

# Überblick

## Planbarkeit

- Problemdefinition und Begriffe

- Komplexität

- Optimalität

- Planbarkeitsanalyse

- CPU-Auslastung

- Zeitbedarfsanalyse

- Antwortzeitanalyse

- Simulation

- Zusammenfassung

# Einhaltung von Terminen

## Taktgesteuerte Systeme $\leadsto$ konstruktiv

- ▶ alle Lastparameter sind à priori bekannt
- ▶ die Konstruktion einer Ablauftabelle trägt ihnen Rechnung
- ▶ Abhängigkeiten können berücksichtigt werden

☞ alle Termine werden eingehalten

- ▶ wenn eine zulässige Ablauftabelle erzeugt werden kann

## Vorranggesteuerte Systeme $\leadsto$ analytisch

- ▶ Lastparameter sind nicht vollständig bekannt
- ▶ Ablauf wird erst zur Laufzeit berechnet
- ▶ Abhängigkeiten müssen explizit gesichert werden

☞ Einhaltung von Terminen muss **explizit überprüft** werden

# Aufgabenstellung

Gegen sei eine Menge periodischer Aufgaben  $T_i$  mit

- $p_i$  der Periode (engl. *period*)
- $D_i$  dem relativen Termin (engl. *deadline*)
- $\phi_i$  der Phase (engl. *phase*)
- $e_i$  der maximalen Ausführungszeit (engl. *WCET*)

der jeweiligen Aufgabe.

👉 Fragestellung:

Ist diese Menge von Aufgaben **zulässig** (engl. *feasible* oder *schedulable*)?

# Zulässigkeit (engl. *Feasibility* oder *Schedulability*)

Ein Ablaufplan ist **gültig** (engl. *valid*), falls

- ▶ jeder CPU gleichzeitig max. ein Arbeitsauftrag zugeteilt wird.
- ▶ jeder Arbeitsauftrag gleichzeitig an max. eine CPU zugeteilt wird.
- ▶ kein Arbeitsauftrag vor seinem Auslösezeitpunkt eingeplant wird.
- ▶ einem Arbeitsauftrag entweder seine
  - ▶ tatsächliche oder seine
  - ▶ maximale Ausführungszeit zugeteilt wird.
- ▶ alle Abhängigkeiten eingehalten werden.

Ein Ablaufplan ist **zulässig** (engl. *feasible*), falls

- ▶ der Ablaufplan gültig ist und
- ▶ alle Arbeitsaufträge termingerecht eingeplant werden.

# Zulässigkeit (engl. *Feasibility* oder *Schedulability*, Forts.)

Eine Menge von Aufgaben ist **zulässig** (engl. *feasible* oder *schedulable*)

- ▶ hinsichtlich eines Ablaufplanungsalgorithmus,
- ▶ falls dieser Algorithmus immer einen zulässigen Ablaufplan erzeugt.

Die Entscheidung, ob eine Menge von Aufgaben planbar ist, hängt also

- ▶ vom verwendeten Ablaufplanungsalgorithmus ab, aber auch
- ▶ von den Eigenschaften der Aufgaben.

- ☞ häufig schränken Algorithmen die Eigenschaften von Aufgaben ein
  - ▶ dies vereinfacht die Frage der Zulässigkeit oft beträchtlich
  - ▶ durch **aufwendigere Analysen** können sie gelockert/aufgehoben werden

# Einschränkungen

- A1 Alle Aufgaben sind periodisch.
- A2 Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden.
- A3 Termine und Perioden sind identisch.
- A4 Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab.
- A5 Alle Aufgaben sind unabhängig von einander, d.h. die einzige gemeinsame Ressource ist die CPU und es existieren keine Einschränkungen hinsichtlich der Auslösezeiten der Arbeitsaufträge.
- A6 Der Overhead durch Unterbrechungen, Ablaufplanung oder Verdrängung ist vernachlässigbar.
- A7 Alle Aufgaben verhalten sich voll-präemptiv.

