

Grundlagen der Echtzeitplanung

1. Grundlegende Begriffe und Konzepte

2. Planungsverfahren (Scheduling)

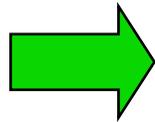
2.1 Planen aperiodischer Tasks

- Planen durch Suchen
- Planen nach Fristen
- Planen abhängiger Tasks

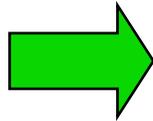
2.2 Planen periodischer Tasks

- Planen nach monotonen Raten
- EDF

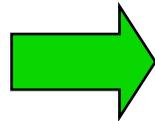
Klassifizierung der Schedulingalgorithmen



**Annahmen über die Systemumgebung:
Einprozessor-System,
Mehrprozessor-System,
verteiltes System,**



**Annahmen über die Task-Menge:
Unterbrechbarkeit,
Unabhängigkeit,
Aktivierung,**



**Planungs(Optimalitäts)kriterium
bezogen auf eine Kostenfunktion**

Notation:

n | async | L_{\max}

3 | non-preemt | N_{late}

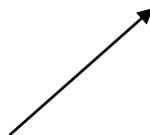
1 | sync | L_{\max}

1 | async, preemt | L_{\max}

Annahmen: System

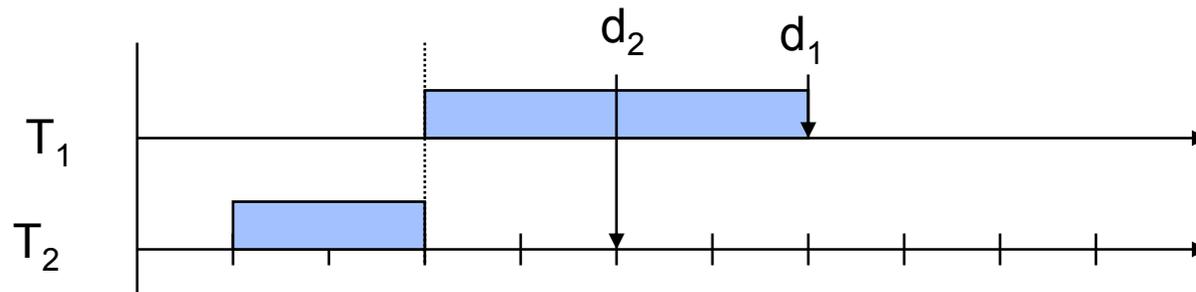
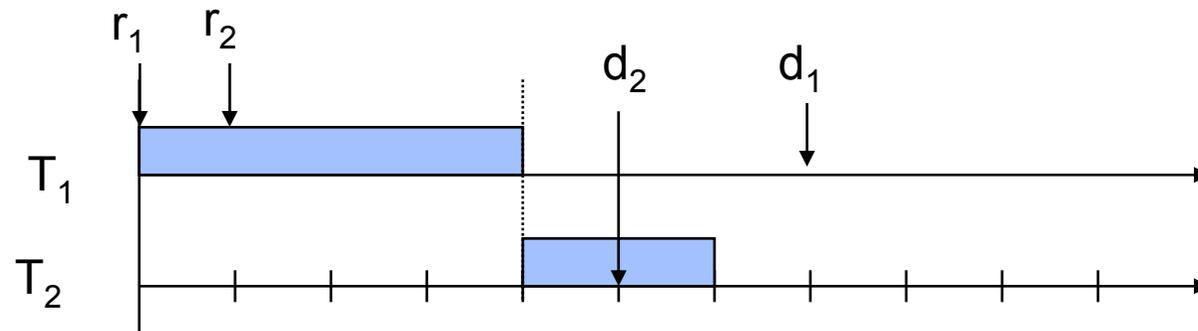
Taskmenge

Kriterium



Planung nicht unterbrechbarer Tasks

Problem:

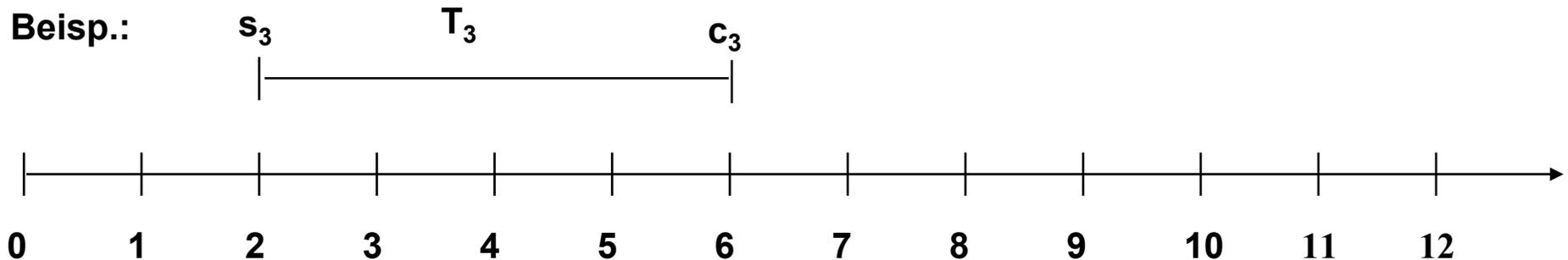


Planen durch Durchsuchen des Lösungsraums

1 | **async** | **feasible**

Gegeben : Menge T nicht unterbrechbarer Tasks mit $|T| = n$, r_i , Δe_i , c_i .

Ein statisches Planungsverfahren soll untersuchen, ob ein Plan existiert. Das Verfahren soll einen Plan in Form einer Folge von Tupeln (i, s_i) mit i : Tasknr. und s_i : Startzeit von T_i .



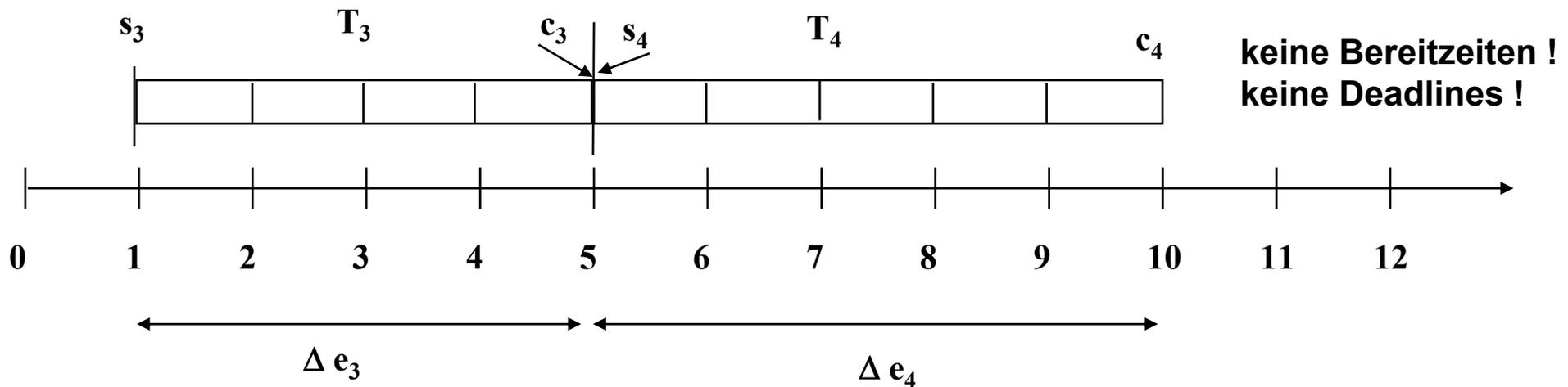
Der Eintrag für T_3 im Plan ist $(i, s_i) = (3, 2)$

