungen (II)

EDF ist ebenfalls nicht optimal bei mehreren Prozessoren:





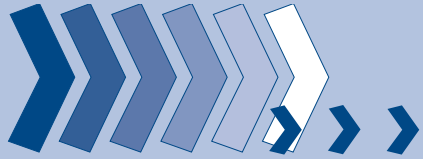
EDF bleibt optimal, wenn die Bedingung $D_i = P_i$ weggelassen wird.

Theorem 0.3. *Gilt für eine Taskmenge \mathcal{T} die Ungleichung*

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

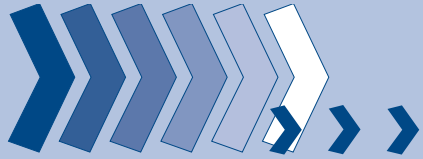
ist \mathcal{T} mit dem EDF-Algorithmus ausführbar.

- 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.

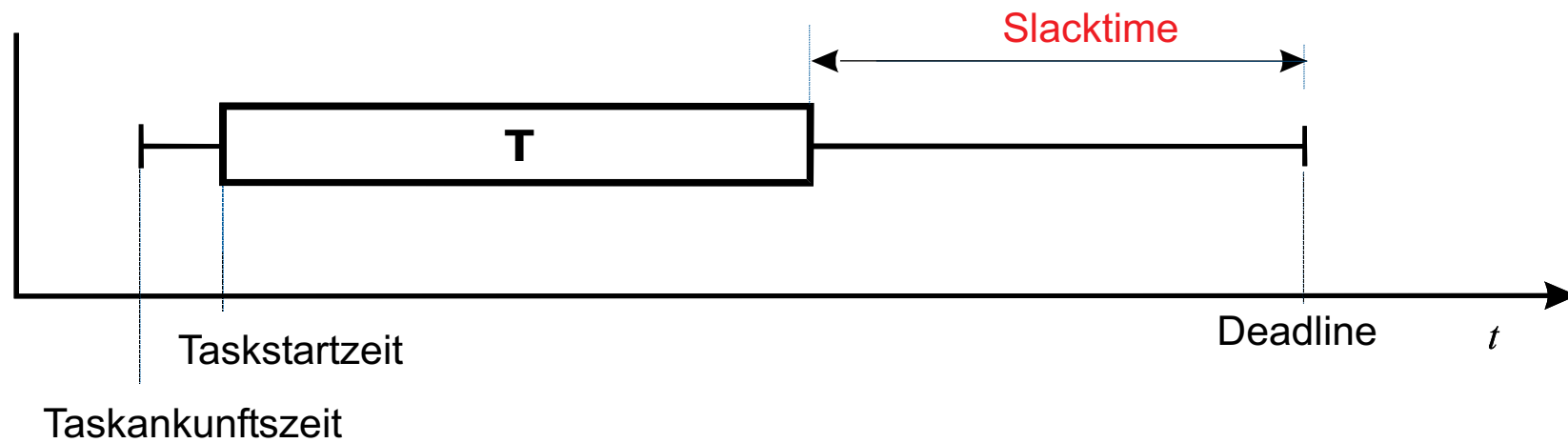


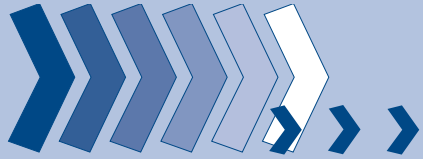
Least Slacktime First (LSF)

- LSF arbeitet mit variablen Prioritäten
- Jede Task erhält eine Priorität entsprechend ihrer Deadline der Differenz zwischen Deadline und (restlicher) Ausführungszeit (*Slacktime*).
Kürzere Slacktime = höhere Priorität.
- Eine lafbereite Tasks mit höherer Priorität unterbricht stets eine Task niederer Priorität.



- Slacktime, auch *Laxity*

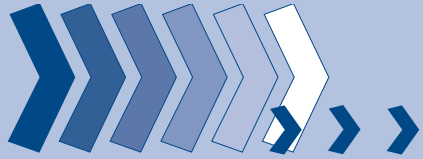




Theorem 0.4. *Unter den Standardannahmen kann LSF für die Taskmenge \mathcal{T} genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für \mathcal{T} existiert.*

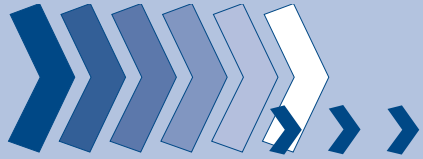
Anders gesagt: Wenn irgendein Schedulingverfahren \mathcal{S} unter den Standardbedingungen für \mathcal{T} einen ausführbaren Schedule erzeugen kann, dann kann LSF das auch.

