

# Echtzeitsysteme

## Zeitgesteuerte Ablaufplanung periodischer Echtzeitsysteme

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de/Lehre/WS19/V-EZS/>

25. November 2019



# Fragestellungen

- Wie bestimmt man eine geeignete **Ablauf-tabelle** für eine gegebene Menge von Aufgaben?
- **Manuelle Bestimmung** zyklischer Ablaufpläne
  - Warum bestimmt man Ablaufpläne manuell?
  - Gibt es Leitlinien, um die manuelle Erstellung zu unterstützen?
- **Algorithmische Bestimmung** zyklischer Ablaufpläne
  - **Heuristische Verfahren**
  - **Optimale Verfahren**
- Wie **flexibel** sind zyklische Ablaufpläne?

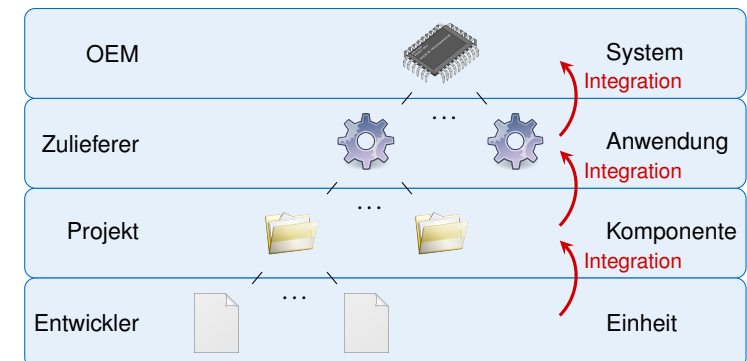


# Gliederung

- 1 Entwicklung – Herangehensweise
  - Ablaufplanung – Bottom-Up
  - Spezifikation – Top-Down
- 2 Manuelle Einplanung
  - Struktur zyklischer Ablaufpläne
- 3 Algorithmische Einplanung
  - List-Scheduling-Verfahren
  - Branch&Bound-Algorithmen
- 4 Moduswechsel
- 5 Zusammenfassung



# Ablaufplanung – Teil des Entwicklungsprozesses



- Der Integrationsprozess verläuft **Bottom-Up**:
  - 1 Bündelung von **Softwareeinheiten** (engl. *unit*) zu **Komponenten**
  - 2 **Komponenten** implementieren Arbeitsaufträge in **Anwendungen**
  - 3 Einplanung der **Arbeitsaufträge** in einer statischen **Ablauf-tabelle**



## Herausforderung Integration

Die **Ablaufplanung** ist **finale Schritt** der Systemerstellung

- ⚠ Inhärent abhängig von den bereitgestellten Edukten
- **SW-Einheiten und -Komponenten**: **Maximale Ausführungszeiten**
- **Anwendung**: Spielraum der Ablaufplanung durch Abbildung  
**Komponenten** → **Arbeitsaufträge** → **Aktivitätsträger**

⚠ Erstellung von Software-Einheit, -Komponente, Anwendung und System fällt meist in **verschiedene Zuständigkeitsbereiche**:

- Softwarekomponenten werden zugekauft (z.B. Betriebssystem, Mathematik- oder Kryptographiebibliothek)
- Zulieferer fügt diese Komponenten zu einer Anwendung zusammen (z.B. ABS, Fahrspurassistent)
- OEM fertigt schließlich das endgültige Produkt (z.B. ein Auto)

Entscheidend ist das **Verhalten des Gesamtsystems**



## Herausforderung Integration (Forts.)

Wenn funktionale Schnittstellenbeschreibungen nicht ausreichen

⚠ **Nachträgliche Änderungen bedeuten beträchtlichen Aufwand**

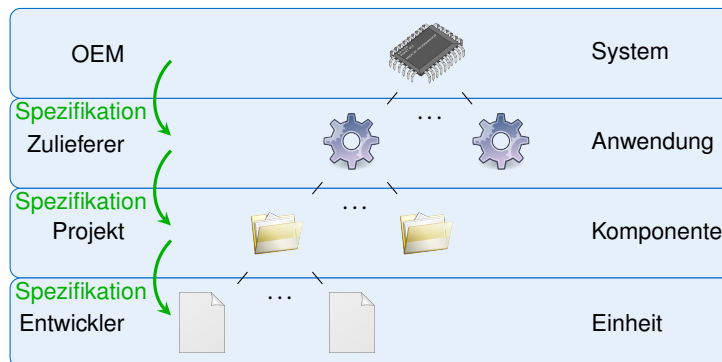
- **Beeinflussung** des Laufzeitverhaltens durch Änderung der
  - Maximalen Ausführungszeit (WCET,  $e$ )
  - Abbildung von Arbeitsaufträgen auf Aktivitätsträger
  - Abhängigkeiten zwischen Arbeitsaufträgen
- Überlast z. B. durch ineffiziente Implementierung bzw. Strukturierung
  - Keine **zulässigen** Ablaufpläne ermittelbar (siehe III-2/31)
  - **Nachbesserungen** falls die Ablaufplanung fehlschlägt