$$\Delta e_3 = 4$$
$$c_3 = 6$$

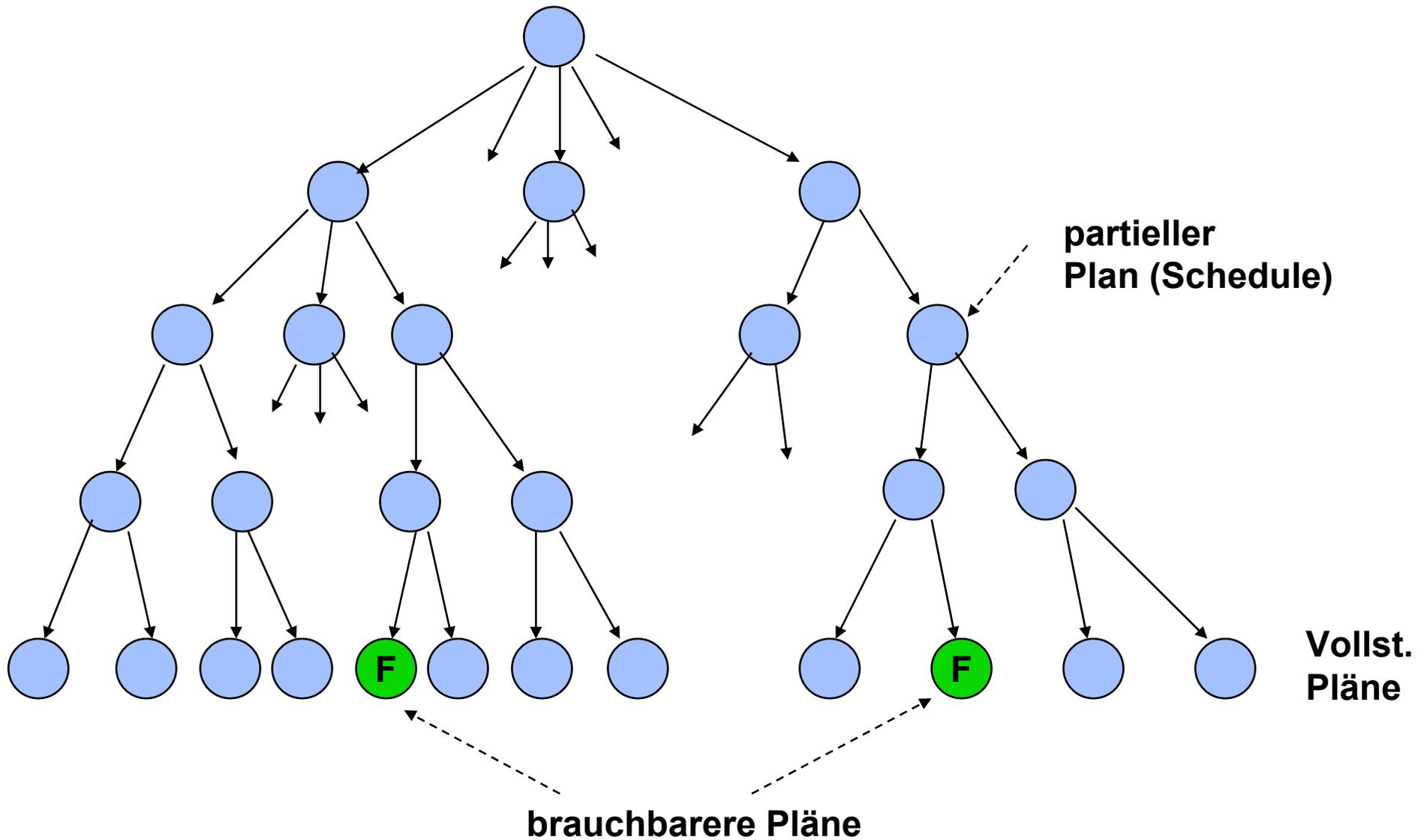
Generierung des Lösungsraums

Es werden alle möglichen Kombinationen von Tasks als Lösungen generiert ohne Beachtung von Bereitzeiten und Fristen.

Für n Tasks gibt es $n!$ Lösungsmöglichkeiten. Die Lösungen einschließlich aller Lösungsschritte können in einem Baum der Tiefe n dargestellt werden. Jeder Knoten des Baums markiert einen bestimmten Stand des Planungsverfahrens.



Suchbaum zur Erstellung eines Plans



Konstruktion eines Algorithmus:

Voraussetzungen: Es werden keine r_i oder d_i berücksichtigt. Mit *start PL* wird eine neue Task T_i in den Plan PL_k aufgenommen, indem die Startzeit s_i von T_i mit der Abschlußzeit c_{i-1} der zuletzt in Plan PL_k aufgenommenen Task T_{i-1} gleichgesetzt wird.

PL_{k+1} ist das Funktionsergebnis von *start PL* (PL_k, i).

T ist die Menge aller Tasks, die eingeplant werden sollen.

X_k ist die Teilmenge von T , für die bereits alle Tasks in PL_k eingeplant wurden mit $|X_k| = k$.

Planungsverfahren 1:

Erstelle Plan, ausgehend von Knoten x .
Die Menge X_k ist dann schon berücksichtigt

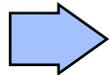
```
schedule ( $PL_k, X_k$ ) :  
  FORALL  $i$  IN  $T \setminus X_k$   
    schedule startPL ( $PL_k, i$ ),  $X_k \cup \{i\}$ )
```

Task i wird in die Menge der verplanten Tasks aufgenommen.

Aufwand des Planungsverfahrens : $O(n!)$
(Dies entspricht einem exponentiellen Aufwand)

Planung unter Einbeziehung von Bereitzeiten und Fristen

Nicht alle kombinatorisch erzeugten Pläne sind gültig, wenn Bereitzeiten und Fristen berücksichtigt werden!



Bewertungsfunktion wird benötigt !

Heuristik: Einplanen einer Task so früh wie möglich, d.h. $s - r = \min$

Vorgehensweise:

Trifft man im Baum auf einen ungültigen Plan, d.h. läßt sich eine Task mit entsprechender Deadline nicht mehr einplanen, wird die Suche in dem entsprechenden Ast nicht weiter fortgesetzt.

Planung unter Einbeziehung von Bereitzeiten und Fristen

Bratley`s Algorithmus (P. Bratley; M. Florian, P. Robillard: “Scheduling with earliest start and due date constraints”, Naval Research Quarterly, 18 (4), 1971

Klasse: $1|non\text{-preemt}|L_{max}$

***earliest PL* (PL_k, i)** : Funktion, die bezogen auf den Plan PL_k die Task T_i in die früheste Lücke einplant, die

1. aufgrund der Bereitzeit r_i von T_i
2. aufgrund des bereits bisher erstellten Plans PL_k möglich ist.

