8 Echtzeitplanung (real-time scheduling)

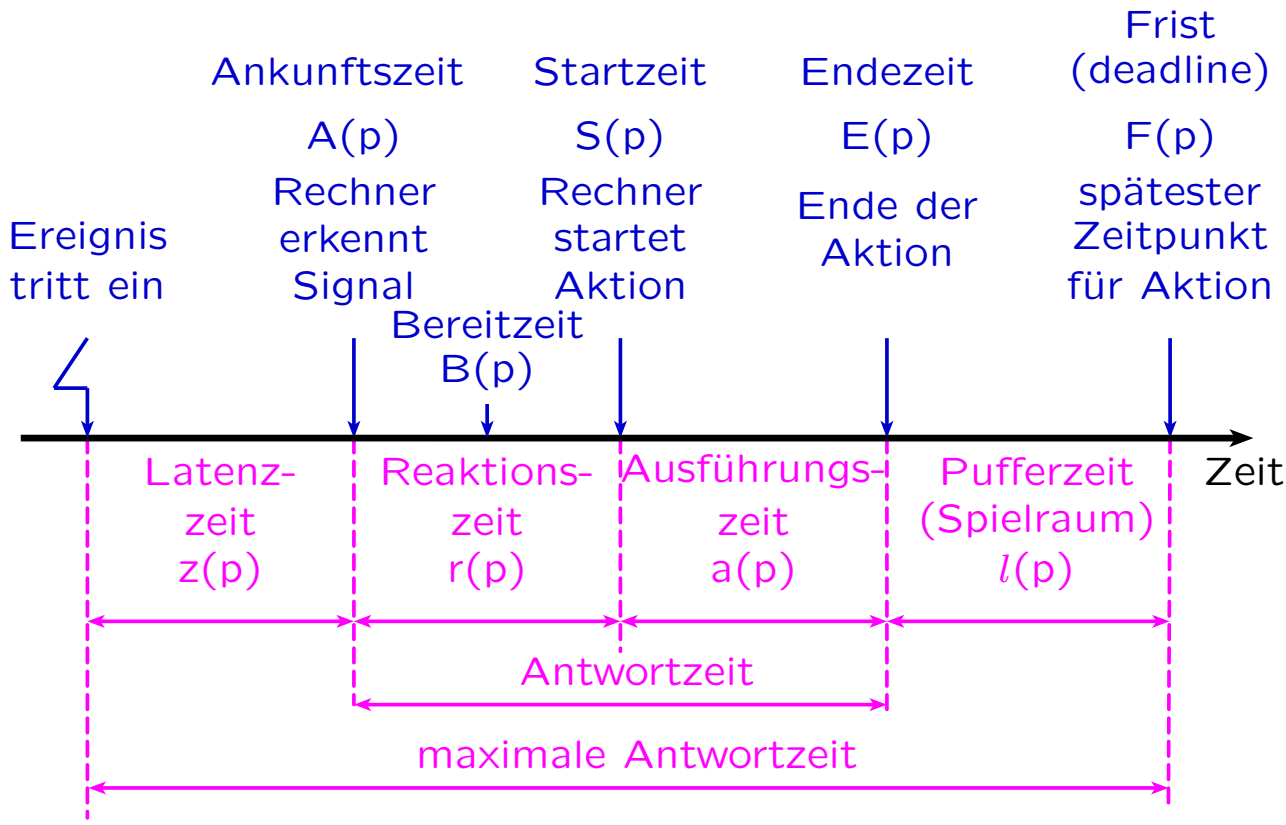
8.1 Grundbegriffe

Die Ablaufgeschwindigkeit der physikalischen Vorgänge im technischen Prozeß bestimmt notwendige Reaktionszeiten des Rechners und legt Antwortzeiten fest.

Es treten perodische und spontane Anforderungen (Zeitpunkte, Signale, Alarme), harte und weiche Zeitbedingungen auf.

Planung für Prozessoren und Betriebsmittel

- statische/dynamische Planung
- explizit/implizite Planung
- offline/online Planung



Zeitablauf für ein Ereignis im Prozeß p

- Anmerkungen
  - ⋆ Großbuchstaben: Zeitpunkte
  - ⋆ Kleinbuchstaben: Zeitdauern
  - ⋆ Reaktionszeit und Ausführungszeit abhängig von der Umgebung, z.B. von anderen wichtigerern Prozessen
  - ⋆ Latenzzeit:
    u.a. Verzögerungen durch lokale Hardware, Übertragung über Netze
  - ⋆ Bereitzeit:
    Der Prozeß darf vor seiner Bereitzeit nicht gestartet werden, da z.B. dann erst benötigte Daten verfügbar; in der Regel $B(p) = A(p)$

- Forderungen an Echtzeitplanung
  - ⋆ a priori zeigen, daß stets Zeitrandbedingungen erfüllt, d.h. $\forall p : E(p) \leq F(p)$
  - ⋆ Einplanung nicht nur CPU, sondern auch andere Betriebsmittel (in Vorlesung nur CPU berücksichtigt)
  - ⋆ Zeitbedingungen einhalten bzgl. parallelen Vorgängen (Rechenprozessen)
  - ⋆ Berücksichtigung von Kausalzusammenhängen (Synchronisation, Vorranggraphen, Präzedenzsysteme)
  - ⋆ Einhalten von Zeitbedingungen bei der Verteilung von Information (Kommunikation, gemeinsames Führungsziel)
  - ⋆ präemptive Prozessorzuteilung
- Nicht trivial, da technische Systeme vielfach komplex und schwer überschaubar

8.2 Dimensionierung der Rechnerleistung

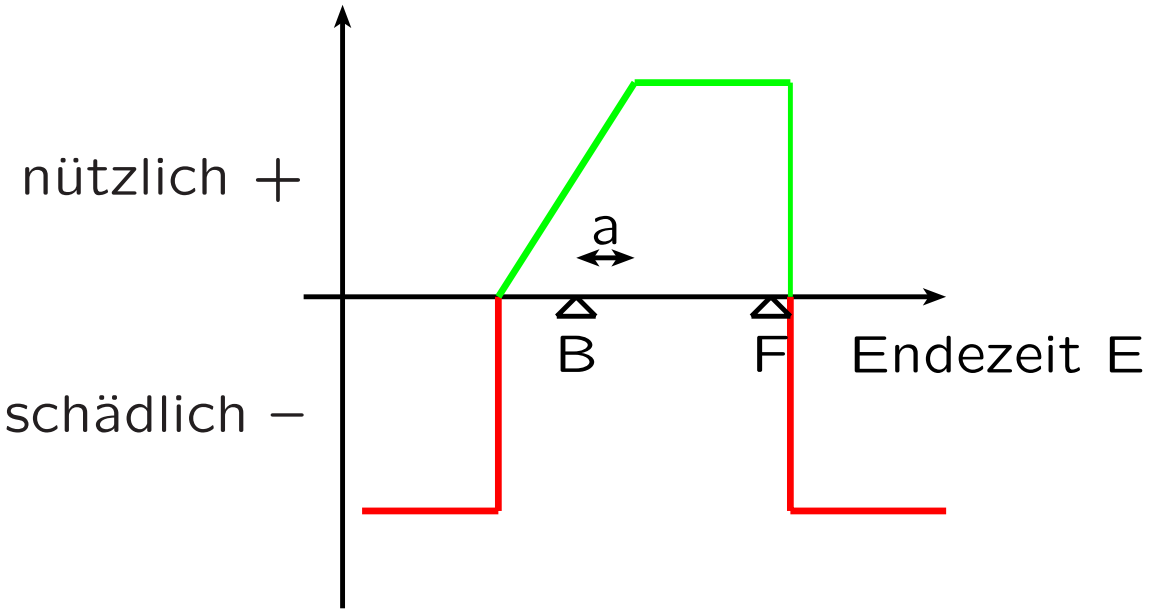
- Aufstellen "worst case"–Analyse
  - ⋆ Bedarf aus bekannten periodischen Anforderungen
  - ⋆ Bedarf aus erwarteten spontanen Anforderungen
  - ⋆ Zuschlag 100% oder mehr zum Auffangen von Lastspitzen.
- Unterschied zu konventionellen Betriebszielen
  - ⋆ keine maximale Auslastung des Prozessors,
  - ⋆ keine Durchsatzoptimierung,
  - ⋆ Abläufe sollten determiniert abschätzbar sein.