Beweis: Ähnlich dem für EDF.

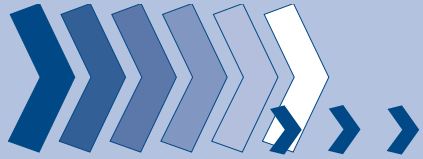


Rate Monotonic Scheduling (RMS)

- RMS arbeitet mit festen Prioritäten
- Task erhält eine Priorität reziprok zu ihrer Periode (Deadline)
- Eine laufbereite Task mit höherer Priorität unterbricht stets eine Task niedrigerer Priorität

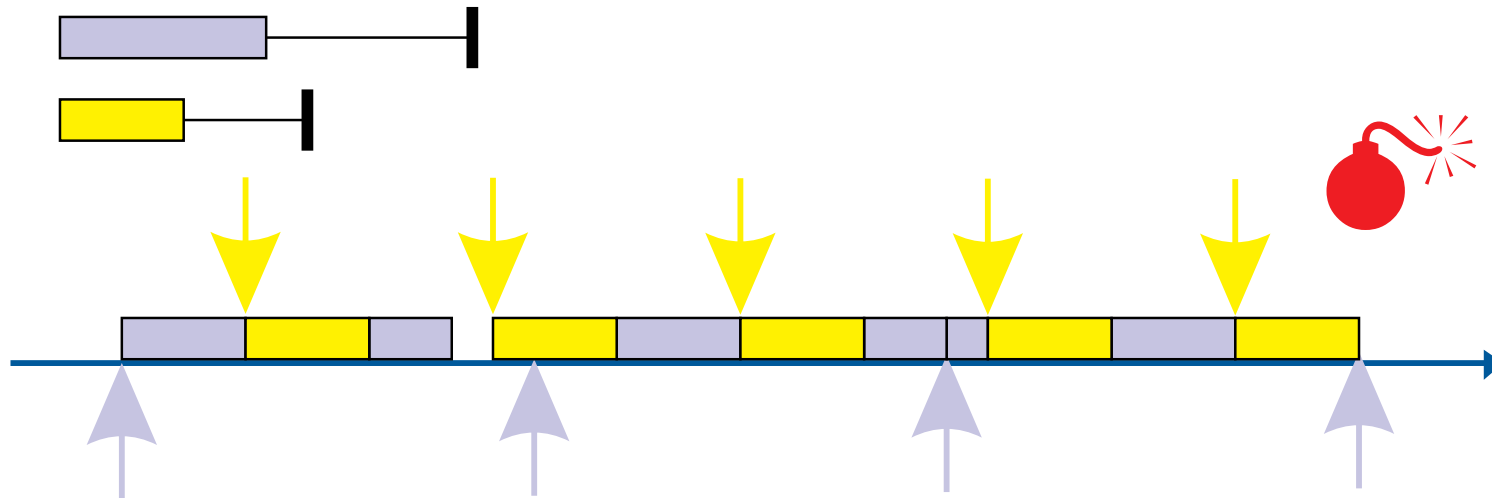


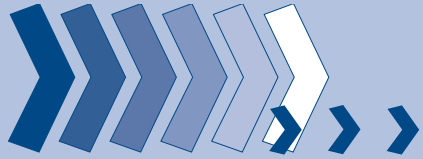
- Jane W. S. Lui: Real-Time Systems, Abschnitte 6.4 und 6.7
- C.M. Krishna und K.G. Shin: Real-Time Systems, Abschnitt 3.2.1
- C.L. Lui und J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, siehe Semesterapparat EMES 2001



Beispiel für RMS-Schedule

- Task 1: $P_1 = 6$, $e_1 = 3$
- Task 2: $P_2 = 10$, $e_2 = 5$
- Task 1 hat höhere Priorität
- ausführbar mit EDF





Kritischer Zeitpunkt bei RMS

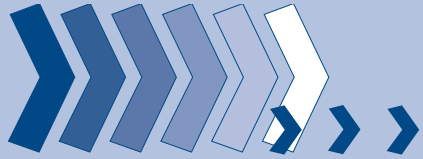
Theorem 0.5. *Der kritische Augenblick ist für jede Task in RMS, wenn sie zeitgleich mit allen höherpriorisierten Tasks ankommt.*

Beweis.