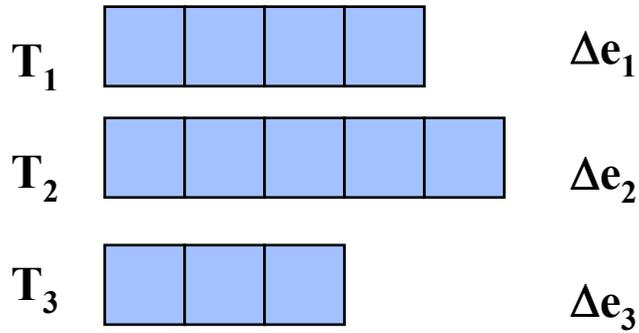
***feasible PL* (PL_k, i)** : Funktion, die testet, ob Task T_i in den bisherigen Plan PL_k integriert werden kann, d.h. ob überhaupt noch eine einplanbare Lücke für Task T_i existiert.

Planungsverfahren 2:

***schedule* (PL_k, X_k)** :

FORALL (i IN $T \setminus X_k$) AND ***feasible PL* (PL_k, i)**

***schedule* (*earliest PL* (PL_k, i), $X_k \cup \{i\}$)**



Taskliste

$T_1, r_1, d_1, \Delta e_1$

$T_2, r_2, d_2, \Delta e_2$

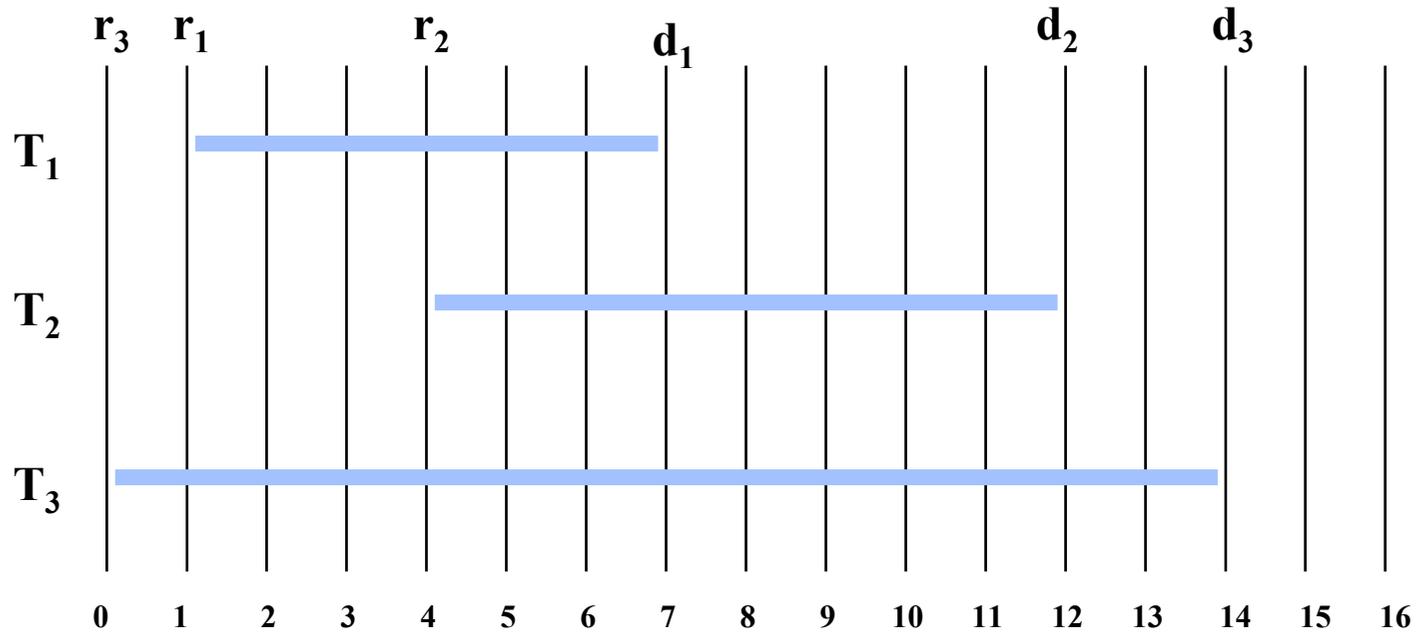
$T_3, r_3, d_3, \Delta e_3$

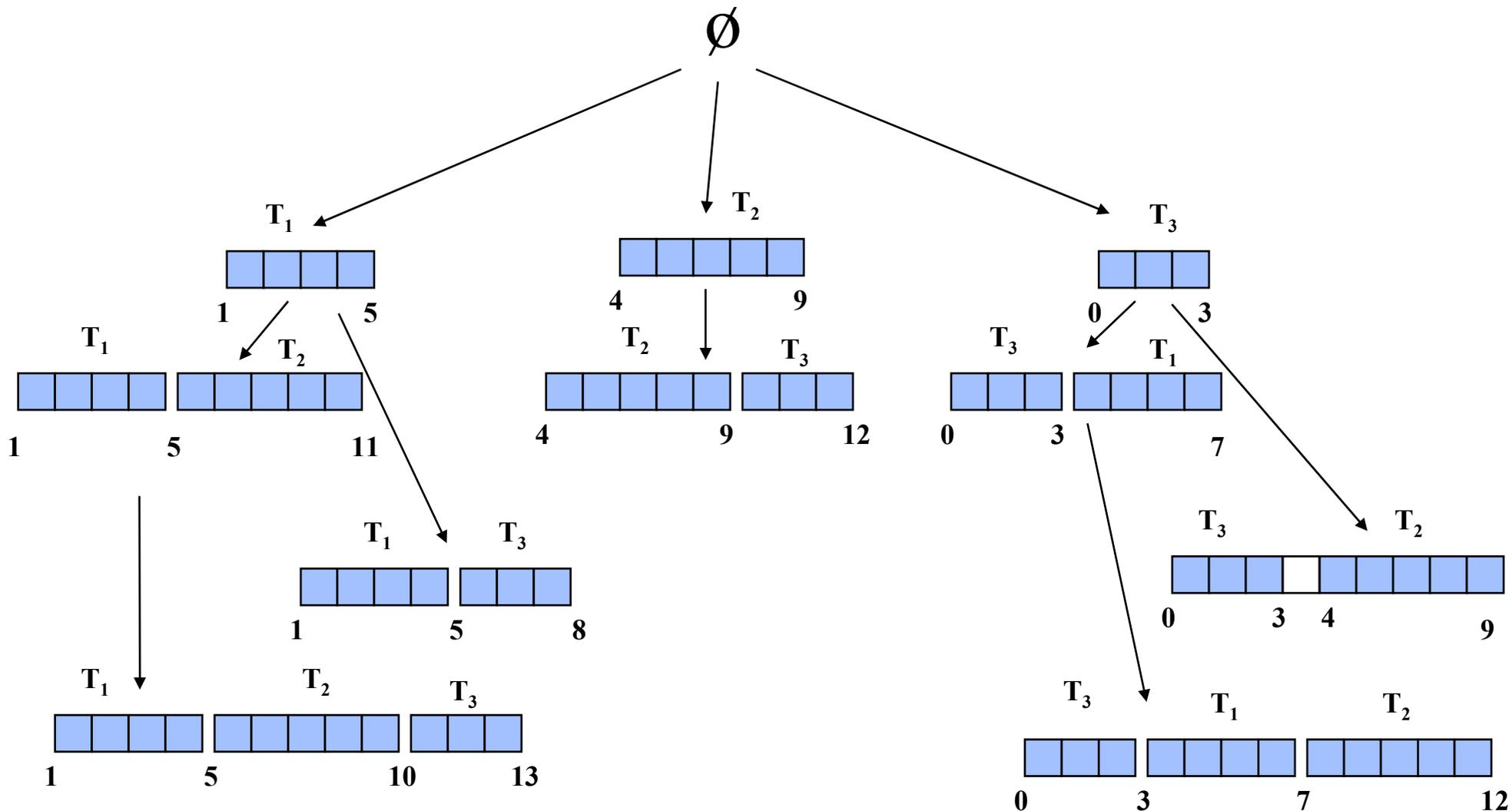


1, 1, 7, 4

2, 4, 12, 5

3, 0, 14, 3





$$\langle (T_1, s_1), (T_2, s_2), (T_3, s_3) \rangle$$

$$= \langle (1, 1), (2, 5), (3, 10) \rangle$$

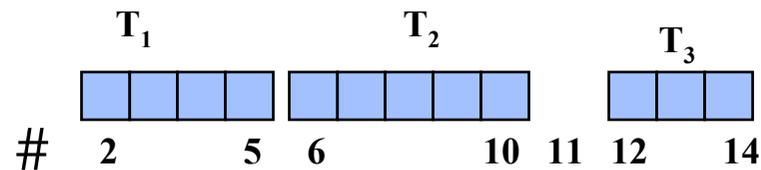
$$\langle (T_3, s_3), (T_1, s_1), (T_2, s_2) \rangle$$

$$= \langle (3, 0), (1, 3), (2, 7) \rangle$$

Gütekriterium

Ist das vorgestellte Planungsverfahren optimal ?

Es gibt Pläne, die gültig sind, aber nicht mit Planungsverfahren 2 erzeugt werden, z.B. solche, die Lücken aufweisen.



Frage: Wenn schon nicht alle möglichen Pläne gefunden werden - gibt es Fälle, in denen ein gültiger Plan existiert, aber nicht erzeugt wird ?

Satz: Gegeben sei eine Menge nicht unterbrechbarer Tasks $T = \{1, \dots, n\}$.
Wenn es gültige Pläne gibt, werden diese durch Planungsverfahren 2 erzeugt.

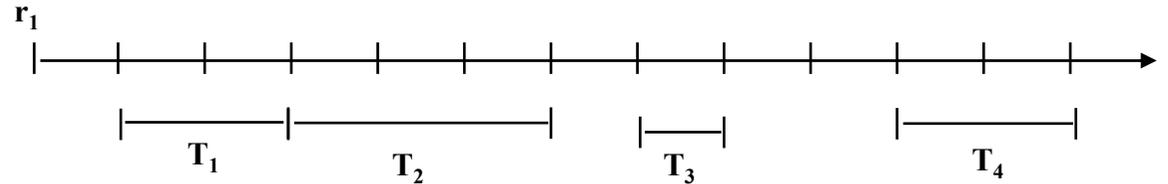
Satz: Gegeben sei eine Menge nicht unterbrechbarer Tasks $T = \{1, \dots, n\}$.

Wenn es gültige Pläne gibt, werden gültige Pläne auch durch Planungsverfahren 2 erzeugt.

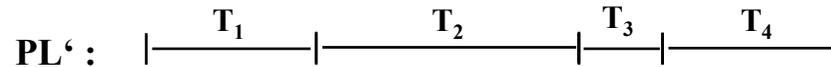
Beweisidee:

Gegeben: PL

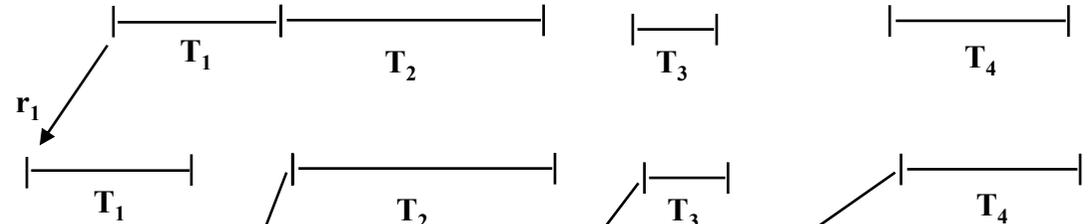
Beweis: $PL \equiv PL'$



1. Schritt: Konstruktion eines Plans PL' nach Bratley's Algorithmus



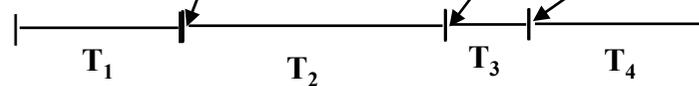