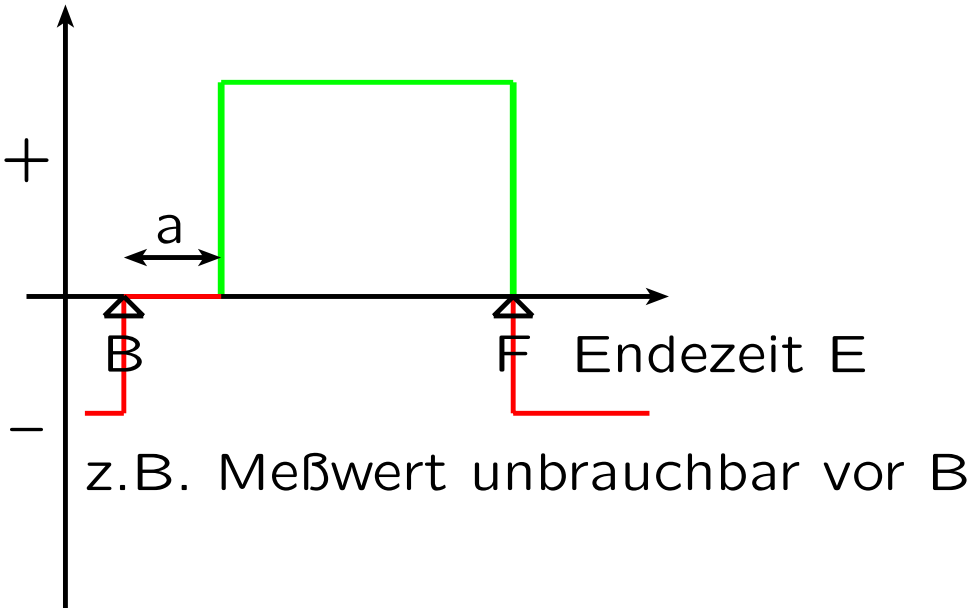
8.3 Schadenstaxonomie

- Programmablaufsteuerung muß Zeitbedingungen (Antwortzeiten) des technischen Prozesses berücksichtigen, damit kein Schaden entsteht.
- Harte Zeitbedingungen
  - ⋆ msec, sec, aber nicht min oder Std.
  - ⋆ Übertretung verursacht materiellen, wirtschaftlichen und/oder Personenschaden
  - ⋆ Beispiele:
    - falsches Mischungsverhältnis ergibt Fehlproduktion
    - falsche Steuerung führt zu Maschinenschaden
    - energiesparende, jedoch instabile Fluglagen gefährden Flug
    - Landung Space–Shuttle
    - Roboter im Gefahrenbereich

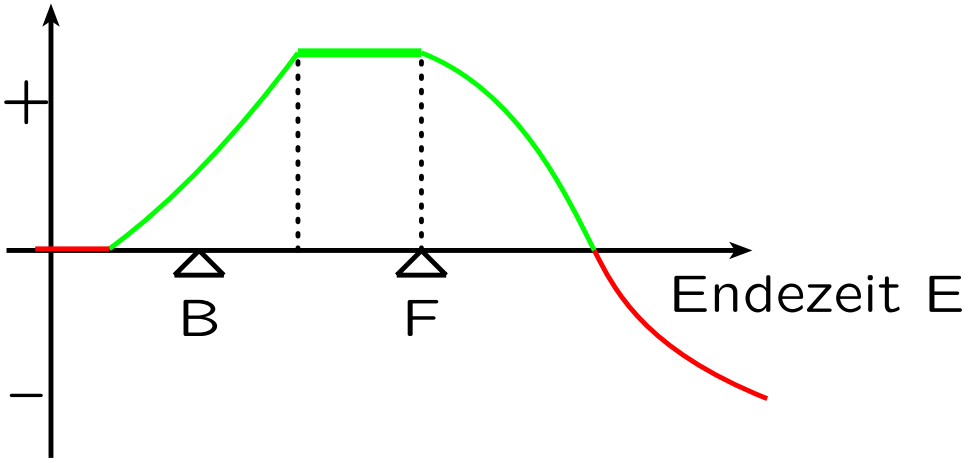
- Weiche Zeitbedingungen
  - ⋆ Übertretung führt zu Störungen
  - ⋆ das Führungsziel wird nicht erreicht
  - ⋆ Verlust der Optimalität, Schaden tolerierbar
  - ⋆ Beispiele:
    - ◦ Verspätete Produktfertigstellung
    - ◦ Höherer Energieverbrauch
    - ◦ tolerierbare Fehlfarben
    - ◦ tolerierbare Maßabweichungen
- Abstufung des Schadens nach Nutzenfunktionen