# Implikationen

Diese Einschränkungen haben Einfluss auf Anwendungen

## Betriebsmittel

- ▶ gemeinsame Betriebsmittel implizieren Synchronisation in ET Systemen
- ▶ Aufgaben sind nicht mehr unabhängig
- ☞ gemeinsame Betriebsmittel sind **nicht möglich!**

## Rangordnung

- ▶ Aufgaben erzeugen Eingaben für andere Aufgaben
- ▶ eine komplexe Aufgabe wird in mehrere einfachere Aufgaben geteilt
- ▶ Aufgaben sind nicht mehr unabhängig
- ☞ komplexe Aufgaben können **nicht geteilt werden!**

## Kommunikation

- ▶ synchroner Nachrichtenversand/-empfang
- ▶ Aufgaben sind nicht mehr unabhängig
- ☞ Aufgaben können **nicht synchron kommunizieren!**

# Die Lösung dieses Problems scheint einfach, ...

## Coffman [31]

Für jede Menge von  $n$  asynchronen, periodischen Aufgaben, die den Kriterien A1 - A7 entsprechen, findet der EDF Algorithmus einen gültigen Ablaufplan, **gdw** für die CPU-Auslastung gilt:

$$U = \sum_{i=1}^n u_i = \sum_{i=1}^n \frac{e_i}{p_i} \leq 1$$

- ▶ Eine Menge von Aufgaben
  - ▶ heißt **synchron**, falls  $\forall i : \phi_i = 0$ ,
  - ▶ ansonsten heißt sie **asynchron**.



# ... ist aber sehr schwierig!

- ▶ verzichtet man auf A3
  - ▶ Termine sind **kürzer** als die Perioden der Aufgaben.
  - ☞ **stark  $\mathcal{NP}$ -hart** (Baruah [32])
- ▶ verzichtet man auf A4
  - ▶ Aufgaben **legen sich schlafen** (engl. *self-suspension*).
  - ☞ **stark  $\mathcal{NP}$ -hart** (Richard [33])
- ▶ verzichtet man auf A5
  - ▶ Der **gegenseitige Ausschluss** wird durch Semaphore gesichert.
  - ☞ **stark  $\mathcal{NP}$ -hart** (Mok [34])
- ▶ verzichtet man auf A7
  - ▶ Harmonische, periodische Aufgaben sind **nicht verdrängbar**.
  - ☞ **stark  $\mathcal{NP}$ -hart** (Cai [35])

# Optimalität

Eine Möglichkeit Ablaufplanungsalgorithmen zu klassifizieren

## Optimalität (engl. *optimality*)

Ein Ablaufplanungsalgorithmus ist optimal (engl. *optimal*) für eine gewisse Klasse von Aufgaben, falls er für eine Menge von Aufgaben dieser Klasse einen gültigen Ablaufplan findet, sofern ein gültiger Ablaufplan existiert.

- ▶ solch ein Algorithmus stellt eine *Referenz* dar
  - ☞ schafft es dieser Algorithmus nicht, schafft es keiner!
- ▶ die (generelle) Zulässigkeit einer Menge von Aufgaben
  - ▶ kann auf die Zulässigkeit für diesen Algorithmus reduziert werden
  - ▶ sofern ein entsprechendes Kriterium existiert

# Optimalität des RM-Algorithmus

Der RM-Algorithmus ist optimal für Systeme, deren Aufgaben

- ▶ **synchron** sind und
- ▶ die Voraussetzungen **A1 - A7** erfüllen.

## Beweisidee (Baruah [36])

- ▶ gegeben sein ein System mit den Aufgaben  $\{T_1, T_2, T_3, \dots, T_n\}$
- ▶ mit Prioritäten  $T_1 \succ T_2 \succ \dots \succ T_n$  (nicht RM-konform)
- ▶ erzeugen einen zulässigen Ablaufplan
- ▶ Prioritäten können hinsichtlich RM umgeformt werden <sup>24</sup>
- ▶ ohne Zulässigkeit des Ablaufplans zu zerstören

---

<sup>24</sup>Man kann die Prioritäten zweier Aufgaben  $T_1$  und  $T_2$ , die das RM-Schema verletzen (für die also  $T_1 \succ T_2$  gilt, obwohl  $p_1 > p_2$ ), tauschen und ohne dabei die Zulässigkeit des Systems zu zerstören.