2. Schritt: PL'_1 entsteht aus PL , indem s_1 von P_1 auf r_1 vorgezogen wird.



PL'_1 ist gültig!

Annahme PL'_k ist gültig!

3. Schritt: PL'_{k+1} entsteht aus PL'_k , indem s_{i+1} auf $\max(r_{k+1}, c_k)$ vorgezogen wird.



$\max(r_{k+1}, c_k)$ sind die für einen gültigen Plan frühesten Zeiten, d.h. gerade die, die durch das Verfahren *earliest PL* (PL_k, i) gefunden werden.

Probleme:

- Die kombinatorische Erzeugung aller Pläne benötigt $n!$ Planungsschritte bei n Tasks.
- Die Konstruktion eines Plans mit diesem Verfahren ist nicht mit polynomiellen Aufwand zu realisieren.
- Tatsächlich ist das Suchverfahren NP-vollständig

Schlechte Nachricht:

Falls die Bereitzeiten nicht alle gleich sind und nicht-unterbrechbare Tasks vorliegen, gibt es kein anderes Verfahren das optimal ist.

Grundlagen der Echtzeitplanung

1. Grundlegende Begriffe und Konzepte

2. Planungsverfahren (Scheduling)

2.1 Planen aperiodischer Tasks

- Planen durch Suchen
- **Planen nach Fristen**
- Planen nach Spielräumen

2.2 Planen periodischer Tasks

- Planen nach monotonen Raten
- EDF

EDF : Earliest Deadline First

Der Prozessor wird derjenigen Task zugeteilt, welche die kürzeste Frist besitzt, d.h. deren Deadline auf der Zeitlinie den kleinsten Wert hat. Wenn es keine rechenbereiten Tasks gibt, bleibt der Prozessor untätig (idle).

- **Eine der verbreitetsten zeitbasierten Planungsstrategien**
- **Anwendbar auf unterbrechbare und nicht unterbrechbare Tasks**
- **Anwendbar in statischen und dynamischen Planungsverfahren**

EDF bei nicht unterbrechbaren Tasks

Voraussetzung: Tasks haben gleiche Bereitzeiten. Klasse: 1| sync, non-preemt| L_{\max}

Die Tasks $T = (1, \dots, n)$ seien nach ihren Fristen geordnet mit: $1 \leq i \leq j \leq n \Rightarrow d_i < d_j$.
Die Funktion *deadline* (PL_k, i) erzeugt aus dem Plan PL_k durch Einfügen von Task i den Plan PL_{k+1} .