(1) Beispiele für harte Randbedingungen



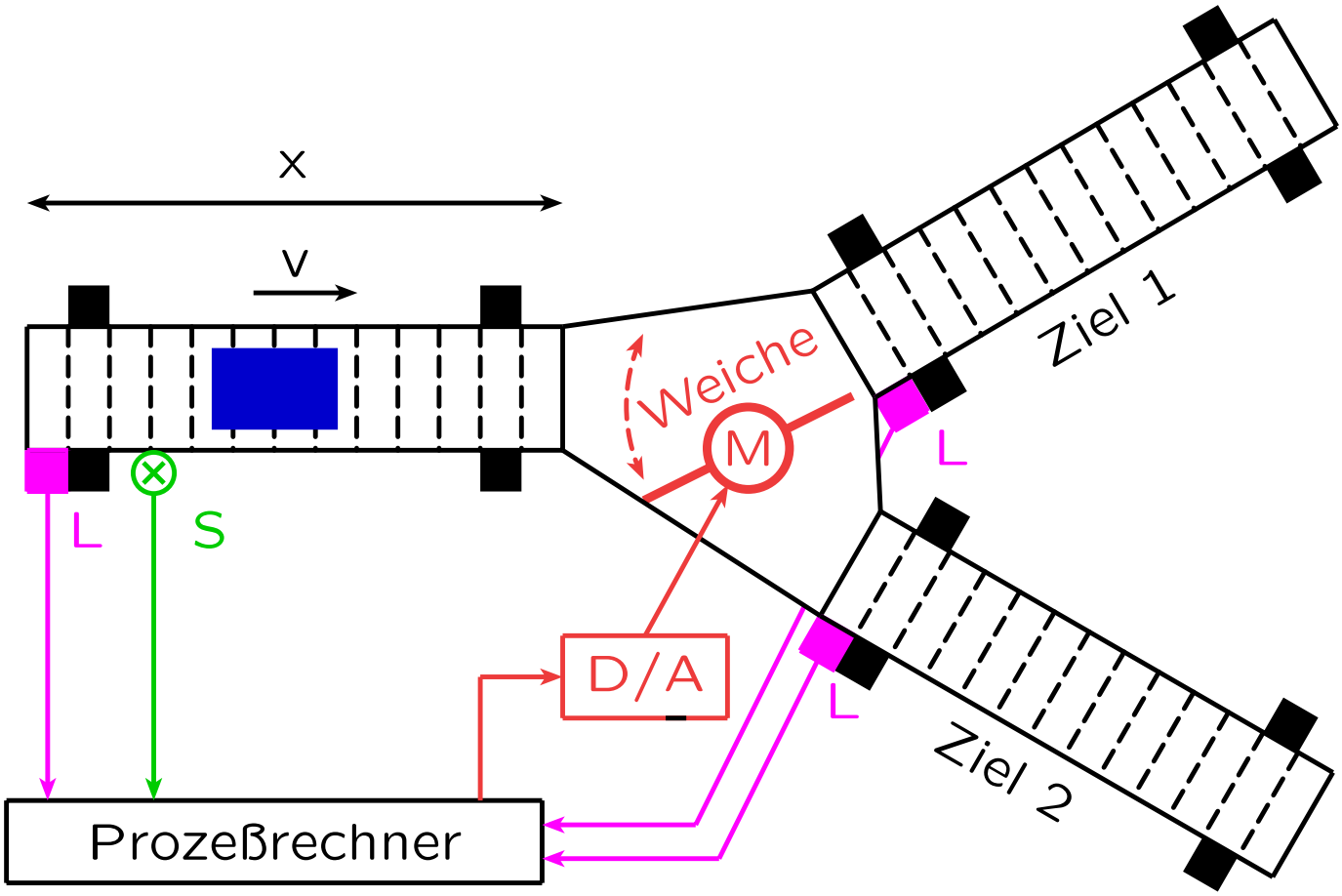


(2) Beispiel für weiche Randbedingungen



8.4 Echtzeitanforderung am Beispiel Paketverteilungsanlage

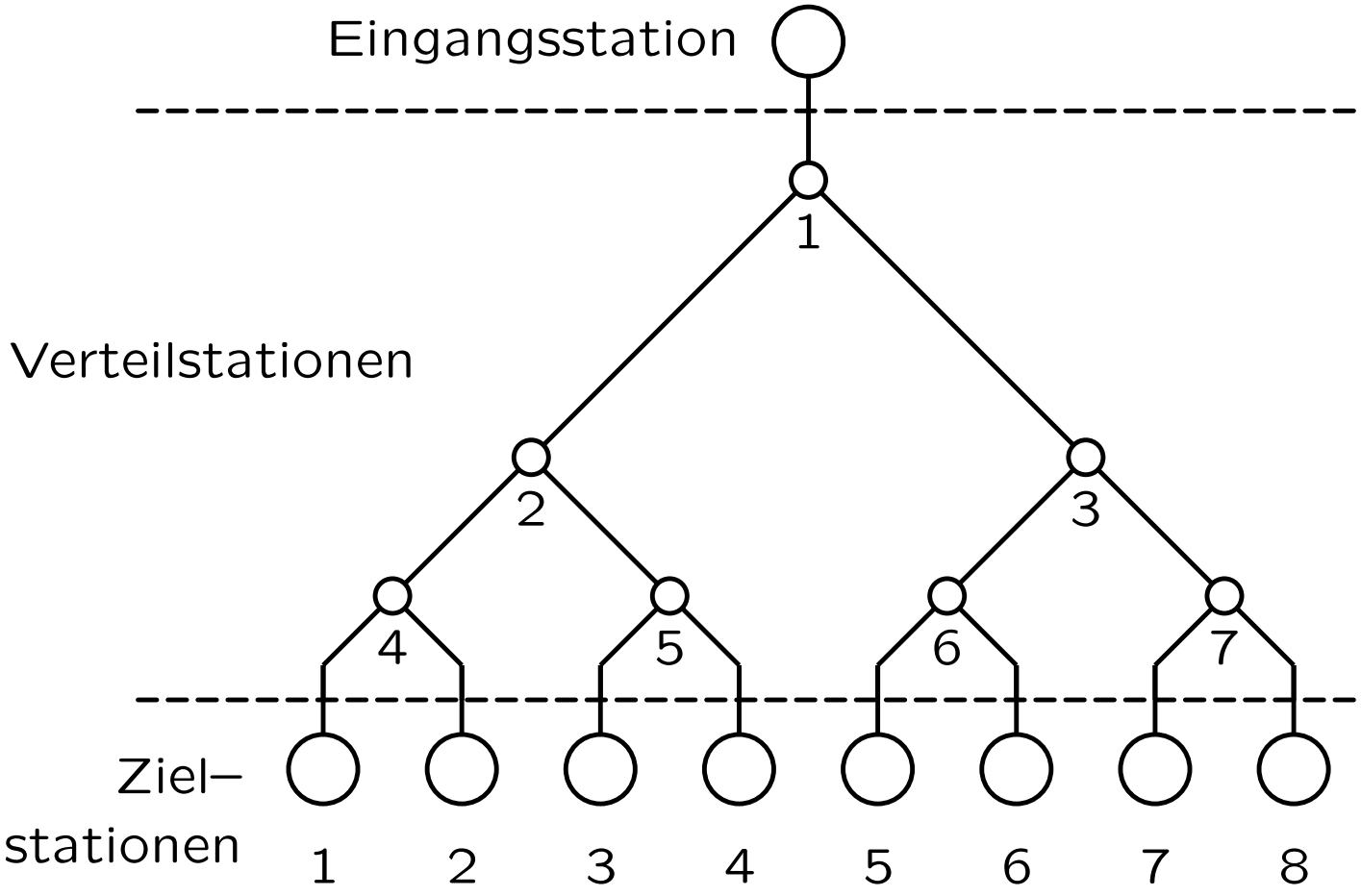
- Verteilstation



L	Lichtschranke
M	Motor für Weiche
S	Strichcode-Leser
v	Geschwindigkeit des Förderbands
x	Abstand Ankunftssignalgeber–Weiche
t_W	Zeitdauer zum Umstellen der Weiche

- Es treten asynchrone Ereignisse auf
- Ablauf
  - ⋆ Rechner erhält Signal beim Eintreffen
  - ⋆ Rechner liest und verarbeitet Strichcode
  - ⋆ Rechner berechnet Weg (Ziel 1 oder Ziel 2)
  - ⋆ Rechner gibt Signal an Weichenmotor (nur falls neuer Weg!)
  - ⋆ Rechner reagiert nur beim Eintreffen von Signalen (ereignisgesteuert)
- von der Lichtschranke bis zur Weiche benötigt Paket die Zeit x/v
- Rechner muß spätestens t_W Zeiteinheiten bevor Paket an Weiche ist dem Weichenmotor das Stellsignal senden
- damit maximale Antwortzeit $\triangle t_{max} = x/v - t_W$

- Schema Gesamtanlage (Version 1)

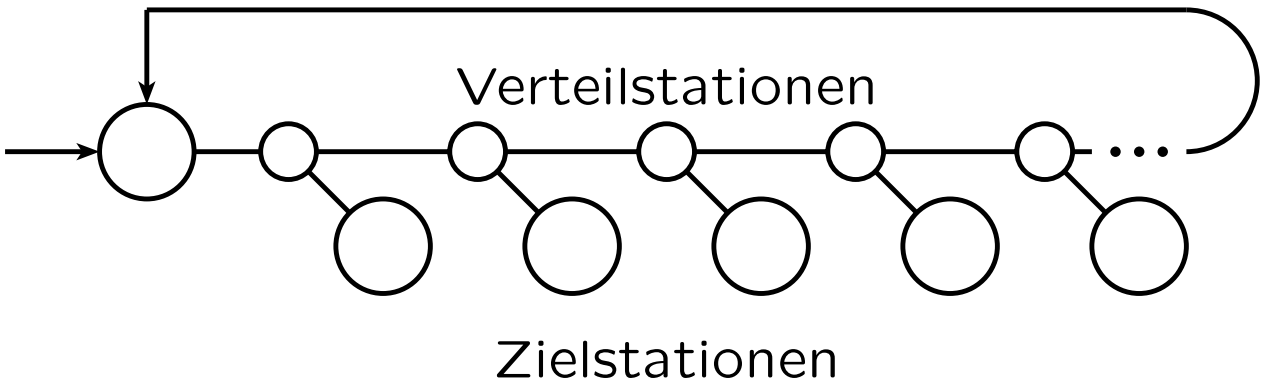


  - ⋆ 8 Zielstationen
  - ⋆ 7 Verteilstationen
  - ⋆ mehrere Pakete gleichzeitig
  - ⋆ Strichcode-Leser nur an Eingangsstation
  - ⋆ Rechner führt Buch, wo welches Paket aufgrund Signale der Lichtschranken und veranlaßt Weichenumstellungen (fehleranfällig, erschwerter Wiederanlauf)

⋆ Signalein–/ausgänge:

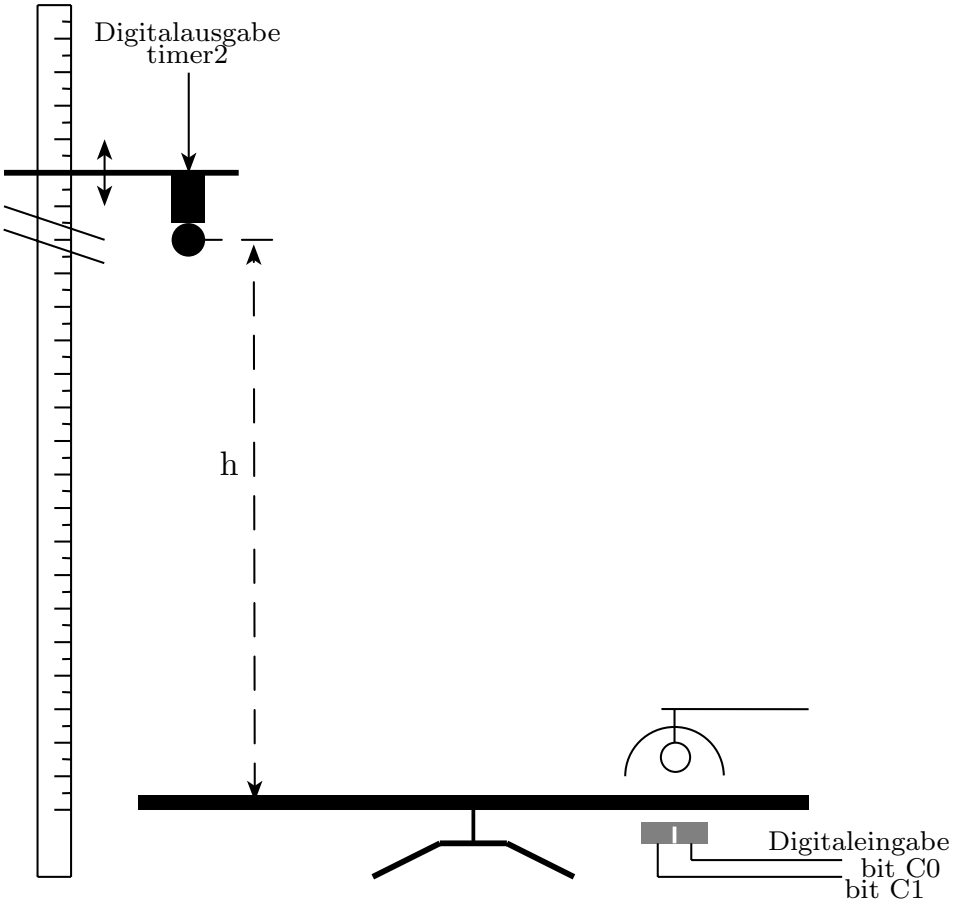
1	Leser Strichcode bei Eingangsstation
15	Lichtschranken
7	Signale für Motor nach links
7	Signale für Motor nach rechts
14	Kontakte für Ist-Weichenstellungen (links/rechts)
44	Signale

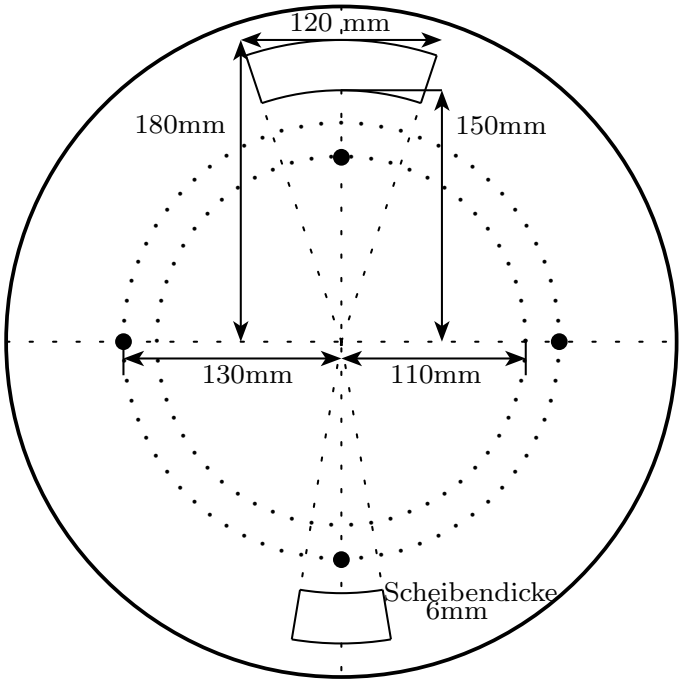
- Schema Gesamtanlage (Version 2)



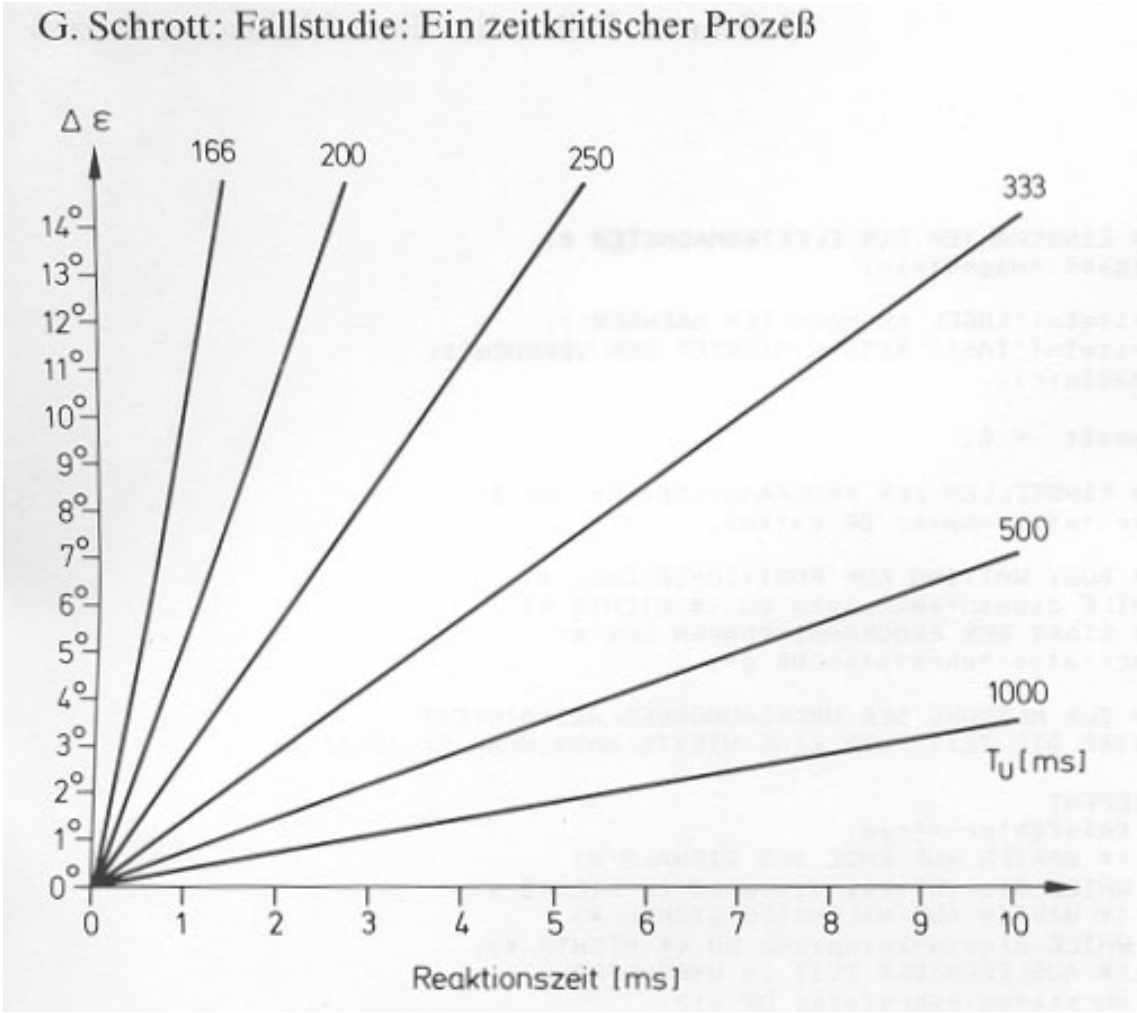
- Exkurs zur Anlagenplanung
  - ⋆ Welche Konfiguration ist besser?
  - ⋆ Kriterien?
  - ⋆ Alternativen mit Einfluß örtlicher Verteilung?
- Exkurs zur Ausfallsicherheit
  - ⋆ Bei Ausfallsicherheit mindestens doppelt so viele Signale → 88 Signale
  - ⋆ Check vor Zielstation, ggf. zurück zu Eingangsstation
  - ⋆ Wiederanlauf nach Fehlern (überall Lesestationen?, alles zurück?)
  - ⋆ Erkennen Stau
  - ⋆ alternative Wege bei Störungen