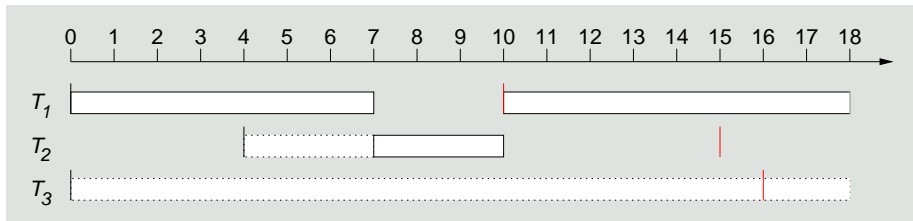
# Nicht-optimalität des RM-Algorithmus

Der RM-Algorithmus ist nicht optimal für Systeme, deren Aufgaben

- ▶ **asynchron** sind und
- ▶ die Voraussetzungen **A1 - A7** erfüllen.

**Beweis** (Baruah [36])

- ▶ Betrachte  $T_1 = (0, 7, 10)$ ,  $T_2 = (4, 3, 15)$ ,  $T_3 = (0, 1, 16)$
- ▶ RM:  $T_1 \succ T_2 \succ T_3$



▶  $T_3$  verpasst bei  $t_{16}$  seinen Termin

▶  $T_1 \succ T_3 \succ T_2$  würde funktionieren

# Optimalität des DM-Algorithmus

Der DM-Algorithmus ist optimal für System, deren Aufgaben

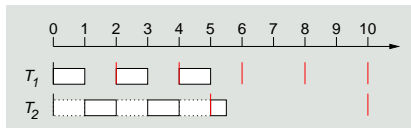
- ▶ **synchron** sind,
- ▶ die Voraussetzungen **A1**, **A2**, sowie **A4** - **A7** einhalten und
- ▶ für deren Termine  $D_i \leq p_i$  gilt.

**Beweisidee** (Baruah [36])

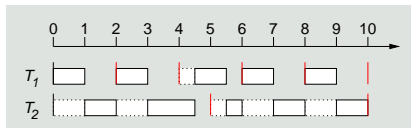
- ▶ Analog zum RM-Algorithmus

# Nicht-optimalität statischer Prioritäten

- ▶ betrachte  $T_1 = (2, 1)$  und  $T_2 = (5, 2.5)$
- ▶ sei  $T_1 \succ T_2$



$t_5$   $T_2$  verpasst Termin



$t_4$   $T_2 \succ T_1$

$t_{10}$  Hyperperiode

- ▶ vor dem Zeitpunkt  $t_4$  muss gelten  $T_1 \succ T_2$
  - ▶ zum Zeitpunkt  $t_4$  muss gelten  $T_2 \succ T_1$
- ☞ Widerspruch zur statischen Vergabe von Prioritäten

# Optimalität von EDF

Der EDF-Algorithmus ist optimal für Systeme, deren Aufgaben

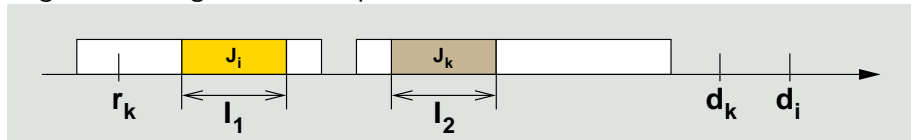
- ▶ beliebige Auslösezeiten
  - ▶ sporadisch/periodisch
  - ▶ synchron/asynchron
- und
- ▶ beliebige Deadlines
  - ▶ länger oder
  - ▶ kürzer als die entsprechende Periode
- besitzen, sowie
- ▶ die Voraussetzungen A2 und A4 - A7 erfüllen.

Beweis (Liu [2, S. 67])

- ▶ Jeder gültige Ablaufplan für solche Systeme
- ▶ lässt sich in einen EDF-Ablaufplan umformen.

# Umformung in einen EDF-Ablaufplan

Gegeben sei folgender Ablaufplan:



- ▶ betrachte alle Paare von Arbeitsaufträgen  $J_i$  und  $J_k$
- ▶ Arbeitsauftrag  $J_i$  wird im Intervall  $l_1$  und  $J_k$  im Intervall  $l_2$  eingeplant
- ▶ der Termin von  $J_k$  sei vor dem Termin von  $J_i$ :  $d_k < d_i$
- ▶ das Intervall  $l_1$  liegt komplett vor  $l_2$ :  $l_1 < l_2$