Planungsverfahren EDF_n :

```
schedule (PL, T) :  
    PL = < > ;  
    i = 1;  
    WHILE ( i ≤ n )  
        BEGIN  
            PL = deadline (PLk, i);  
            i = i + 1  
        END  
    END
```

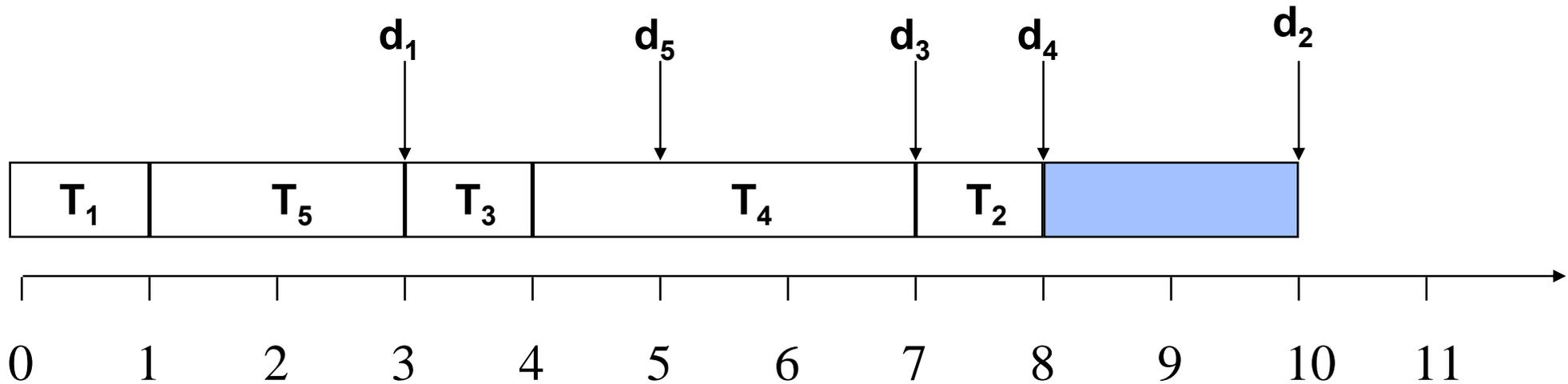
Satz (Jackson 1955) EDD: Earliest Due Date:

Gegeben sei eine Menge unabhängiger Tasks. Alle Tasks werden synchron aktiviert. Jeder Algorithmus, der die Tasks in der Reihenfolge nicht absteigender Deadlines ausführt ist optimal im Hinblick auf die Minimisierung der maximalen Verspätung.

Klasse: 1 | non-preemt, sync | L_{\max}

Beispiel 1:

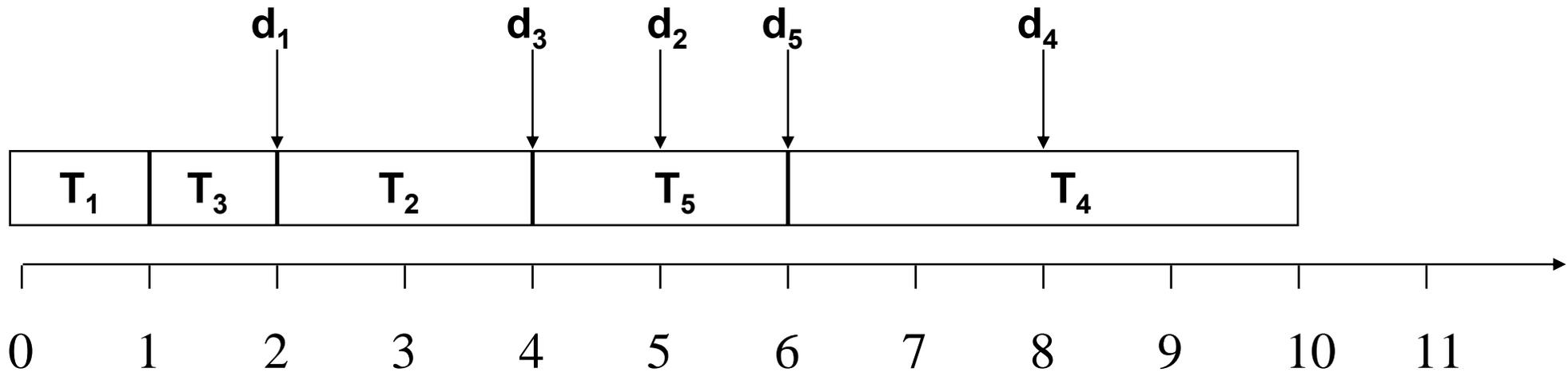
	T_1	T_2	T_3	T_4	T_5
Δe_i	1	1	1	3	2
d_i	3	10	7	8	5



$$L_{\max} = L_4 = -1$$

Beispiel 2:

	T_1	T_2	T_3	T_4	T_5
Δe_i	1	2	1	4	2
d_i	2	5	4	8	6

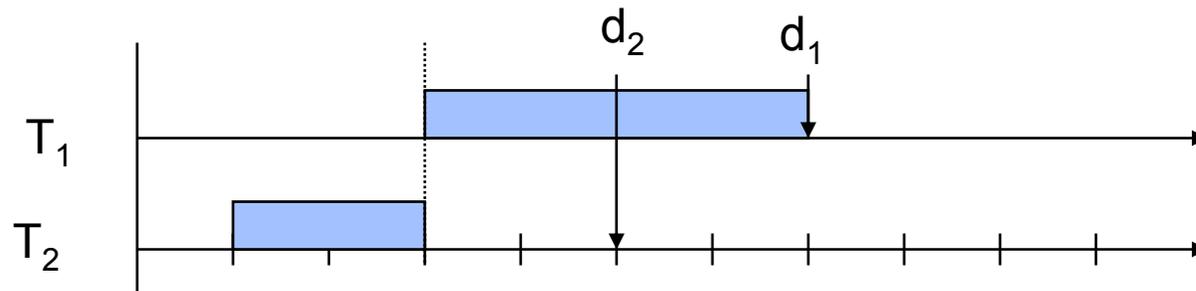
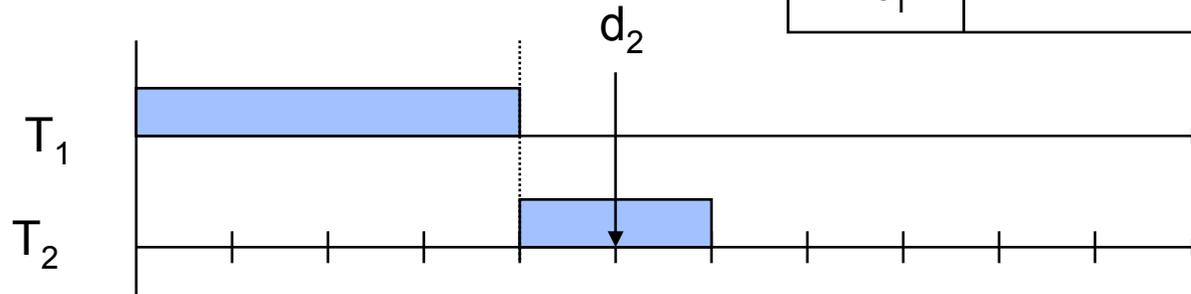


$$L_{\max} = L_4 = 2$$

EDF ist im allgemeinen Fall nicht optimal bei einem nicht-unterbrechbaren Task-Modell.