Spezifikation des zeitlichen Verhaltens von Softwarekomponenten

- Beispielsweise durch vorgezogene/iterative Analyse/Ablaufplanung



## Spezifikation zeitlichen Verhaltens



Die Spezifikation erfolgt **Top-Down**:

- 1 OEM weist den Anwendungen Zeitschlitze im Ablaufplan zu
- 2 Anwendungen **verteilen** die Rechenzeit auf Softwarekomponenten
- 3 Komponenten und Einheiten müssen mit ihrer Rechenzeit **haushalten**



## Spezifikation zeitlichen Verhaltens (Forts.)

- Idee der **Rahmenkonstruktion** (engl. *framework*)
  - **Hollywood-Prinzip**: „Don't call us, we'll call you!“
  - OEM muss die Anwendungsstruktur **vorgeben**

Globale Planung von zeitlichen Abläufen

- Zeitschlitze und deren Einhaltung werden zu lokalen Belangen
- Problemlösung im selben Zuständigkeitsbereichs möglich

⚠ Erstellung eines globalen Ablaufplans **erfordert Vorabwissen**

- Rückgriff auf **zurückliegende Entwicklungsprojekte**
- Erkenntnis aus der Entwicklung von **Prototypen**

Leitlinien für die Erstellung **gut strukturierter, zyklischer Ablaufpläne** sind wünschenswert und sinnvoll



## Gliederung

- 1 Entwicklung – Herangehensweise
  - Ablaufplanung – Bottom-Up
  - Spezifikation – Top-Down
- 2 Manuelle Einplanung
  - Struktur zyklischer Ablaufpläne
- 3 Algorithmische Einplanung
  - List-Scheduling-Verfahren
  - Branch&Bound-Algorithmen
- 4 Moduswechsel
- 5 Zusammenfassung



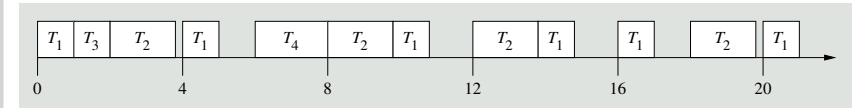
## Regelmäßigkeit zyklischer Abläufe (engl. *cyclic executive*) [3]

Die Busy Loop wird erwachsen...



Einplanungsentscheidungen periodischer Aufgaben können in **unregelmäßigen Abständen** wirksam werden (vgl. IV-1/4)

- Beispiel: 0, 1, 2, 4, 6, 8, 10, 12, 14, 16, 18



■ **Regularität** von Einplanungsentscheidungen trägt wesentlich zu **Determinismus** und **Analysierbarkeit** bei



Erfordert **gute Anordnung** eines zyklischen Ablaufplans (Strukturiertheit)

- Einplanungsentscheidungen nicht zu beliebigen Zeitpunkten treffen



## Rahmen (engl. *frames*)

Strukturelemente von zyklischen Ablaufplänen

### Die Rahmenlänge $f$

Zeitpunkte von Einplanungsentscheidungen unterteilen die Echtzeitachse in **Intervalle fester Länge  $f$**  (engl. *frame size*)



Entscheidungen erfolgen nur am Rahmenanfang

- Aufträge einer Aufgabe werden am Anfang eines Rahmens ausgelöst

⚠ Innerhalb eines Rahmens ist Verdrängung ausgeschlossen

→ Phase einer periodischen Aufgabe ist ein Vielfaches von  $f$

- Verantwortungsbereich des *Dispatchers* erweitert sich

- Einlastung von Arbeitsaufträgen am Rahmenanfang
  - **Überwachung/Durchsetzung** von Einplanungsentscheidungen
    - Wurde ein eingeplanter Auftrag tatsächlich **ausgelöst**?
    - Ist dieser Arbeitsauftrag auch zur Ausführung **bereit**?
    - Liegt eine **Terminverletzung** vor → steht eine Fehlerbehandlung an?
- Beeinflusst im hohen Maße den Wert für  $f$



## Randbedingungen für die Rahmenlänge

Rahmenlänge  $f$  für  $n$  Aufgaben genau richtig wählen...