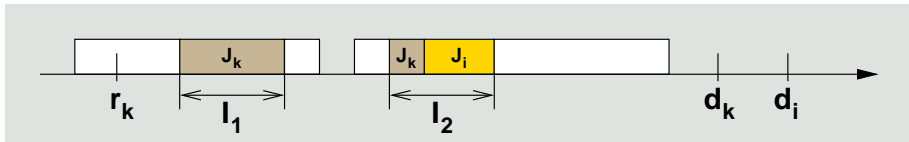
Fall 1:  $r_k > l_1$  ▶  $J_k$  kann nicht in  $l_1$  eingeplant werden  
☞ der Ablaufplan hat bereits EDF-Form



# Umformung in einen EDF-Ablaufplan Forts.

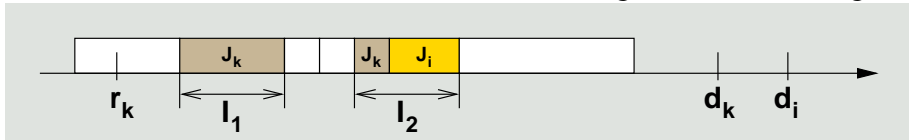
Fall 2: oBdA.  $r_k < l_1$ : tausche  $J_i$  und  $J_k$

- ▶  $d(l_1) < d(l_2)$
- ▶  $J_k$  wird in passende Stücke aufgeteilt (Verdrängung!)



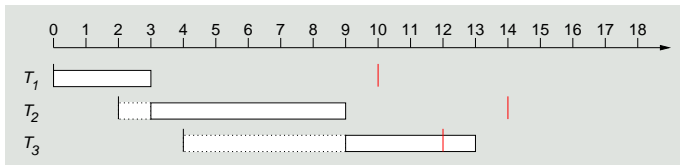
- ▶  $d(l_1) \geq d(l_2)$
- ▶ trivial

Fülle verbliebene Ruheintervalle durch Verschiebung von Arbeitsaufträgen

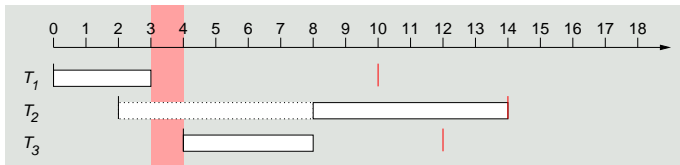


# Nicht-optimalität vorranggesteuerter Ablaufplanung für beliebige (in diesem Fall nicht-verdrängbare) Aufgaben

- ▶ betrachte  $T_1 = (0, 3, 10)$ ,  $T_2 = (2, 6, 14)$  und  $T_3 = (4, 4, 12)$
- ▶ der EDF-Algorithmus versagt bei diesem System



- ▶ obwohl ein zulässiger Ablaufplan existiert



- ▶ dieser lässt allerdings den Prozessor kurz untätig
- ▶ dieser Plan wird von keinem vorranggesteuerten Algorithmus gefunden!

# Planbarkeitsanalyse

Welche Möglichkeiten gibt es ...

- ▶ CPU-Auslastung (engl. *loading factor*)
  - ▶ Zu welchem Prozentsatz wird der Prozessor **maximal** beansprucht
  - ☞ bevorzugte Methode für **dynamische Prioritäten**
- ▶ Zeitbedarfsanalyse (engl. *processor demand*)
  - ▶ Wieviel Rechenzeit wird innerhalb eines Zeitintervalls benötigt?
  - ☞ neuere Methode für **dynamische Prioritäten**
- ▶ Antwortzeitanalyse (engl. *response time analysis*)
  - ▶ Wie lange benötigt eine Aufgabe **maximal** bis zu ihrer Fertigstellung?
  - ☞ präzise Methode für **statische Prioritäten**
- ▶ Simulation (engl. *simulation*)
  - ▶ Wird in einem bestimmten Intervall eine Deadline verfehlt?
  - ☞ bevorzugte Methode für **statische Prioritäten**

# Rechenzeitbedarf (engl. *processor demand*)

Gegeben sei eine Menge von Aufgaben  $\mathcal{T}$  und ein Zeitintervall  $[t_1, t_2)$ , dann ist der **Rechenzeitbedarf** dieser Aufgaben im Intervall  $[t_1, t_2)$ :

$$h_{[t_1, t_2)} = \sum_{t_1 \leq r_k, d_k \leq t_2} e_k$$

- ▶ das sind alle Aufgaben
  - ▶ deren Auslösezeitpunkt und
  - ▶ deren absoluter Termininnerhalb dieses Intervalls liegt.