- Sei $\mathcal{T}: T_1, T_2$ mit $P_1 < P_2$.
- Sei $r_2 = 0$

Fall 1: T_1 wird zu $r_2 = 0$ ausgeführt, d.h., $-e_1 < r_1 \leq 0$.

- Dann ist $t_{k,2} = e_2 + (e_1 - |r_1|) + \left\lceil \frac{e_2}{P_1 - e_1} \right\rceil e_1$
- Alles außer r_1 ist konstant: $t_{k,2} = C - |r_1|$
- Dies ist am größten, wenn $|r_1|$ am kleinsten ist, $\Rightarrow r_1 = 0$.



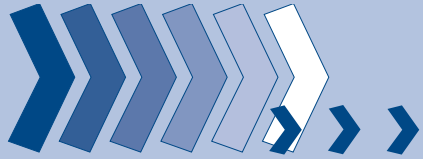
Beweis von Theorem 0.5 (Forts.)

Fall 2: T_1 wird nicht zu r_2 ausgeführt, d.h. $0 \leq r_1 < P_2 - e_2$

- Dann ist $t_{k,2} = e_2 + \left\lceil \frac{e_2 - r_1}{P_1 - e_1} \right\rceil e_1$
- Alles außer r_1 ist konstant,
- $t_{k,2}$ ist am größten, wenn r_1 am kleinsten ist
- $\Rightarrow r_1 = 0$.

Ähnliche Argumentation bei mehr als 2 Tasks.



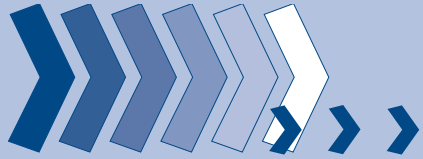


Optimalität von RMS

Theorem 0.6. *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} .

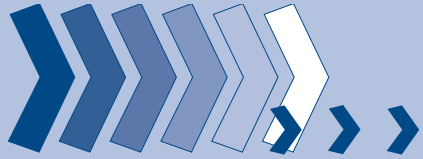
Aber: \mathcal{M} kann kein Fixed-Priority-Verfahren sein.



Beweis von Theorem 0.6

- \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)
- Austausch von p_i und p_j läßt den Schedule ausführbar:
 - Nur Stellen interessant, an denen T_i von T_j an Ausführung gehindert wird.
 - Da Deadline von T_j offensichtlich nach Deadline von T_i , können Taskstückchen vertauscht werden
- Wiederholen, bis \mathcal{S} zum RMS-Schedule geworden ist.





Schedulingkriterium für RMS

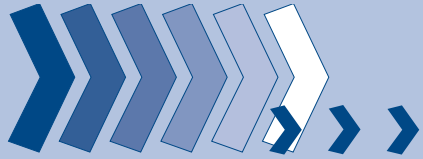
- Untere Schranke der obere Grenze

Theorem 0.7. *Ein Menge \mathcal{T} von n Tasks ist mit RMS unter den Standardbedingungen immer dann ausführbar, wenn*

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

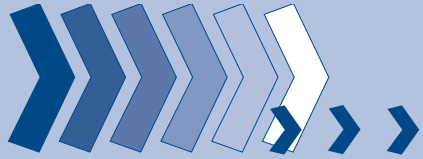
- Hinreichend, aber nicht notwendig

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



Grenzwert für Lastschränke

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

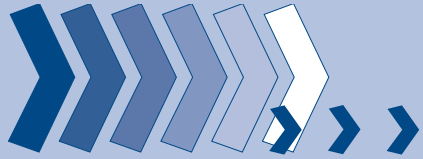


Beweis von Theorem 0.7 (Spezialfall)

Der Beweis von Theorem 0.7 für den Fall, daß keine Periode mehr als doppelt so lang wie die kürzeste Periode ist ($P_{max} \leq 2 \cdot P_{min}$).