Terminüberwachung ermöglichen →  $f$  **hinreichend kurz**

- 1 Erfordert eine rechtzeitige Auslösung:  $f \leq p_i$ , für alle  $1 \leq i \leq n$
- 2 Möglich unter der Bedingung:  $2f - ggT(p_i, f) \leq D_i$ , für alle  $1 \leq i \leq n$

- Rahmen passend auf die anstehenden Aufgaben verteilen
  - Mindestens ein Rahmen zwischen Auslösung und Termin jedes Auftrags



Strukturbedingungen einhalten

- 3  $f$  teilt die Hyperperiode  $H$  so, dass gilt:  $\lfloor p_i/f \rfloor - p_i/f = 0$ , für ein  $1 \leq i \leq n$ 
  - Ermöglicht die zyklische Ausführung des Ablaufplans

- Das Intervall  $H$  heißt **großer Durchlauf** (engl. *major cycle*)
  - Intervall der Länge  $f$  heißt **kleiner Durchlauf** (engl. *minor cycle*)



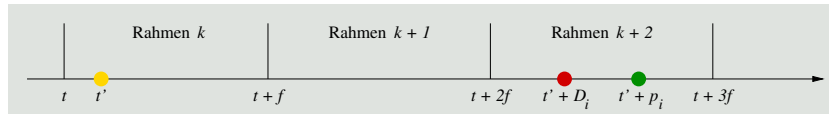
Das Regelwerk wird auf Seite 14 mit der 4. Regel abgeschlossen



## Randbedingungen für die Rahmenlänge (Forts.)

Platzierung einer Aufgabe auf der Echtzeitachse

- Feststellung eines passenden Bereichs für  $f$  von  $T = (p_i, e_i, D_i)$ :<sup>1</sup>



- $t$ : Anfang des Rahmens  $k$  in dem ein Auftrag in  $T_i$  ausgelöst wird
- $t'$ : Zeitpunkt der Auslösung des betreffenden Auftrags
- Rahmen  $k + 1$  erlaubt die Kontrolle des bei  $t'$  ausgelösten Jobs
  - Der Rahmen sollte daher zwischen  $t'$  und  $t' + D_i$  des Jobs liegen
- Dies ist erfüllt, wenn gilt:  $t + 2f \leq t' + D_i$  bzw.  $2f - (t' - t) \leq D_i$ 
  - $t' - t$  ist mindestens größter gemeinsamer Teiler von  $p_i$  und  $f$  [3]

<sup>1</sup> Befindet sich  $f$  in diesem Bereich, gibt es wenigstens einen Rahmen zwischen der Auslösezeitpunkt und dem Termin jedes Arbeitsauftrags der betreffenden Aufgabe.



## Randbedingungen für die Rahmenlänge (Forts.)

Rahmenlänge  $f$  für  $n$  Aufgaben genau richtig wählen...



Verdrängung von Aufträgen vermeiden  $\rightarrow f$  hinreichend lang

- 4 Erfüllt, wenn gilt:  $f \geq \max(e_i^f)$ , für  $1 \leq i \leq H/f$

- $e_i^f$  gibt die WCET aller Aufträge im Rahmen  $i$  an
- Jeder Auftrag läuft in der durch  $f$  gegebenen Zeitspanne komplett durch
- Erste Abschätzung nach unten:  $f \geq \max(e_i)$ , für  $1 \leq i \leq n$



Ermittlung von  $\max(e_i^f)$  erfolgt nachgelagert

- Regel 1-3 sind notwendige Bedingungen  $\rightarrow$  zu testende Rahmenlängen
- Fragestellung: Welcher Auftrag kommt in welchem Rahmen zu liegen?
  - Kann erst durch konkrete Ablaufplanung beantwortet werden
- Iterativer Prozess  $\rightarrow$  Wiederholung für jedes potentielle  $f$

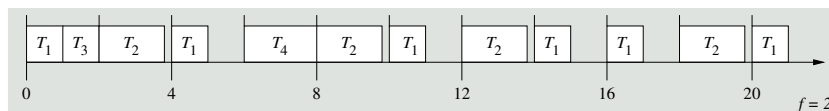


Nicht immer ist eine Lösung möglich  $\rightarrow$  Aufträge zerschneiden (S.16)



## Randbedingungen für die Rahmenlänge (Forts.)

$T_i = (p_i, e_i)$ ,  $D_i = p_i$  und  $\phi_i = 0$

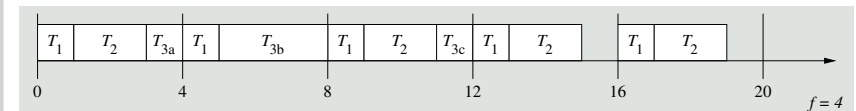


- Beispiel:  $T_1 = (4, 1)$ ,  $T_2 = (5, 1.8)$ ,  $T_3 = (20, 1)$ ,  $T_4 = (20, 2)$ 
  - Ganzzahlige Teiler der Hyperperiode sind 2, 4, 5, 10 und 20 ( $H = 20$ )
  - Nur  $f = 2$  erfüllt jedoch alle Bedingung 1-3 (Folie 12) zugleich
  - $FI \geq 2$  (Bedingung 4) gilt  $\rightarrow$  alle Aufträge eines jeden Rahmens laufen durch
- Weiteres Beispiel:  $T_x = (15, 1, 14)$ ,  $T_y = (20, 2, 26)$ ,  $T_z = (22, 3)$ 
  - Mögliche Rahmenlängen in  $H$ : 3, 4, 5, 10, 11, 15, 20, 22 ( $H = 660$ )
  - Nur  $f = 3, 4$  oder  $5$  erfüllt Bedingungen
  - Planung ergibt dass  $FI \geq 3$  gilt