## CPU-Auslastung (engl. *loading factor*)

Die **CPU-Auslastung** einer Menge von Arbeitsaufträgen während eines Intervalls  $[t_1, t_2)$ , ist der Anteil des Intervalls, der nötig ist, um diese Arbeitsaufträge auszuführen:

$$u_{[t_1, t_2)} = \frac{h_{[t_1, t_2)}}{t_2 - t_1}$$

Für eine Aussage über die Zulässigkeit einer Menge von Aufgaben  $T$  ist **absolute** CPU-Auslastung (engl. *absolute loading factor*) von Bedeutung. Dies ist die

- ▶ maximale CPU-Auslastung
- ▶ über alle Intervalle  $[t_1, t_2)$

$$u = \max_{0 \leq t_1 < t_2} u_{[t_1, t_2)}$$

# Zulässigkeitstest

## Liu und Layland [37]

Für jede Menge von  $n$  asynchronen, periodischen Aufgaben, die den Kriterien **A1** - **A7** entsprechen, findet der EDF Algorithmus einen gültigen Ablaufplan, **gdw** für die CPU-Auslastung gilt:

$$U = \sum_{i=1}^n u_i = \sum_{i=1}^n \frac{e_i}{p_i} \leq 1$$

- ▶ Coffman zeigt dies auch für asynchrone Aufgaben (S. 9- 8)
- ▶ schließlich zeigt Spuri [38] die “Optimalität” des EDF-Algorithmus
  - ▶ Aufgaben wie oben
  - ▶ synchron oder asynchron
  - ▶ Kriterium:  $U \leq 1$

# Beliebige Termine und Perioden

☞ die Bedingung A3 (S. 9- 6), soll gelockert werden

$$D_i \geq p_i$$

- ▶ die Kriterien von Layland/Liu und Coffman gelten nach wie vor [39]
- ▶ diese Kriterien sind **notwendig** und **hinreichend**

$$D_i < p_i$$

## Baruah [39]

Für eine hybride Menge von  $n$  Aufgaben  $T$ , findet der EDF-Algorithmus einen gültigen Ablaufplan, wenn gilt:

$$U = \sum_{i=1}^n \frac{e_i}{\min\{D_i, T_i\}} \leq 1$$

- ▶ **hybride** Menge von Aufgaben: periodische und sporadische Aufgaben
- ▶ diese Kriterium ist **nur hinreichend**!

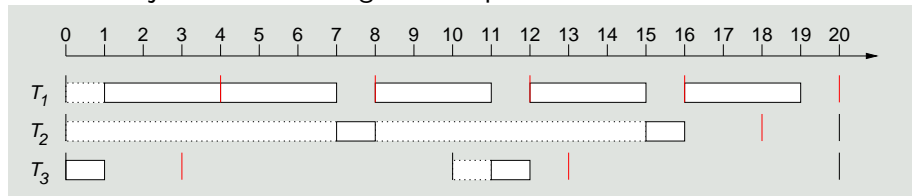
# Dieser Test ist pessimistisch ...

Betrachte folgende Aufgaben:  $T_1 = (0, 4, 3, 4)$ ,  $T_2 = (0, 20, 2, 18)$ ,  
 $T_3 = (0, 10, 1, 3)$

$$\blacktriangleright \sum_i \frac{e_i}{\min\{D_i, T_i\}} = \frac{3}{4} + \frac{2}{18} \frac{1}{3} = \frac{43}{36} > 1$$

☞ das System ist laut des Test (S. 9- 23) **nicht zulässig!**

Es existiert jedoch ein zulässiger Ablaufplan:



☞ eine geeignete Analysemethode für hybride Systeme wird benötigt



# Maximierung des Rechenzeitbedarfs

- ▶ hybrides System  $\leadsto$  entsprechendes synchrones, periodisches System
  - ▶ alle sporadischen Aufgaben
    - ▶ haben Phase  $\phi_i = 0$
    - ▶ treten mit ihrer maximalen Frequenz auf
- ▶ der Rechenzeitbedarf solcher Systeme ist im Intervall  $[0, t)$  maximal
  - ▶ man kann zeigen:  $\forall t_1, t_2 : h_{[t_1, t_2)} \leq h_{[0, t_2 - t_1)}$
- ▶ der Rechenzeitbedarf im Intervall  $[0, t)$  ist:

$$h(t) = \sum_{D_i \leq t} (1 + \lfloor \frac{t - D_i}{p_i} \rfloor) e_i$$

- ▶ alle Arbeitsaufträge, die vor  $t$  beendet sein müssen
- ▶ multipliziert mit der maximalen Anzahl ihrer Aktivierungen