- Kugelfallversuch
  einfaches Beispiel mit harten Zeitanforderungen


Digitalausgabe
timer2
h
Digitaleingabe
bit C0
bit C1


120 mm
180mm
150mm
130mm
110mm
Scheibendicke
6mm

G. Schrott: Fallstudie: Ein zeitkritischer Prozeß



8.5 Weitere Bezeichnungen

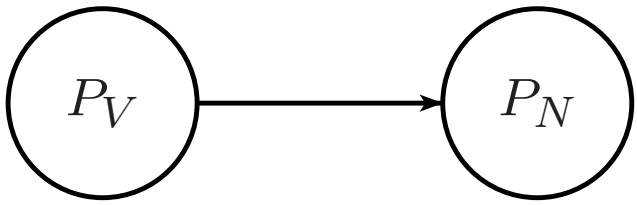
- Algorithmen zur korrekten Zuteilung von Rechenleistung bei mehreren zu bearbeitenden Aktionen (Rechenprozessen) $P_1, P_2, \ldots, P_n$ müssen für jeden Prozeß P_i beinhalten:
  - $F(p_i)$
    spätester Zeitpunkt (Frist) für Antwort an technischen Prozeß
  - $B(p_i)$
    Bereitzeitpunkt (frühester Startzeitpunkt)
  - $a(pi)$
    Gesamtausführungszeit
- Weitere wichtige Größen, die von der aktuellen Zeit t abhängig sind:
  - $a(p_i, t)$
    Restausführungszeit des Prozesses P_i zur Zeit t; wobei
    $a(p_i, B(p_i)) = a(p_i)$
    $a(p_i, F(p_i)) = 0$, falls Prozeß zeitgerecht beendet

- ⋆ $\triangle t(p_i, t)$
  Zeitspanne bis zur Frist des Prozesses P_i; wobei
  $\triangle t(p_i, t) = F(p_i) - t$
- ⋆ $l(p_i, t)$
  Zur Zeit t noch verfügbarer Spielraum für Prozeß P_i:
  $l(p_i, t) = \triangle t(p_i, t) - a(p_i, t)$
  falls:
  $l(p_i, t) < 0 \rightarrow P_i$ kann nicht rechtzeitig fertiggestellt werden
  Echtzeitforderung verletzt!

- Anmerkungen
  - ⋆ Laufzeit $a(pi)$ von Programmen bei Echtzeitumgebung bekannt, da Aufgaben und Aufgabenprofile festgelegt (Analyse des Codes, Messungen)
  - ⋆ ggf. Präzedenzen der Aktionen P_i beachten (z.B. P_i muß vor P_j ausgeführt werden)

8.6 Präzedenzsysteme

- Folgen von untereinander abhängigen Aktionen (z.B. "pipes")
- Bereitzeiten, Antwortzeiten oder Fristen ergeben sich evtl. erst aus vorherigem Ablauf
- Beispiel für Graph-Darstellung



Vorgänger Nachfolger

  - ⋆ Es gelte:

$$\begin{aligned} a(p_V) &= 1 & \qquad a(p_N) &= 3 \\ F(p_V) &= 3 & \qquad F(p_N) &= 5 \\ B(p_V) &= 0 & \qquad B(p_N) &= 0 \\ \rightarrow \triangle t(p_V, 0) &= 3 & \qquad \rightarrow \triangle t(p_N, 0) &= 5 \end{aligned}$$

  - ⋆ Falls von P_V Frist $F(p_V)$ gerade noch eingehalten wird, kann der Nachfolgeprozeß nicht rechtzeitig fertig werden

- Bei der Planung mit Präzedenzsystemen muß also berücksichtigt werden, daß auch Folgeprozesse noch rechtzeitig fertig werden

- Verfahren hierzu:
  Es wird ein normalisiertes Präzedenzsystem PS' anstelle des ursprünglichen Präzedenzsystems PS eingeführt
- Normalisierung (Durchführung zur Zeit t)
  - ⋆ $a'(p_i) = a(p_i)$ und $a'(p_i, t) = a(p_i, t)$
  - ⋆ $F'(p_i, t) =$

$$\begin{cases} F(p_i) & \text{falls } \mathcal{N}(p_i) = \emptyset \\ min\left(F(p_i),\ \underset{q \in \mathcal{N}(p_i)}{min}\ (F'(q) - a'(q,t))\right) & \text{sonst} \end{cases}$$

  - ⋆ wobei $\mathcal{N}(p_i)$ die Menge der unmittelbaren Nachfolger von p_i
  - ⋆ rekursive Berechnung, beginnend mit Prozessen ohne Nachfolger im Präzedenzgraphen
  - ⋆ Anmerkungen zu $B'(p_i)$:
    - ○ hängen die Bereitzeiten von externen Vorgängen ab, die unabhängig vom Scheduling der Prozesse sind, dann gilt $B'(p_i) = B(p_i)$

- hängen die Bereitzeiten von der Beendigung vorheriger Prozesse ab, dann ist $B'(p_i)$ abhängig vom konkreten Scheduling.

⋆ ähnlich wie in der Netzplantechnik lassen sich früheste und späteste Start-Termine für die Prozesse angeben. Diese sind aber für das Scheduling ohne Bedeutung, da ja die konkreten Präzedenzen berücksichtigt werden müssen.

- Präzedenzsystem nur dann planbar, falls zugehöriges normalisiertes PS' planbar, d.h. es dafür einen zulässigen Plan gibt

8.7 Pläne

- Phasen der Planung:

  (1) Test auf Einplanbarkeit (feasibility check)

  (2) Planberechnung (schedule construction)

  (3) Umsetzung auf Zuteilung im Betriebssystem (dispatching)

- Gesucht:
  Plan mit aktueller Start– und Endezeit für jede Aktion P_i

- Darstellung z.B. als nach der Zeit geordnete Liste von Tupeln (P_i, S(P_i), E(P_i), BM); wobei BM die Menge der benötigten Btriebsmittel (kann entfallen, wenn immer nur eines geplant wird)

- Falls Prozeß unterbrochen wird: mehrere Tupel je P_i mit Anfangs- und Endzeitpunkten von aktiven Intervallen

- Zulässiger Plan
  Ein Plan, bei dem alle Prozesse einer Prozeßmenge eingeplant werden und dabei keine Präzedenzrestriktionen und keine Zeitanforderungen verletzt werden

- Optimales Planungsverfahren
  Verfahren ist optimal, wenn es für jede Prozeßmenge unter den gegebenen Randbedingungen einen zulässigen Plan findet, falls ein solcher existiert
- statische Planung
  - ⋆ erfordert Kenntnis aller Prozesse P_i und von $B(p_i)$, $a(p_i)$ und $F(p_i)$
  - ⋆ abgeschlossenes System
  - ⋆ exakte Algorithmen gehören in diese Gruppe
- dynamische Planung
  - ⋆ unvorhergesehen eintreffende Aktionen (Prozesse)
  - ⋆ unbekannte Prozeßparameter
  - ⋆ keine Gesamtoptimierung
- Grundstrategien zur Prozeßbearbeitung
  - ⋆ Präemptives (unterbrechbares) Abarbeiten
    - ◦ Aktionen (Prozesse) haben Prioritäten
    - ◦ Prioritäten statisch oder dynamisch berechnet