## Konflikte und deren Auflösung

Taskparameter zugunsten einer guten Ablaufplananordnung korrigieren



Arbeitsaufträge sind in Scheiben zu schneiden, falls nicht alle Randbedingungen erfüllbar sind

- Beispiel:  $T_1 = (4, 1)$ ,  $T_2 = (5, 2, 7)$ ,  $T_3 = (20, 5)$ :
  - $f \geq \max(e_i^f)$  gilt für  $f \geq 5$  und  $2f - ggT(p_i, f) \leq D_i$  gilt für  $f \leq 4$
- $T_3 = (20, 5)$  ist aufzuteilen in  $T'_3 = \{(20, 1), (20, 3), (20, 1)\}$ 
  - Drei Teilaufgaben  $T_{3a} = (20, 1)$ ,  $T_{3b} = (20, 3)$ ,  $T_{3c} = (20, 1)$
  - $\rightarrow$  Das resultierende System hat fünf Tasks und die Rahmenlänge  $f = 4$
- $T_3 = (20, 5)$  in zwei Teilaufgaben aufzuteilen, bleibt erfolglos:
  - $\{(20, 4), (20, 1)\}$  geht nicht, wegen  $T_1 = (4, 1)$
  - $\{(20, 3), (20, 2)\}$  geht nicht, da für  $T_{3b} = (20, 2)$  kein Platz bleibt

!?



## Entstehungsprozess eines zyklischer Ablaufplans

Gegenseitige Abhängigkeit von Entwurfsentscheidungen

- 1 Rahmenlänge festlegen (vgl. IV-3/12)
  - Mögliche Konflikte erkennen
- 2 Arbeitsaufträge in Scheiben aufteilen (vgl. IV-3/16)
  - Insbesondere kann dies zur Folge haben, andere Programm- bzw. Modulstrukturen herleiten zu müssen
  - Die erforderlichen **Programmtransformationen** geschehen bestenfalls (semi-) automatisch durch spezielle Compiler
- 3 Arbeitsaufträge in die Rahmen platzieren

⚠ Rahmenlänge **querschneidende nicht-funktionale Eigenschaft**



## Vor-/Nachteile zyklischer Ablaufpläne



Zyklisches Ablaufmodell liefert **wohlgeordnete Ablaufpläne**

- Eine feste Rahmengröße mit definierten Schranken
- Ablaufplanung (→ Zuteilung Aufträge zu Rahmen) findet **offline** statt  
→ **Einlastung** und **Terminüberwachung** zu definierten Zeitpunkten
- **Busy-Loop-Verhalten** innerhalb eines Rahmens (vgl. IV-1/12)
  - Sequentielle, kooperative Abarbeitung der Aufträge
  - Keine individuelle **Laufzeitüberwachung** und **Ausnahmebehandlung**
  - Anfällig für Jitter und mangelnde Periodizität
- + **Niedrige Verwaltungsgemeinkosten**
  - **Einlastung** und **Terminüberwachung** findet nur an den Rahmengrenzen statt
  - Keine Verdrängung (engl. **preemption**) (vgl. III-2/13)
  - Minimalistisches Laufzeitsystem (Dispatcher+Terminprüfung genügt)
- + **Hohe Vorhersagbarkeit**
  - Einziger Interrupt ist der Zeitgeber an den Rahmengrenzen  
→ **Unterbrechungsfreier Durchlauf** innerhalb der Rahmen  
→ Vereinfacht die WCET-Analyse ungemein (vgl. Kapitel III-3)



## Gliederung

- 1 Entwicklung – Herangehensweise
  - Ablaufplanung – Bottom-Up
  - Spezifikation – Top-Down
- 2 Manuelle Einplanung
  - Struktur zyklischer Ablaufpläne
- 3 Algorithmische Einplanung
  - List-Scheduling-Verfahren
  - Branch&Bound-Algorithmen
- 4 Moduswechsel
- 5 Zusammenfassung



## Handarbeit ist mühsam!



Statische Ablaufpläne werden sehr schnell **umfangreich**

- Ablauftabellen werden zur Hyperperiode aufgeblasen
- Beispiel:  $T_1 = (20, 3)$ ,  $T_2 = (15, 2)$ ,  $T_3 = (2, 0.25)$ 
  - Resultiert in einer Ablauftabelle mit 37 Einträgen
  - Fügt man  $T_4 = (40, 3)$  hinzu, werden daraus 77 Einträge
- Algorithmische Ablaufplanung ist schwierig (vgl. IV-2/33)
  - Im Allgemeinen ist Ablaufplanung **stark NP-hart**