# Zulässigkeitstest

Der EDF-Algorithmus, erzeugt für jede hybride Menge von Aufgaben einen zulässigen Ablaufplan, **gdw**:

$$\forall t : h(t) \leq t$$

- ▶ entspricht direkt dem Satz von Spuri (S. 9- 22)
- ▶ ist als Kriterium aber so nicht brauchbar
  - ▶ schließlich gibt es unendlich viele Intervalle  $[0, t)$
  - ▶ alle zu überprüfen ist einfach unmöglich

☞ Einschränkung der zu überprüfenden Intervalle

# Tätigkeitsintervalle

## Liu und Layland [37]

Kann der EDF-Algorithmus für eine Menge periodischer Aufgaben keinen zulässigen Ablaufplan finden, so wird ein Termin im ersten Tätigkeitsintervall verpasst.

- ▶ diese Eigenschaft wurde später auch gezeigt für
  - ▶ Mengen synchroner, periodischer Aufgaben mit  $D_i \leq p_i$  und
  - ▶ generische Mengen synchroner, periodischer Aufgaben
- ▶ sei  $L$  nun die Länge des ersten Tätigkeitsintervalls
  - ▶ die maximale Länge des zu prüfenden Intervalls ist nun beschränkt
- ▶  $h(t) \leq t$  muss nicht für alle Zeitpunkte in  $[0, t)$  geprüft werden
  - ▶  $\{e_1, e_2, \dots\} = mp_i + D_i; i = 1 \dots n, m = 0, 1, \dots$
  - ▶ wobei alle  $e_i < L$  genügen
  - ▶ Zeitbedarf erhöht sich nur bei Auslösung eines Arbeitsauftrags

# Ansatz

## Antwortzeit

- ▶ Zeitdauer zwischen Auslösezeit und Terminationszeitpunkt (S. 4- 9)

## Idee

- ▶ Terminationszeitpunkt vor dem **absoluten** Termin
- ▶ Antwortzeit  $\omega_i$  kürzer als der **relative** Termin  $D_i$
- ☞ für jede Aufgabe  $T_i : \omega_i \leq D_i$

## Voraussetzungen

- ▶ die Bedingungen **A1 - A7** müssen eingehalten werden
- ▶ Konzept ist jedoch erweiterbar

## Probleme

- ☞ Wie berechnet man die Antwortzeit?
- ☞ Wann wird die **maximale** Antwortzeit erreicht?

# Berechnung der Antwortzeit

- ▶ die Antwortzeit  $\omega_i$  einer Aufgabe  $T_i$  berechnet sich zu

$$\omega_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k; 0 < t \leq p_i$$

- ▶ die Aufgabe endet, bevor das Ereignis erneut eintritt
- ▶ setzt sich zusammen, aus
  - ▶ der WCET von  $T_i$  selbst und
  - ▶ den WCETs der Aufgaben mit höherer Priorität  $T_1, \dots, T_{i-1}$

- ▶ zu prüfen ist nun

$$\omega_i(t) \leq t;$$

$$t = jp_k; \quad k = 1, 2, \dots, i; \quad j = 1, 2, \dots, \lfloor \min(p_i, D_i) / p_k \rfloor$$

- ▶ Zeitbedarf erhöht sich nur bei Auslösung von Aufgaben höherer Priorität
  - ▶ bis das Ereignis erneut eintritt oder der Termin der Aufgabe erreicht ist
- ☞ ist die Ungleichung für **einen** Zeitpunkt  $t$  erfüllt, ist  $T_i$  **zulässig**

# Beispiel: Berechnung der maximalen Antwortzeit

mit den Aufgaben  $T_1 = (\phi_1, 3, 1)$ ,  $T_2 = (\phi_2, 5, 1.5)$ ,  $T_3 = (\phi_3, 7, 1.25)$ ,  $T_4 = (\phi_4, 9, 0.5)$