Beweis besteht aus folgenden Schritten:

- **A:** Konstruktion eines bestimmten Taskset T
- **B:** Beweis, daß T unter RMS ausführbar ist.
- **C:** Beweis, daß T unter RMS den Prozessor voll ausnutzt.
- **D:** Beweis, daß jede andere Taskmenge, die den Prozessor voll ausnutzt, eine größere Last erzeugt.
- **E:** Beweis, daß T mindestens eine Last von $n(2^{\frac{1}{n}} - 1)$ erzeugt.
- **F:** Verallgemeinerung für beliebige Periodenverhältnisse



Konstruieren Taskset $T = \{T_1, T_2 \dots T_n\}$

- Alle Tasks sind nach ihrer Periode geordnet:

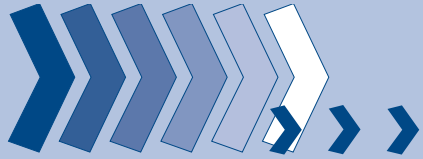
$$P_1 < P_2 < \dots < P_n$$

- Die Ausführungszeiten aller Tasks (außer der letzten) ist die Differenz zwischen der Periodenlänge der nächsten Task und der eigenen Periodenlänge.

$$\forall i, i = 1, \dots, n - 1 : e_i = P_{i+1} - P_i$$

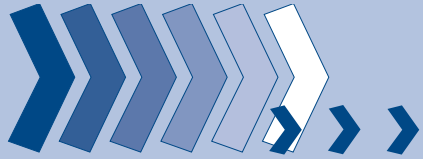
- Die Ausführungszeit der letzten Task ist die Differenz aus der eigenen Periode und dem doppelten der Summe aller anderen Ausführungszeiten.

$$e_n = P_n - 2 \sum_{i=1}^{n-1} e_i$$



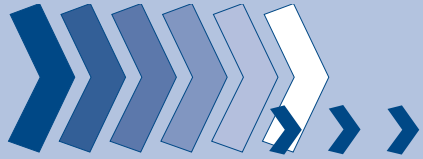
Analyse der Zeitforderungen

- Theorem 0.7 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
(siehe Paper im Semesterapparat)
- 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



Schedulingbedingung nach TDA

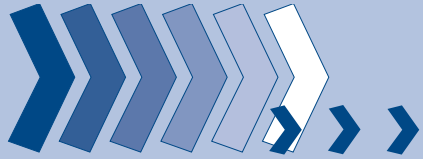
- Die Zeitanforderungsfunktion (*time demand function*) einer Task T_i wird mit $w_i(t)$ bezeichnet.
- $w_i(t)$ ist wie folgt definiert:

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

- Tasks sind nach Prioritäten geordnet, T_1 hat die höchste Priorität.

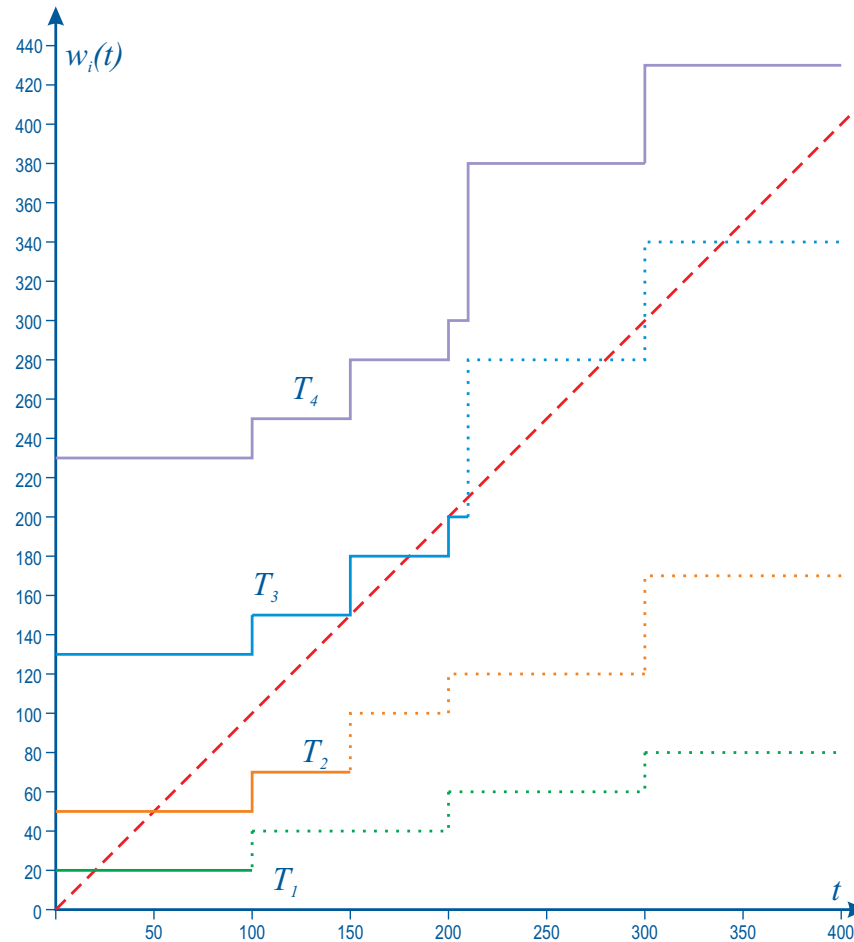
Theorem 0.8. *Eine Menge \mathcal{T} von unterbrechbaren, periodischen und unabhängigen Tasks ist mit einem Fixed-Priority-Verfahren \mathcal{S} ausführbar, wenn*