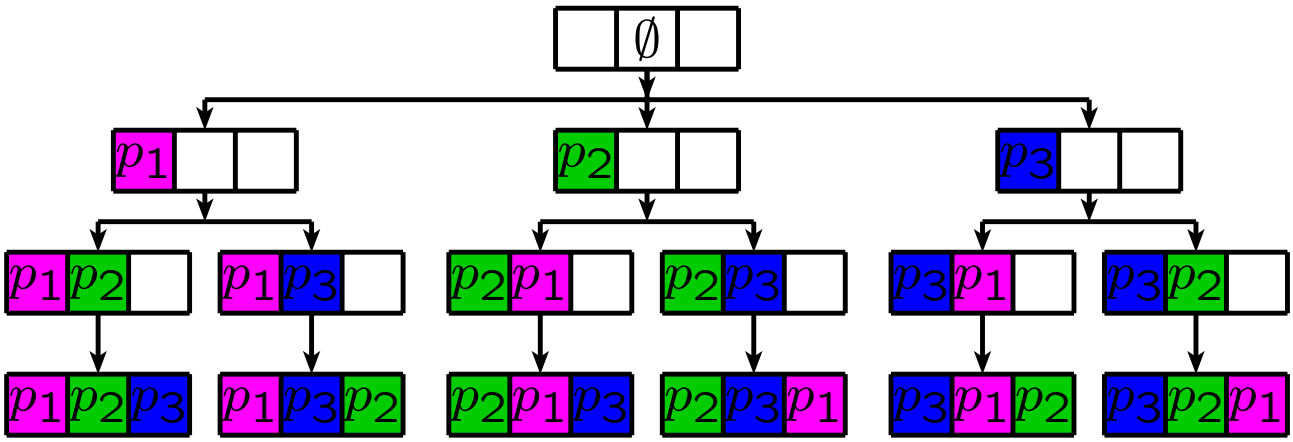
- Ausführung einer Aktion wird sofort unterbrochen, sobald Aktion mit höherer Priorität eintrifft
- unterbrochene Aktion wird an der Unterbrechungsstelle fortgesetzt, sobald keine Aktion höherer Priorität ansteht
- typisch für Echtzeitaufgaben (mit Ausnahme von Programmteilen, die zur Sicherung der Datenkonsistenz nicht unterbrochen werden dürfen)
- Nachteil: häufiges Umschalten reduziert Leistung

⋆ Nichtpräemptives (ununterbrechbares) Abarbeiten

- Eine begonnene Aktion wird beendet, selbst wenn während der Ausführung Aktionen höherer Dringlichkeit eintreffen
- z.B. Schreiben in Druckerpuffer, Schließen eines Ventils
- Nachteil: evtl. Versagen (zu hohe Reaktionszeit) des Systems beim Eintreffen "unvorhergesehener" Anforderungen

8.8 Planen durch Suchen

- Zunächst ununterbrechbare Aktionen vorausgesetzt
- exakte Planung über Durchsuchen des Lösungsraums
- Beispiel:
  - ⋆ n=3 Prozesse P_1, P_2, P_3 und 1 CPU
  - ⋆ Suchbaum:



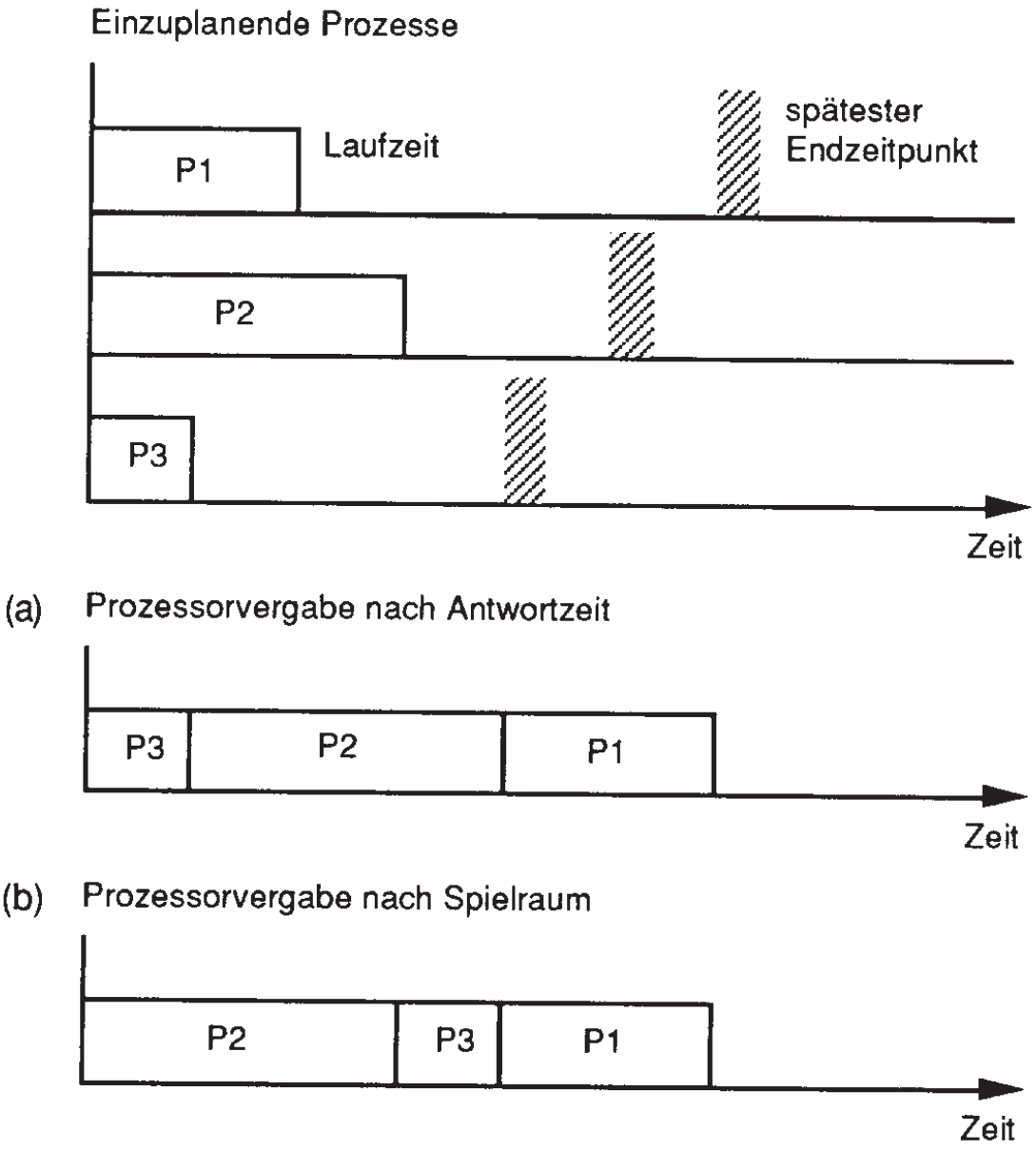
- n! (hier 6) Permutationen zu bewerten
- Bei n zu planenden Prozessen und einem Mehrfachbetriebsmittel aus m Einheiten hat ein Knoten $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ Unterknoten

- wegen Komplexität normalerweise nicht einsetzbar
- leichte Verbesserung der Komplexität durch
  - ⋆ Abbrechen der Bildung einer Sequenz, wenn bei einer der Aktionen eine Zeitüberschreitung
  - ⋆ Heuristiken, z.B. vorsortieren nach $B(p_i)$
- viele Algorithmen z.B. im Buch: J.Blazewicz,et al.: Scheduling in Computer and Manufacturing Systems. 2.Auflage, Springer 1994
- Bei unterbrechbaren Aktionen führt Granulariät der Zeitintervalle mit Präzedenzen der Aktionen zu System mit ununterbrechbaren Aktionen
- Weitere Probleme treten durch Synchronisation der Aktionen und die Konkurrenz um weitere Betriebsmittel auf (siehe z.B. J.Xu, D.L.Parnas: Scheduling Processes with release times, deadlines, precedence and exclusion relations. IEEE Trans. on SW Eng., Vol.16, 1990, S.360).