► Antwortzeit  $\omega_1$  von  $T_1$

►  $\omega_1(3) = 1 \leq 1$

☞  $T_1$  ist zulässig

► Antwortzeit  $\omega_2$  von  $T_2$

►  $\omega_2(3) = 1.5 + \lceil \frac{3}{3} \rceil = 2.5 \leq 3$

☞  $T_2$  ist zulässig

► Antwortzeit  $\omega_3$  von  $T_3$

►  $\omega_3(3) = 1.25 + \lceil \frac{3}{3} \rceil 1 + \lceil \frac{3}{5} \rceil 1.5 = 3.75 > 3$

►  $\omega_3(5) = 1.25 + \lceil \frac{5}{3} \rceil 1 + \lceil \frac{5}{5} \rceil 1.5 = 4.75 \leq 5$

☞  $T_3$  ist zulässig

► Antwortzeit  $\omega_4$  von  $T_4$

►  $\omega_4(3) = 0.5 + \lceil \frac{3}{3} \rceil 1 + \lceil \frac{3}{5} \rceil 1.5 + \lceil \frac{3}{7} \rceil 1.25 = 4.25 > 3$

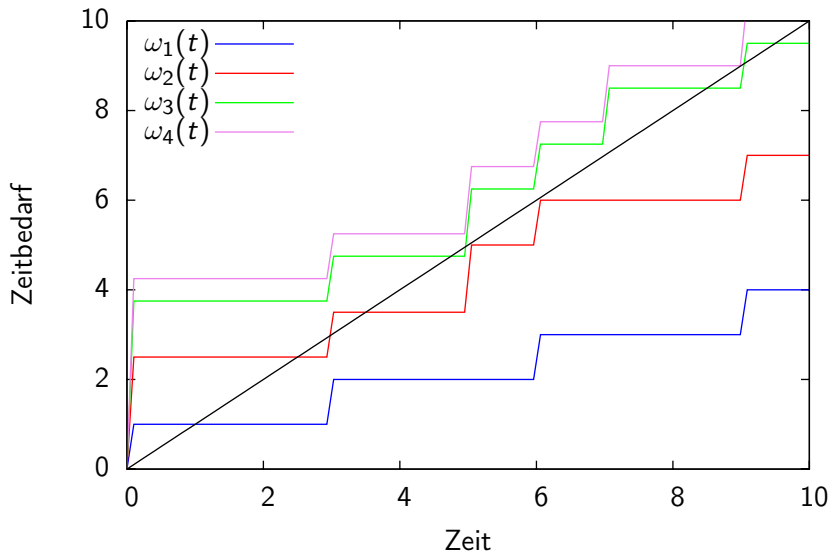
►  $\omega_4(5) = 0.5 + \lceil \frac{5}{3} \rceil 1 + \lceil \frac{5}{5} \rceil 1.5 + \lceil \frac{5}{7} \rceil 1.25 = 5.25 > 5$

►  $\omega_4(6) = 0.5 + \lceil \frac{6}{3} \rceil 1 + \lceil \frac{6}{5} \rceil 1.5 + \lceil \frac{6}{7} \rceil 1.25 = 6.75 > 6$

►  $\omega_4(7) = 0.5 + \lceil \frac{7}{3} \rceil 1 + \lceil \frac{7}{5} \rceil 1.5 + \lceil \frac{7}{7} \rceil 1.25 = 7.75 > 7$

►  $\omega_4(9) = 0.5 + \lceil \frac{9}{3} \rceil 1 + \lceil \frac{9}{5} \rceil 1.5 + \lceil \frac{9}{7} \rceil 1.25 = 9.00 \leq 9$

☞  $T_4$  ist zulässig

Zeitbedarfsfunktionen der Aufgaben  $T_1$ ,  $T_2$ ,  $T_3$  und  $T_4$ 

# Fadensynchronisation

Erweiterung des Konzepts, Aufhebung von Bedingung A5

- ▶ Arbeitsaufträge können blockiert werden
  - ▶ wenn sie Betriebsmittel nachfragen,
  - ▶ die bereits von Arbeitsauftrag niedriger Priorität belegt sind.