**Automatisierte Berechnung** von Ablauftabellen

- Computer sind dafür da, große Datenmengen schnell zu verarbeiten
- Exponentielles Wachstum der Laufzeit ist auch für Computer fatal
  - Entwicklung **heuristischer** und **optimaler Verfahren**
- Verfahren haben zunehmende Praxisrelevanz
  - Verbesserte WCET-Analyse ebnet den Zugang zu solchen Verfahren



## Algorithmische Lösungsverfahren – Überblick

Grundlegende Aufgabenstellung: Berechnung einer statischen Ablaufabelle für eine Menge periodischer Aufgaben (vgl. IV-2/28)

Existierende Verfahren erfüllen deutlich mehr Anforderungen:

- Berücksichtigung gerichteter und ungerichteter Abhängigkeiten
- Verteilte Systeme und Mehrkern- sowie Mehrprozessorsysteme
- Beschleunigung von Arbeitsaufträgen durch Duplizierung

...

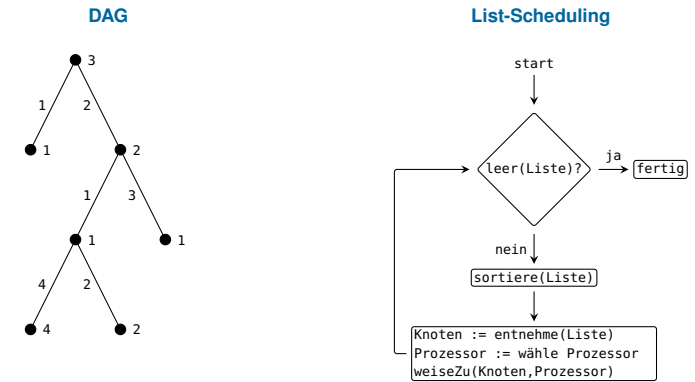
Kategorien algorithmischer Lösungsverfahren:

- **Heuristiken**  $\rightarrow$  effizient, finden u.U. keine (existierende) Lösung Folie 22
  - Genetische Algorithmen, List-Scheduling [4], ...
- **Optimale Verfahren**  $\rightarrow$  finden eine Lösung, sofern existierend Folie 25
  - Lineare Programmierung [7], Branch&Bound [1], ...
  - $\rightarrow$  Exponentiell wachsende Laufzeit im schlimmsten Fall



## List-Scheduling

- List-Scheduling als Klasse heuristischer Verfahren, die gerichtete, azyklische Graphen (engl. *directed acyclic graph, DAG*) ordnen [6]



Knotengewichte  $\sim$  Berechnung ( $\rightarrow$  Jobs)

Kantengewichte  $\sim$  Kommunikation

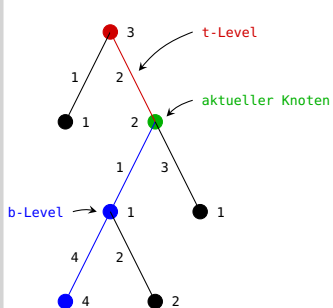
verwaltet Knoten in einer sortierten Liste



## b-Level und t-Level

Grundlage der Sortierkriterien beim List-Scheduling

Position eines Knotens im DAG bestimmt die Position in der Liste



**t-Level** Der längste Pfad von einer Wurzel zum aktuellen Knoten

- Variiert während der Ablaufplanung
- Kommunikation *verschwindet* u.U.

**b-Level** Der längste Pfad vom aktuellen Knoten zu einem Endknoten

- Während der Planung i.d.R. konstant

**Pfadlänge** Summe der Knoten- und Kantengewichte ohne aktuellen Knoten

! Gewichtung von t-Level und b-Level Algorithmus-spezifisch

b-Level  $\sim$  Knoten auf dem **kritischen Pfad** werden bevorzugt

t-Level  $\sim$  Plant Knoten entlang der **topologischen Ordnung**



## Beispiele

**HLFET (Highest Level First with Estimated Times)** [2]

- Vernachlässigt Kommunikation  $\sim$  Kantengewichte = 0
- Verwendet b-Level als alleiniges Sortierkriterium

**ISH (Insertion Scheduling Heuristic)** [5]

- Sortierkriterium  $\sim$  wie HLFET
- Evtl. entstehen Intervalle der Untätigkeit (engl. *idle time slots*)
  - Auffüllung durch Knoten aus der Bereitliste falls möglich