Beispiel:

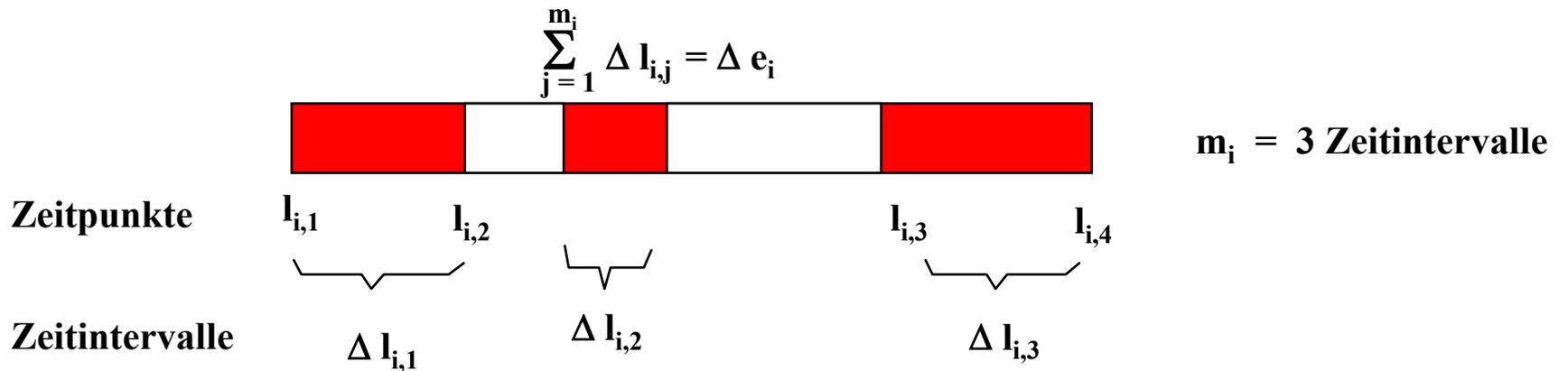
	T_1	T_2
r_i	0	1
Δe_i	4	2
d_i	7	5



EDF für unterbrechbaren Tasks

Motivation: Planung wenn nicht alle Bereitzeiten gleich sind und Tasks dynamisch bereit werden.

- Zerlegung der Ausführungszeit einer Task



Randbedingungen:

- 1.) $\sum_{j=1}^{m_i} \Delta l_{i,j} = \Delta e_i$
- 2.) $r_i \leq s_i = l_{i,1} \leq l_{i,m_i} + \Delta l_{i,m_i} = c_i \leq d_i$

Plan wird dargestellt als Folge von Tupeln:
 $(i, l_{i,j}, \Delta l_{i,j})$

Notwendiges und hinreichendes Kriterium für unterbrechbare Tasks

- ➔ Neue Tasks können hinzukommen, deshalb muss der Test bei jeder neuen Task dynamisch ausgeführt werden.
- ➔ Tasks sind (einschließlich einer neu hinzukommenden Task) nach aufsteigenden **Deadlines** geordnet
- ➔ Da Tasks unterbrochen werden können wenn eine neue Task zum Zeitpunkt t bereit wird, kann es sein, dass Tasks noch nicht vollständig ausgeführt sind, d.h. noch ein Rest ΔR_i ausgeführt werden muss.

Die maximale (worst-case) Zeit bis zum Abschluss c zum Zeitpunkt t kann dann Berechnet werden durch:

$$c_i = \sum_{k=1\dots i} \Delta R_k$$

Deshalb kann ein Plan unter der folgenden Bedingung garantiert werden:

$$\forall i = 1, \dots, n \text{ gilt: } \sum_{k=1\dots i} \Delta R_k \leq d_i$$

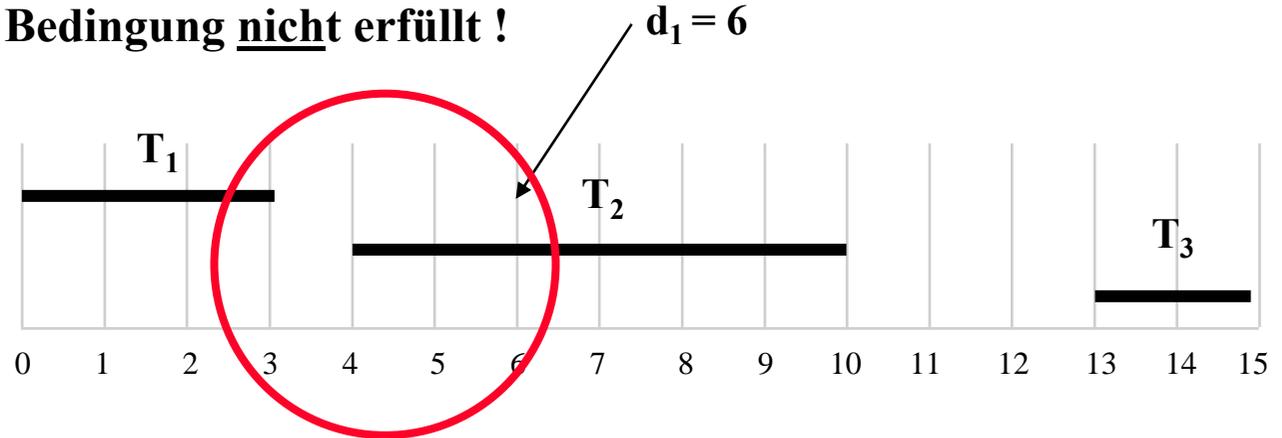
Beispiel: hinreichende aber nicht exakte Bedingung für Einplanbarkeit

Bedingung: $\forall i,j, d_i < d_j: d_i \leq d_j - \Delta e_j$

In beiden Fällen wird die Bedingung nicht erfüllt !

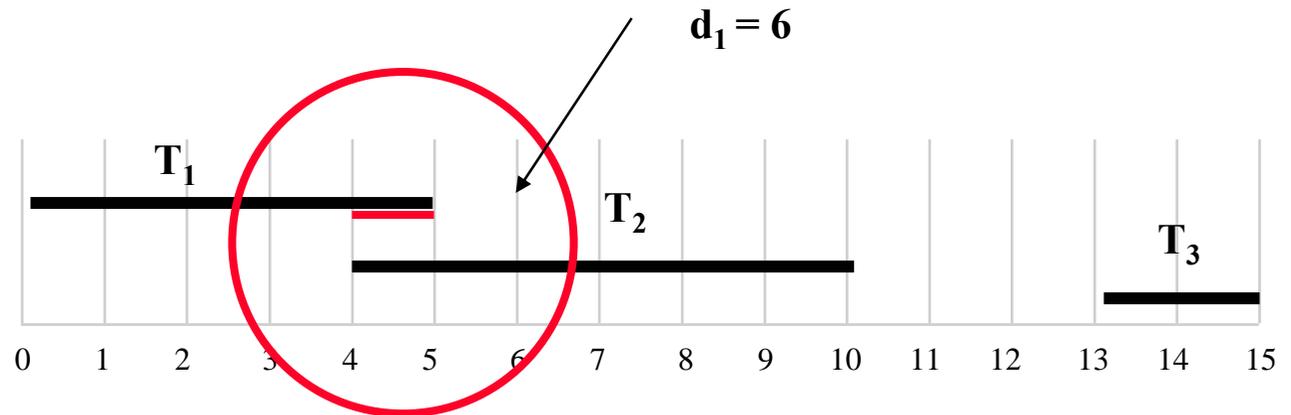
- $T_1: \Delta e_1 = 3, r_1 = 0, d_1 = 6$
- $T_2: \Delta e_2 = 6, r_2 = 4, d_2 = 10$
- $T_3: \Delta e_3 = 2, r_3 = 13, d_3 = 15$

➡ **planbar !!**



- $T_1: \Delta e_1 = 5, r_1 = 0, d_1 = 6$
- $T_2: \Delta e_2 = 6, r_2 = 4, d_2 = 10$
- $T_3: \Delta e_3 = 2, r_3 = 13, d_3 = 15$

➡ **nicht planbar !!**



$d_i \leq d_j - \Delta e_j$ lässt unberücksichtigt, wann T_1 tatsächlich abgeschlossen ist !