☞ das muss bei der Bestimmung der Antwortzeit berücksichtigt werden

$$\omega_i(t) = e_i + bt_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k; 0 < t \leq p_i$$

- ▶ die Blockadezeit  $bt_i$  der Aufgabe  $T_i$  verlängert die Antwortzeit

☞ die Blockadezeit hängt vom Synchronisationsprotokoll ab

- ▶ NPCS (S. 7- 13):  $bt_i = \max_{i+1 \leq k \leq n}(cs_k)$
- ▶ Priority Inheritance (S. 7- 18):  $bt_i = \min(n, j) \max_{i+1 \leq k \leq n}(cs_k)$ 
  - ▶  $n$  Betriebsmittel und Konflikt mit  $j$  Aufgaben niedrigerer Priorität
- ▶ Priority Ceiling (S. 7- 22):  $bt_i = \dots$



# Wann wird die Antwortzeit maximal?

- ▶ **kritischer Zeitpunkt** (engl. *critical instant*)
  - ▶ Auslösung eines Arbeitsauftrags an seinem kritischen Zeitpunkt
  - ☞ die **maximale Antwortzeit** wird erreicht
- ▶ der Arbeitsauftrag einer Aufgabe  $T_i$ , der an einem kritischen Zeitpunkt ausgelöst wird
  - ▶ hat die **maximale Antwortzeit** aller Arbeitsaufträge in  $T_i$ 
    - ▶ falls diese ihre Termine einhalten
  - ▶ **verpasst seinen Termin**
    - ▶ falls irgendein Arbeitsauftrag in  $T_i$  seinen Termin verpasst
- ▶ Liu und Layland [37]: ein kritischer Zeitpunkt
  - ▶ in Systemen mit **statischen** Prioritäten tritt ein
  - ▶ falls **zusammen** mit einem Arbeitsauftrag einer Aufgabe  $T_i$
  - ▶ Arbeitsaufträge aller Aufgaben **höherer Priorität**  $T_1, \dots, T_{i-1}$  ausgelöst werden
- ☞ In Systemen mit **dynamischen** Prioritäten
  - ▶ lässt sich ein solcher kritischer Zeitpunkt **nicht** identifizieren,
  - ▶ was die Antwortzeitanalyse ungemein **aufwendig** macht.

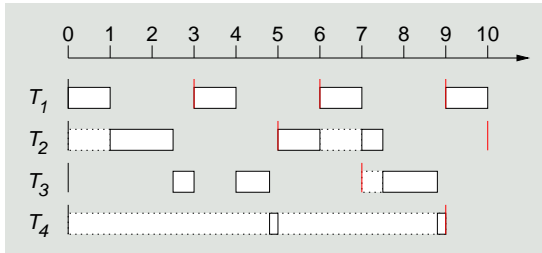
# Simulation

- Vorteil**
- ▶ Analysemethoden sind sehr **komplex** und schwer verständlich
  - ▶ Planungsalgorithmen sind relativ **einfach**
  - ☞ Konstruktion eines Ablaufplans!

- Voraussetzung**
- ▶ Simulation muss den *worst case* treffen

- Lösung**
- ▶ Simulation muss am kritischen Zeitpunkt beginnen

Vergleiche Beispiel auf S. 9- 30



☞ Methode, die in vielen industriellen Werkzeugen vorzufinden ist

# Resümee

**Zulässigkeit** einer Menge von Aufgaben

- ▶ **zentrale Frage**: ist eine Menge von Aufgaben zulässig?
- ▶ d.h. können alle Aufgaben termingerecht abgearbeitet werden?

**Komplexität** dieser Fragestellung

- ▶ Entscheidung der Zulässigkeit ist sehr, sehr schwierig
- ▶ nur Sonderfälle sind in polynomieller Laufzeit berechenbar
- ▶ nahezu alle Varianten des Problems sind **stark  $\mathcal{NP}$ -hart**

**Optimalität** eines Ablaufplanungsalgorithmus

- ▶ ist ein Ablaufplanungsalgorithmus **optimal**,
- ▶ findet er einen zulässigen Ablaufplan für eine bestimmte Aufgaben,
- ▶ **gdw** ein zulässiger Ablaufplan existiert.

**Planbarkeitsanalyse** für dynamische und statische Prioritäten

- ▶ dynamische Prioritäten  $\leadsto$  CPU-Auslastung, Zeitbedarfsanalyse
- ▶ statische Prioritäten  $\leadsto$  Antwortzeitanalyse, Simulation