**DLS (Dynamic Level Scheduling)** [8]

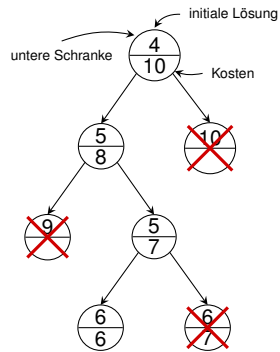
- Sortierkriterium: *Dynamic Level*
  - Differenz: b-Level – frühestem Startzeitpunkt für einen Prozessor
  - Wird nach jedem Durchlauf neu berechnet
- Bereitliste enthält zunächst nur die Wurzelknoten des DAG
  - Hinzufügen von Knoten nach Einplanung des Vorgängers



## Optimale Suche mittels Branch&Bound

Planungsproblem  $\rightarrow$  Suchproblem in einem Suchbaum

- Potentiell Betrachtung des **kompletten Suchraums**  $\leadsto$  exp. Laufzeit



- 1 Berechne eine **initiale Lösung**
  - $\rightarrow$  Ein (evtl. unzulässiger) Ablaufplan
  - Bestimmung der **tatsächlichen Kosten**
  - Bestimmung einer **unteren Schranke**
- 2 Leite **verbesserte initiale Lösungen** ab
  - $\rightarrow$  Branch-Schritt
  - Verwerfen ungeeigneter Lösungen
  - $\rightarrow$  Reduktion des Suchraums: Bound-Schritt
- 3 Wiederhole diese Schritte ...  
bis zur **optimalen Lösung**
  - oder klar ist, dass **keine Lösung existiert**

⚠ Um Optimalität zu erreichen, müssen im Branch-Schritt **alle Möglichkeiten** ausgeschöpft werden, eine Lösung zu verbessern



## Branch&Bound – Statische Ablaufplanung

Wo kommen die initiale Lösung, die Kosten und die untere Schranke her?

- **Initiale Lösungen** sind bereits vollständige gültige Ablaufpläne
  - Diese können aber noch Termine verletzen, sind also nicht zulässig
  - $\rightarrow$  Ein Verfahren für deren Bestimmung wird benötigt
- **Kosten** einer Lösung sind die vorhandenen **Verspätungen**
  - $\rightarrow$  Maximale Terminüberschreitung aller Jobs
- **Untere Schranken** durch Vereinfachung des Planungsproblems
  - Ohne die Optimalität des Algorithmus zu verletzen
- **Verbesserung** durch Manipulation des Planungsproblems
  - Arbeitsauftrag mit der größten Terminüberschreitung früher einplanen
  - $\rightarrow$  Verspätung ohne Verletzung der ursprünglichen Vorgaben reduzieren

**Ziel:** Eine Lösung finden, deren Kosten kleiner oder gleich 0 sind. Der zugehörige Ablaufplan ist daher zulässig.



## Der Algorithmus von Abdelzaher und Shin [1]

Ein Beispiel für einen Branch&Bound-Algorithmus für die statische Ablaufplanung

Leistungsumfang des Algorithmus

- Auslösezeiten, Ausführungszeiten, Termine
- Gerichtete und ungerichtete Abhängigkeiten
- Einkern-, Mehrkern- und Mehrprozessorsysteme
- Verteilte Systeme und nachrichtenbasierte Kommunikation

⚠ Der Algorithmus führt jedoch **keine Allokation** durch!

**Initiale Lösung:** Globaler EDF-Algorithmus (G-EDF)

- Erweitert um die Behandlung ungerichteter Abhängigkeiten
- $\rightarrow$  Für obiges Planungsproblem **nicht optimal**

**Kosten:** Ablaufplan mittels EDF bestimmen

**Untere Schranke:** Vereinfachung bis EDF optimal ist!

- $\rightarrow$  Entfernen ungerichteter oder kernübergreifender Abhängigkeiten

**Verbesserung:** Durch gezieltes Hinzufügen von Abhängigkeiten



## Gliederung

- 1 Entwicklung – Herangehensweise
  - Ablaufplanung – Bottom-Up
  - Spezifikation – Top-Down
- 2 Manuelle Einplanung
  - Struktur zyklischer Ablaufpläne
- 3 Algorithmische Einplanung
  - List-Scheduling-Verfahren
  - Branch&Bound-Algorithmen
- 4 Moduswechsel
- 5 Zusammenfassung



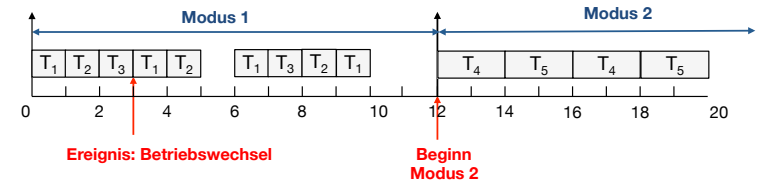
## Moduswechsel: Flexibilität in zeitgesteuerten Systemen