Beispiel 1:

Fall 1:	$\Delta e_1 = 3$	$d_1 = 4$	}	Bed. 1 und 2 erfüllt Bed. 3 nicht erfüllt
	$\Delta e_2 = 4$	$d_2 = 7$		
	$\Delta e_3 = 3$	$d_3 = 9$		
	$\Delta e_4 = 5$	$d_4 = 15$		

Fall 2:	$\Delta e_1 = 2$	$d_1 = 4$	}	Bed. 1 und 2 erfüllt Bed. 3 nicht erfüllt
	$\Delta e_2 = 4$	$d_2 = 7$		
	$\Delta e_3 = 3$	$d_3 = 9$		
	$\Delta e_4 = 5$	$d_4 = 15$		

Da die Tasks synchron aktiviert werden, kann $t=0$ angenommen werden.

Fall 1:

T_1	$c_1 = \Delta R_1$	$(\Delta R_1=3) = 3$	\leq	d_1
T_2	$c_2 = c_1 + \Delta R_2$	$(\Delta R_2=4) = 7$	\leq	d_2
T_3	$c_3 = c_2 + \Delta R_3$	$(\Delta R_3=3) = 10$	\geq	$d_3 !!$

Test erkennt: Nicht planbar

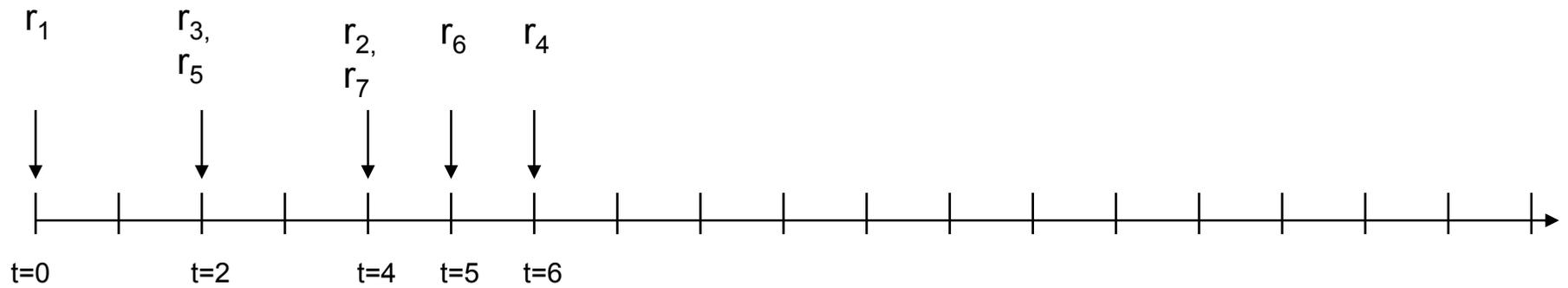
Fall 2:

T_1	$c_1 = \Delta R_1$	$(\Delta R_1=2) = 2$	\leq	d_1
T_2	$c_2 = c_1 + \Delta R_2$	$(\Delta R_2=4) = 6$	\leq	d_2
T_3	$c_3 = c_2 + \Delta R_3$	$(\Delta R_3=3) = 9$	\leq	d_3
T_4	$c_4 = c_3 + \Delta R_4$	$(\Delta R_4=5) = 14$	\leq	d_4

Test erkennt: Planbar

Beispiel 2:

	T1	T2	T3	T4	T5	T6	T7
r_i	0	4	2	6	2	5	4
Δe_i	2	4	4	2	3	3	2
d_i	4	8	12	10	13	18	20



	Abschlußzeit		Rest		Deadline	
t=0						
T_1	$c_1 = \Delta R_1$	$= 2$	$(\Delta R_1 = \Delta e_1)$	$\leq d_1 = 4$	$/T_1$ kann vollständig ausgeführt werden	
t=2						
T_3	$C_3 = t + \Delta R_3$	$= 6$	$(\Delta R_3 = \Delta e_3)$	$\leq d_3 = 12$	$/T_3$ bekommt bis t=4 2 Einh. Rechenzeit	
T_5	$C_5 = C_3 + \Delta R_5$	$= 9$	$(\Delta R_5 = \Delta e_5)$	$\leq d_5 = 13$	$/T_5$ bekommt bis t=4 keine Rechenzeit	
t=4						
T_2	$c_2 = t + \Delta R_2$	$= 8$	$(\Delta R_2 = \Delta e_2)$	$\leq d_2 = 8$	$/T_2$ bekommt bis t=5 1 Einh. Rechenzeit	
T_3	$C_3 = C_2 + \Delta R_3$	$= 10$	$(\Delta R_3 = \Delta e_3 - 2)$	$\leq d_3 = 12$	$/T_3$ bekommt bis t=5 keine Rechenzeit	
T_5	$C_5 = C_3 + \Delta R_5$	$= 13$	$(\Delta R_5 = \Delta e_5)$	$\leq d_5 = 13$	$/T_5$ bekommt bis t=5 keine Rechenzeit	
T_7	$C_7 = C_5 + \Delta R_7$	$= 15$	$(\Delta R_7 = \Delta e_7)$	$\leq d_7 = 20$	$/T_7$ bekommt bis t=5 keine Rechenzeit	
t=5						
T_2	$C_2 = t + \Delta R_2$	$= 8$	$(\Delta R_2 = \Delta e_2 - 1)$	$\leq d_2 = 8$	$/T_2$ bekommt bis t=6 1 Einh. Rechenzeit	
T_3	$C_3 = C_2 + \Delta R_3$	$= 10$	$(\Delta R_3 = \Delta e_3 - 2)$	$\leq d_3 = 12$	$/T_3$ bekommt bis t=6 keine Rechenzeit	
T_5	$C_5 = C_3 + \Delta R_5$	$= 13$	$(\Delta R_5 = \Delta e_5)$	$\leq d_5 = 13$	$/T_5$ bekommt bis t=5 keine Rechenzeit	
T_6	$C_6 = C_5 + \Delta R_6$	$= 16$	$(\Delta R_6 = \Delta e_6)$	$\leq d_6 = 18$	$/T_6$ bekommt bis t=5 keine Rechenzeit	
T_7	$C_7 = C_6 + \Delta R_7$	$= 18$	$(\Delta R_7 = \Delta e_7)$	$\leq d_7 = 20$	$/T_7$ bekommt bis t=5 keine Rechenzeit	
t=6						
T_2	$C_2 = t + \Delta R_2$	$= 8$	$(\Delta R_2 = \Delta e_2 - 2)$	$\leq d_2 = 8$	$/T_2$ kann bis t=8 vollständig ausgef. werden	
T_4	$C_4 = C_2 + \Delta R_4$	$= 10$	$(\Delta R_4 = \Delta e_4)$	$\leq d_4 = 10$	$/T_4$ bekommt keine Rechenzeit	
T_3	$C_3 = C_4 + \Delta R_3$	$= 12$	$(\Delta R_3 = \Delta e_3 - 2)$	$\leq d_3 = 12$	$/T_3$ bekommt keine Rechenzeit	
T_5	$C_5 = C_4 + \Delta R_5$	$= 15$	$(\Delta R_5 = \Delta e_5)$	$> d_5 = 13$	$/T_5$ benötigt 3 Einh. Rechenzeit, die aber bis zur Deadline nicht mehr vorhanden sind.	

nicht planbar !