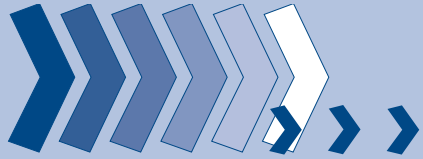
$$\forall i : (\exists t, 0 < t \leq P_i : w_i(t) \leq t)$$



Beispiel für TDA

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

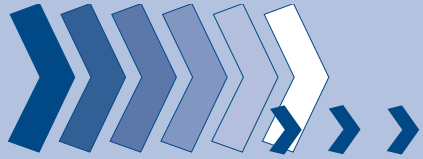




Variationen von RMS

Betrachten Änderungen der Grundbedingungen:

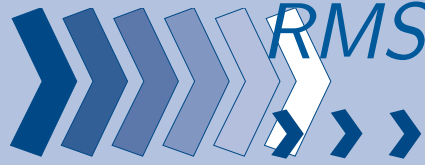
- RMS für einfach periodische Taskmengen
- Deadline Monotonic Scheduling (DMS)



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.

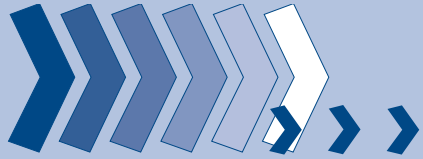
Theorem 0.9. *Unter den Standardannahmen kann RMS für jede einfach periodische Taskmenge \mathcal{T} , genau dann einen ausführbaren Schedule erzeugen, wenn irgendein ausführbarer Schedule für \mathcal{T} existiert.*



RMS-Schedulingbedingung für einfach periodische Tasksets

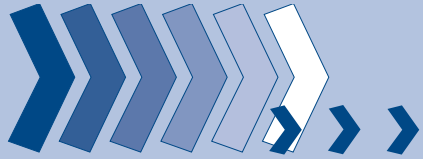
Theorem 0.10. [Schedulingbedingung] Eine Menge \mathcal{T} von einfach periodischen, unabhängigen und unterbrechbaren Tasks T_i , in der für alle T_i gilt: $D_i \leq P_i$, ist genau dann mit RMS ausführbar, wenn

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



Deadline Monotonic Scheduling (DMS)

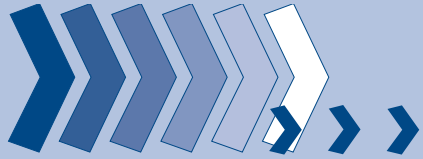
- DMS ist eine Verallgemeinerung von RMS
- Es gilt nicht mehr $D_i = P_i$
- Statische Zuweisung von Prioritäten entsprechend ihrer relativen Deadline
- Für $D_i = P_i$ gilt $DMS = RMS$



Bei beliebigen Deadlines gilt: DMS ist RMS überlegen

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

Theorem 0.11. *DMS ist unter den Fixed-Scheduling-Verfahren mit beliebigen Deadlines optimal.*



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