- ⚠ Eine für alle Lastsituationen passende statische Ablauftabellen
  - In der Praxis nur schwer zu realisieren
  - Negativbeispiel: Wartung von Steuergeräten im Auto
    - Diagnosedaten werden in normales Kommunikationsverhalten eingebettet
    - Niedrige *Nutzlast* (engl. *payload*) ist die Folge
- ⚠ Statische Ablauftabellen orientieren sich am **schlimmsten Fall**
  - Arbeitsaufträge belegen **immer** die zugewiesene WCET
  - Auch wenn sie zwar periodisch, aber nur selten ausgelöst werden
- **Entflechtung** der Arbeitsaufträge ist das Ziel
  - Arbeitsaufträge befinden sich nur in einer gemeinsamen Ablauftabelle, wenn sie auch zusammen ausgelöst werden können
- 🔗 Gruppierungen von Arbeitsaufträgen definieren **Betriebszustände**
  - Repräsentiert durch eine eigene statische Ablauftabelle
  - Wechsel des Betriebszustands impliziert auch ihren Wechsel
- ⚠ Betriebswechsel erfordert ein systemweit **koordiniertes Vorgehen**



## Rekonfiguration des Aufgabensystems

Änderung von Aufgabenanzahl und -parametern



- ⚠ Umstellen auf einen neuen statischen Ablaufplan bedeutet mehr als nur einen **Tabellenwechsel** zu vollziehen:
  - 1 Zerstörung und Erzeugung von periodischen Aufgaben
    - Einige periodische Aufgaben sind nicht mehr erforderlich  
~ **Betriebsmittelfreigabe**
    - Andere Aufgaben müssen dem System neu hinzugefügt werden  
~ **Betriebsmittelanforderung**
    - Manche Aufgaben überdauern den Betriebswechsel  
~ **Parametererhaltung**
  - 2 Einlagerung und Aktivierung der neuen Ablauftabelle
    - Taskparameter und neuer Ablaufplan wurden *à priori* bestimmt



## Arten von Betriebswechsel

Aperiodischer oder sporadischer Job

- 🔗 Wechsel vom speziellen Arbeitsauftrag (engl. *mode-change job*) durchführen lassen → **nichtperiodische Aufgabe**
  - Antwortzeit des Betriebswechsels minimieren (Hyperperiode!)
  - Verbunden mit einem weichen oder harten Termin
- **Aperiodisch** → Betriebswechsel mit weichem Termin
  - Mit höchster Dringlichkeit ausgeführt als aperiodischer Auftrag
    - Kommt vor allen anderen aperiodischen Aufträgen zum Zuge
  - **Zerstörung aperiodischer/sporadischer Jobs** ist problematisch
    - Ausführung aperiodischer Jobs wird hinausgezögert, bis der Betriebswechsel vollendet worden ist
    - Im Falle sporadischer Jobs stehen zwei Optionen zur Verfügung:
      - (a) Betriebswechsel wird unterbrochen und später fortgesetzt
      - (b) Übernahmepfung berücksichtigt den neuen Ablaufplan
  - Ziel ist Minimierung der Antwortzeit für den Betriebswechsel
- **Sporadisch** → Betriebswechsel mit hartem Termin
  - Anwendung muss die evtl. Abweisung des Wechsels behandeln
    - Betriebswechsel muss ggf. hinausgezögert werden



## Gliederung

- 1 Entwicklung – Herangehensweise
  - Ablaufplanung – Bottom-Up
  - Spezifikation – Top-Down
- 2 Manuelle Einplanung
  - Struktur zyklischer Ablaufpläne
- 3 Algorithmische Einplanung
  - List-Scheduling-Verfahren
  - Branch&Bound-Algorithmen
- 4 Moduswechsel
- 5 Zusammenfassung



## Resümee

- **Entwicklungsprozesse** führen verschiedenste Akteure zusammen
  - Firmen/Arbeitsgruppen sind u.U. über den ganzen Globus verstreut
  - Eine **zeitliche Spezifikation** der Abläufe ist wünschenswert
    - Sie ermöglicht die Entwicklung **top-down** zu strukturieren
    - Wird durch eine manuelle, statische Ablaufplanung unterstützt
- **Struktur zyklischer Ablaufpläne** ↪ gute Anordnung, Determinismus
  - Rahmen, Rahmenlänge, Scheiben; *major/minor cycle*
- **Algorithmische Einplanung** ordnet gerichtete, azyklische Graphen
  - Entlastung bei der Lösung eines komplexen Problems
  - List-Scheduling- und Branch&Bound-Algorithmen
- **Moduswechsel** durch aperiodischen oder sporadischen Auftrag
  - Tabellenwechsel, Betriebsmittelfreigabe/-anforderung, Nachladen