EDF Planung für unterbrechbare Tasks

EDF kann zu einem Zeitpunkt t nur auf rechenbereite Tasks angewandt werden, d.h. wenn t zwischen Bereitzeit und Deadline liegt und die Task noch nicht vollständig ausgeführt ist.

Ready (t) ist die nach Fristen geordnete Liste.

$$\text{Ready}(t) = \{T_i \mid r_i \leq t < d_i \text{ und } \text{rest}(i, t) > 0\}$$

mit $\text{rest}(i, t) = \Delta e_i - \sum_{j \in \text{PL}} \Delta l_{i,j}$

d.h. $\text{rest}(i, t)$ umfaßt alle Ausführungszeiten von Task i , die in den, bis zum Zeitpunkt t aufgestellten Plan noch nicht aufgenommen wurden.

Die Funktion $edf(\text{Ready}(t))$ bestimmt zum Zeitpunkt t die Task aus Ready (t), welche die nächste Deadline besitzt. Da aber jederzeit neue Tasks hinzukommen können, die eingeplant werden müssen, ergibt sich für den nächsten Zeitpunkt nach t , an dem geplant werden muß:

$$\text{nextavail}(t) = \begin{cases} \min \{r_i \mid r_i > t\} & \text{falls } \{r_i \mid r_i > t\} \neq \emptyset \\ \max \{d_i\} & \text{sonst} \end{cases}$$

$\min \{r_i \mid r_i > t\}$ bezeichnet dabei die nächste neu hinzugekommene Bereitzeit r_i nach dem Zeitpunkt t . Es muß also neu geplant werden, wenn eine neue Task mit der Bereitzeit $r_i > t$ hinzukommt, oder die Deadline d_i der gerade ausgeführten Task erreicht wird.

Das Planungsverfahren endet, wenn:

1.) $\sum_{i \in P} \text{rest}(i, t) = 0$, d.h. alle Tasks zu Ende geführt sind

2.) $\neg \text{feasible PL}(i, t) \Leftrightarrow t + \text{rest}(i, t) > d_i$, d.h. wenn eine rechenbereite Task ihre Frist verletzen wird.

Die Boolesche Funktion: $\text{allin PL}(t)$ liefert den Wert "wahr", wenn 1.) oder 2.) gilt (d.h. das Planungsverfahren beendet ist) und den Wert "falsch" sonst (d.h. noch planbare Reste vorhanden sind).

Planungsverfahren für unterbrechbare Tasks nach EDF_u :

schedule (PL, T):

PL = < > ;

t = min { r_i | $r_i \in T$ };

WHILE \neg *allin* PL(t) DO

IF *Ready* (t) = \emptyset

THEN t = *nextavail* (t); ELSE

BEGIN

i = *edf* (*Ready* (t));

IF \neg *feasible* (i, t) THEN BREAK;

Δt = min (*rest* (i,t), *nextavail* (t) - t) ;

PL = PL (i, t, Δt);

t = t + Δt ;

END

Satz:

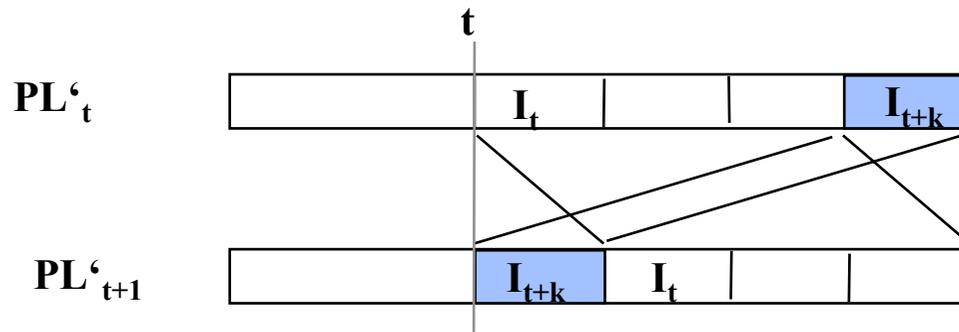
Gegeben seien die unterbrechbaren Tasks $T \{1, \dots, n\}$. Wenn es brauchbare Pläne gibt, dann werden sie durch den Planungsalgorithmus EDF_u gefunden (Horn 1974)

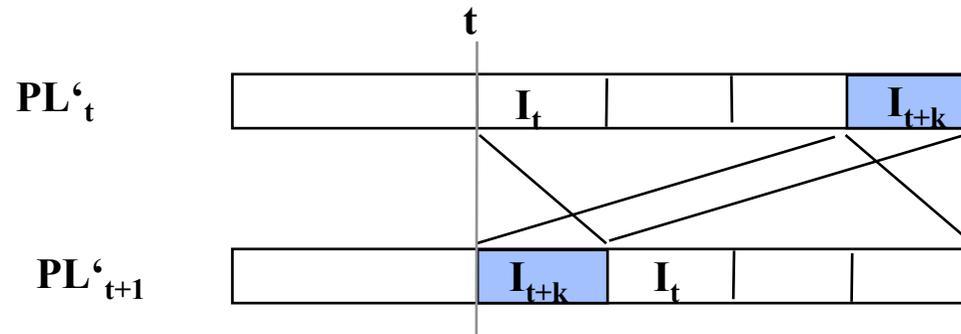
Beweis:

Ein brauchbarer Plan PL wird in einen nach EDF_u erzeugten Plan PL' umgeformt. Das geschieht im Raster von Zeitintervallen I_t der Länge Δt_G .

- Sei PL ein brauchbarer Plan.
- Sei PL'_{t_0} mit $t_0 = \min \{s_i \mid i \in T\}$ identisch mit PL und Ausgangspunkt der Konstruktion.
- Sei PL'_t ein bis zum Zeitpunkt t brauchbarer Plan, der nach EDF_u erzeugbar ist.

Der Plan PL'_t wird in Plan PL'_{t+1} durch Verschieben von Intervallen I_t der Länge Δt_G umgeformt.





Beweis (cont.)

- In PL'_t wird das Zeitintervall I_{t+k} $k \geq 0$ gesucht, dessen zugeordnete Task i die früheste Deadline besitzt.
- I_{t+k} rückt in PL'_{t+1} auf I_t . Dadurch verschieben sich alle dazwischenliegenden Zeitintervalle nach hinten.
- Das Planungsverfahren EDF_u hätte das Intervall ebenfalls an Task i vergeben, da diese die früheste Deadline besitzt (durch $edf(\dots)$).
- Der erzeugte Plan ist brauchbar, da alle Intervalle I_{t+1} bis I_{t+k} nur solchen Tasks zugeordnet sind, deren Deadlines $\geq d_i$ sind.