## Literaturverzeichnis

- [1] Abdelzaher, T. F. ; Shin, K. G.:  
Combined Task and Message Scheduling in Distributed Real-Time Systems.  
In: *IEEE Transactions on Parallel and Distributed Systems* 10 (1999), Nr. 11, S. 1179–1191.  
<http://dx.doi.org/10.1109/71.809575>. –  
DOI 10.1109/71.809575
- [2] Adam, T. L. ; Chandy, K. M. ; Dickson, J. R.:  
A comparison of list schedules for parallel processing systems.  
In: *Communications of the ACM* 17 (1974), Nr. 12, S. 685–690.  
<http://dx.doi.org/10.1145/361604.361619>. –  
DOI 10.1145/361604.361619. –  
ISSN 0001–0782
- [3] Baker, T. P. ; Shaw, A. C.:  
The Cyclic Executive Model and Ada.  
In: *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, IEEE Computer Society Press, 1988, S. 120–129
- [4] Casavant, T. L. ; Kuhl, J. G.:  
A taxonomy of scheduling in general-purpose distributed computing systems.  
In: *IEEE Transactions on Software Engineering* 14 (1988), Nr. 2, S. 141–154.  
<http://dx.doi.org/10.1109/32.4634>. –  
DOI 10.1109/32.4634. –  
ISSN 0098–5589



## Literaturverzeichnis (Forts.)

- [5] Kruatrachue, B. ; Lewis, T. G.:  
Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems / Oregon State University.  
Corvallis, OR, USA, 1976. –  
Forschungsbericht
- [6] Kwok, Y.-K. ; Ahmad, I. :  
Static scheduling algorithms for allocating directed task graphs to multiprocessors.  
In: *ACM Computing Surveys* 31 (1999), Nr. 4, S. 406–471.  
<http://dx.doi.org/10.1145/344588.344618>. –  
DOI 10.1145/344588.344618. –  
ISSN 0360–0300
- [7] Schild, K. ; Würtz, J. :  
Off-Line Scheduling of a Real-Time System.  
In: *Proceedings of the 13th ACM Symposium on Applied Computing (SAC '98)*.  
New York, NY, USA : ACM Press, 1998. –  
ISBN 0–89791–969–6, S. 29–38



## Literaturverzeichnis (Forts.)

- [8] Sih, G. C. ; Lee, E. A.:  
A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures.  
In: *IEEE Transactions on Parallel and Distributed Systems* 4 (1993), Nr. 2, S. 175–187.  
<http://dx.doi.org/10.1109/71.207593>. –  
DOI 10.1109/71.207593. –  
ISSN 1045–9219



# EZS – Cheat Sheet

## Typographische Konvention

Der erste Index gibt die Aufgabe an (z. B.  $D_i$ ), der Zweite (optional) bezieht sich auf den Arbeitsauftrag (z. B.  $d_{i,j}$ ). Exponenten zeigen verschiedene Varianten einer Eigenschaft an (z. B.  $T^{HW}$ ,  $T^{MED}$ ,  $T^{LD}$ ). Funktionen beschreiben zeitlich variierende Eigenschaften (z. B.  $P(t)$ ).

## Eigenschaften

$t$  (Real-)Zeit  
 $d$  Zeitverzögerung (engl. delay)

## Strukturelemente

$E_i$  Ereignis (engl. event)  
 $R_i$  Ergebnis (engl. result)  
 $T_i$  Aufgabe (engl. task)  
 $J_{i,j}$  Arbeitsauftrag (engl. job) der Aufgabe  $T_i$

## Temporale Eigenschaften

Allgemein

$r_i$  Auslösezeitpunkt (engl. release time)  
 $e_i$  Maximale Ausführungszeit (WCET)  
 $D_i$  Relativer Termin (engl. deadline)  
 $d_i$  Absoluter Termin  
 $\omega_i$  Antwortzeit (engl. response time)  
 $\sigma_i$  Schlupf (engl. slack)  
Periodische Aufgaben  
 $p_i$  Periode (engl. period)  
 $\phi_i$  Phase (engl. phase)

## Aufgaben – Tupel

$T_p = (p, e, D, \phi)$  Periodische Aufgabe ohne Priorität (zeitgesteuert oder dynamische Taskpriorität),  $D = p$  und  $\phi = 0$  sind der Reihe nach optional

$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$  Arbeitsauftrag

## Ablaufplanung

$P_i$  Priorität (engl. priority) der Aufgabe  
 $T_i$   
 $\Omega_i$  Prioritätsebenen (engl. number of priorities)  
 $h_{\Delta_i}$  Rechenzeitbedarf (engl. demand)  
 $u_{\Delta_i}$  CPU-Auslastung (engl. utilisation)  
 $U$  Absolute CPU-Auslastung  
 $H$  Hyperperiode (großer Durchlauf, engl. major cycle)  
 $f$  Rahmenlänge (kleiner Durchlauf, engl. minor cycle)  
 $e'_i$  WCET aller Aufträge im Rahmen  $i$