8.9 Zeitplanung bei Einprozessor–Systemen

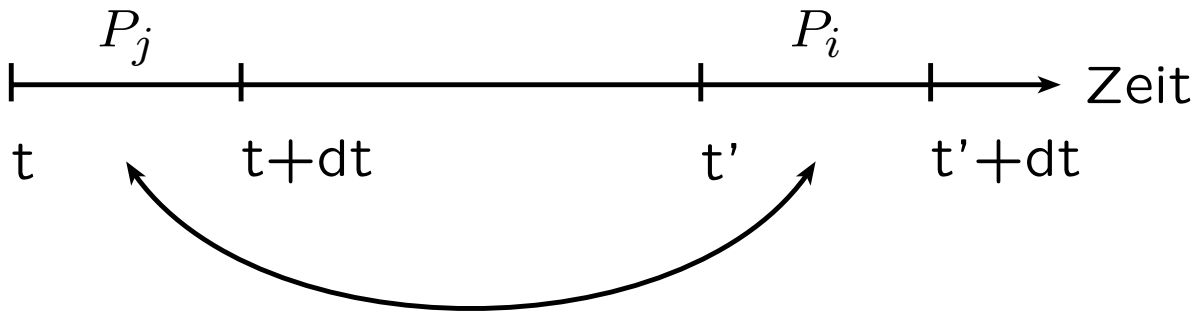
- Einziges verplantes Betriebsmittel ist 1 CPU
- Kriterien zur Prüfung auf Einhaltbarkeit aller Zeitbedingungen sind relativ einfach
  - ⋆ n Prozesse
  - ⋆ geordnet nach Fristen:
    $F(p_i) < F(p_{i+1}) \quad (i = 1 \ldots, n-1)$
  - ⋆ gleiche Bereitzeiten $B(p_i) = 0$
  - ⋆ es muß notwendig gelten:
    $\forall i : \triangle t(p_i, t) \geq \quad \sum_{k=1}^{i} \quad a(p_k, t) \quad (i = 1, \ldots, n)$
- gesucht: Strategien zur dynamisch korrekten Ordnung der Prozesse
- S1: Einplanung nach Fristen
  next deadline scheduling
  der Prozeß mit der nächsten Frist F erhält als nächster die CPU
- S2: Einplanung nach kleinstem Spielraum
  least slack time scheduling
  least laxity scheduling
  der noch nicht beendete Prozeß p mit dem momentan kleinsten Spielraum l(p, t) erhält als nächster die CPU

- ⋆ $l(p_j, t) = F(p_j) - t - a(p_j, t)$
- ⋆ solange p_i bearbeitet wird, ist $l(p_i, t)$ konstant; für alle wartenden p_j nimmt $l(p_i, t)$ linear in t ab.
- ⋆ erkennt früher als S1, wenn Zeitbedingungen nicht eingehalten werden können



- Voraussetzung: Einprozessorsystem, gleiche Bereitzeiten,
  Satz: Die Zuteilungsstrategien S1 und S2 sind optimal
- Beweis für Optimalität von S1
  - ⋆ Beweisidee: Tausch in existierendem Plan
  - ⋆ zulässiger Plan PlanZ sei bekannt
  - ⋆ PlanF sei Plan nach Fristen
  - ⋆ Prozesse seien nach Fristen geordnet:
    $F(p_i) \leq F(p_{i+1})$
  - ⋆ pz(Plan, t) liefert den Index des Prozesses, der im genannten Plan zur Zeit t aktiv ist
  - ⋆ PlanZ wird schrittweise in PlanF überführt
  - ⋆ PlanZ(t) ist der bis zur Zeit t modifizierte Plan
  - ⋆ PlanZ(0)=PlanZ, d.h. Ausgangssituation
  - ⋆ Annahme: PlanZ(t) und PlanF seien bis zum Zeitpunkt t identisch
  - ⋆ zum Zeitpunkt t gilt:
    i = pz (PlanF, t)
    j = pz (PlanZ(t), t)
    i ≠ j (nach Voraussetzung)
  - ⋆ betrachtet Zeitintervall dt

- ⋆ Fall $i < j$:
  - ◦ $F(p_i) \leq F(p_j)$
    wegen Ordnung d. Prozesse
  - ◦ $t + dt \leq F(p_j)$
    da PlanZ zulässiger Plan
  - ◦ da p_i in PlanF im Intervall (t, t+dt) rechnet und die Pläne bis zur Zeit t identisch sind, kann p_i auch in PlanZ noch nicht beendet sein, also gilt:
    $\exists t' \geq t + dt : i = pz(PlanZ(t), t') =$
    $pz(PlanZ(t), t' + dt)$
    $t' + dt \leq F(p_i) \leq F(p_j)$
  - ◦ Übergang von Planz(t) zu PlanZ(t+dt) erfolgt durch zeitliches Tauschen der Aktivitätsphase von p_i mit der von p_j



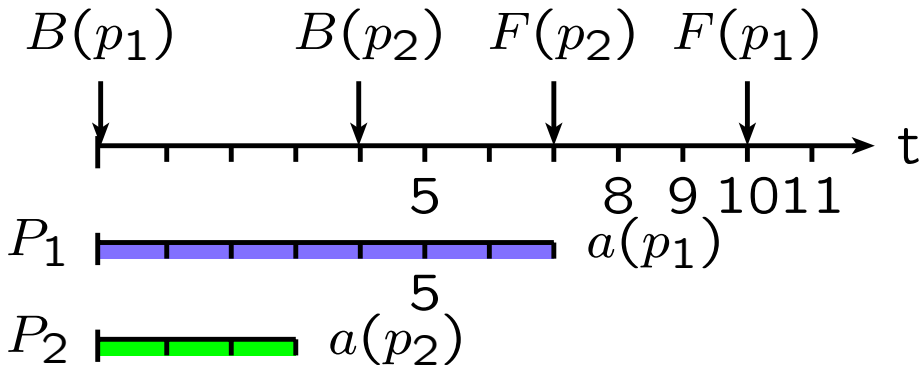
  - ◦ Zeitbedingungen sind dadurch nicht verletzt

- ⋆ Fall $i > j$:
  - ◦ rechnet in PlanF zur Zeit t der Prozeß p_i, dann sind alle Prozesse p_j mit $j < i$ bereits beendet
  - ◦ PlanF und PlanZ(t) stimmen bis zur Zeit t überein
  - ◦ der Fall $i > j$ kann also gar nicht auftreten
- ⋆ Damit ist gezeigt: Jeder zulässige Plan kann ohne Verletzung der Fristen in den durch die Fristenplanung erzeugten Plan überführt werden
- ⋆ Fristenplanung ist also bei Einprozessorsystemen optimal, egal ob präemptiv oder nichtpräemptiv

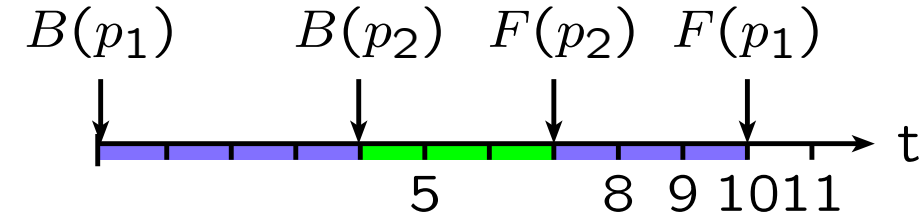
• Beweis für Optimalität von S2
Beweisidee wie für S1

- Beispiel, daß $B(p_i) = 0$ notwendig
  S1 und S2 liefern unzulässige Pläne, falls dies nicht der Fall (vgl. folgende Seite)
- allerdings Modifikation möglich, so daß korrekte Pläne auch in diesem Fall:
  - ⋆ präemptive Strategie
  - ⋆ Neuplanung bei jeder Bereitzeit
  - ⋆ Einplanung nur derjenigen Prozesse, deren Bereitzeit erreicht ist
  - ⋆ entspricht Neuplanung, wenn ein Prozeß aktiv wird
  - ⋆ ev. Zeitscheiben für Prozesse mit gleichem Spielraum

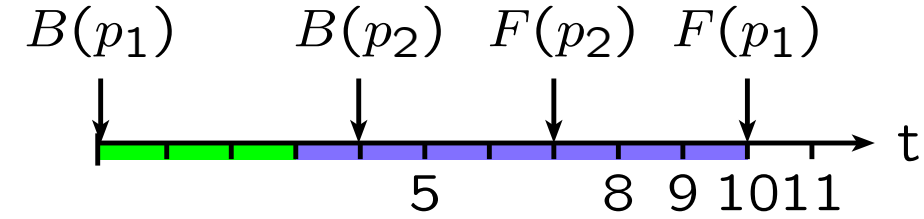
Lastprofil:



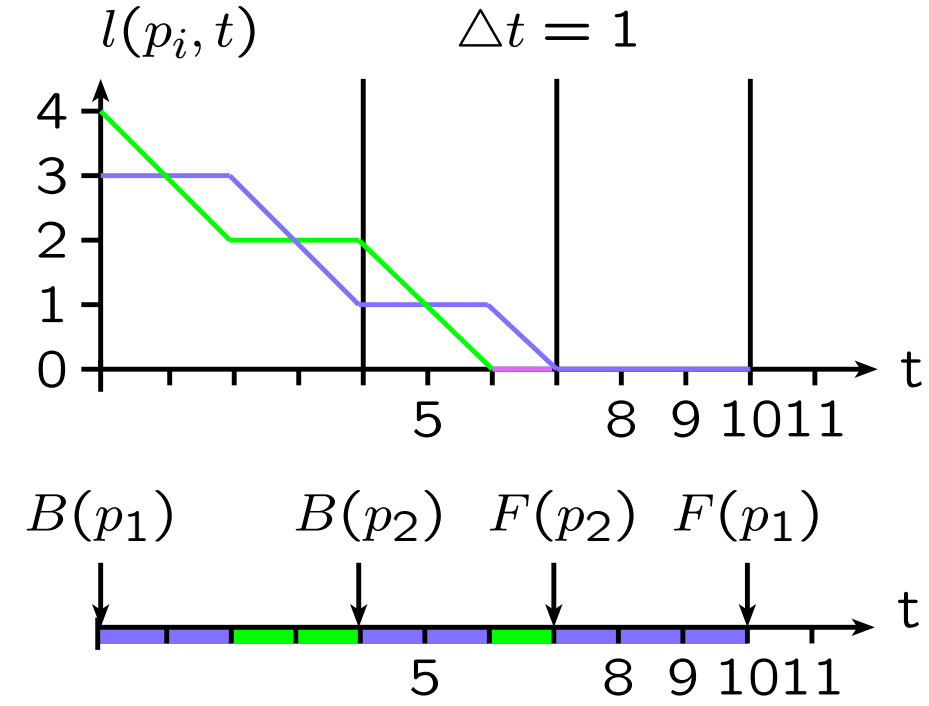
korrekter Plan:



Fristenplanung:



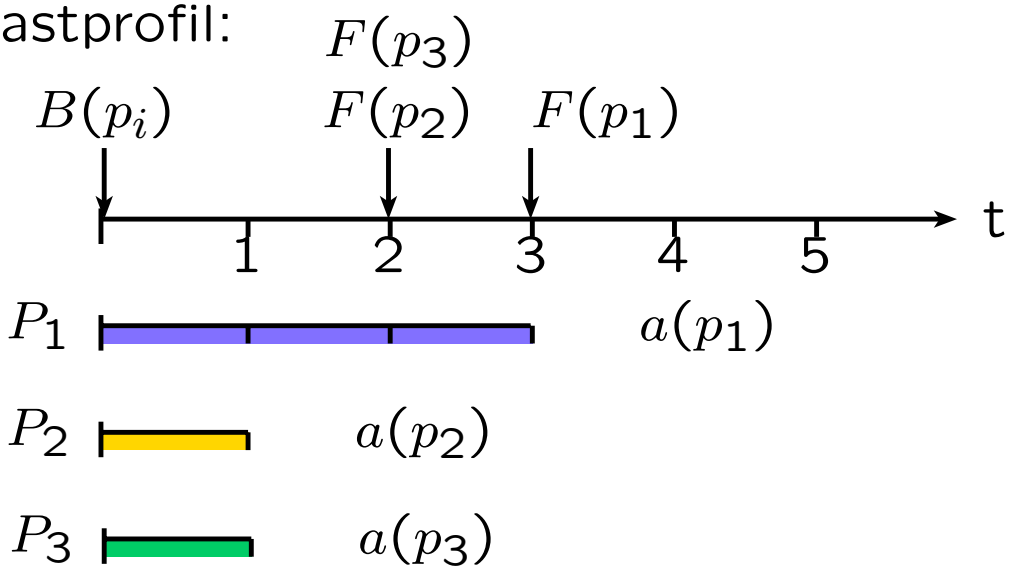
kleinster Spielraum:



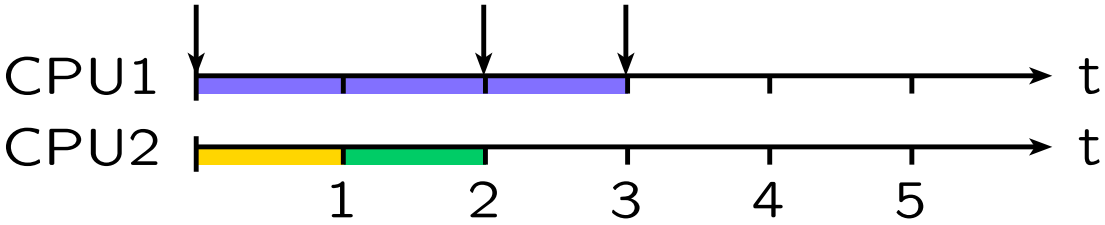
8.10 Zeitplanung bei Mehrprozessor–Systemen

- gehört zur Klasse der Zuteilungsverfahren für homogene, austauschbare Betriebsmittel
- S1 nicht optimal, egal ob präemptiv oder nichtpräemptive Strategie
- S2 geht nur, falls alle Bereitzeitpunkte $B(p_i)$ gleich
- korrekte Zuteilungsalgorithmen erfordern das Abarbeiten von Suchbäumen mit NP–Aufwand oder geeignete Heuristiken
- Buch: J. Blazewicz, et al.
- Gegenbeispiel zu S1 ist optimal
  - ⋆ präemptiv
  - ⋆ 2 Prozessoren CPU_1 und CPU_2
  - ⋆ 3 Prozesse p_1, p_2, p_3 mit $B(p_i) = 0$
  - ⋆ S1 genügt nicht! P1 zu spät beendet
  - ⋆ S2 liefert die korrekte Zuteilung, da alle Bereitzeiten gleich

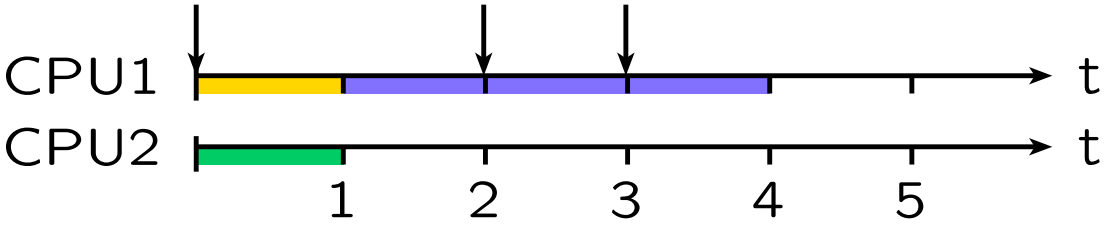
Lastprofil:



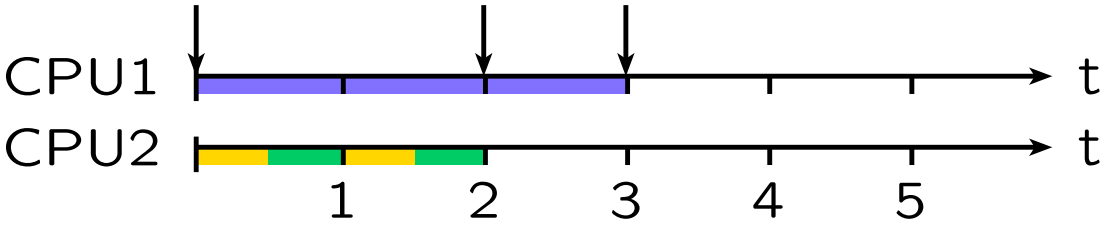
korrekter Plan:



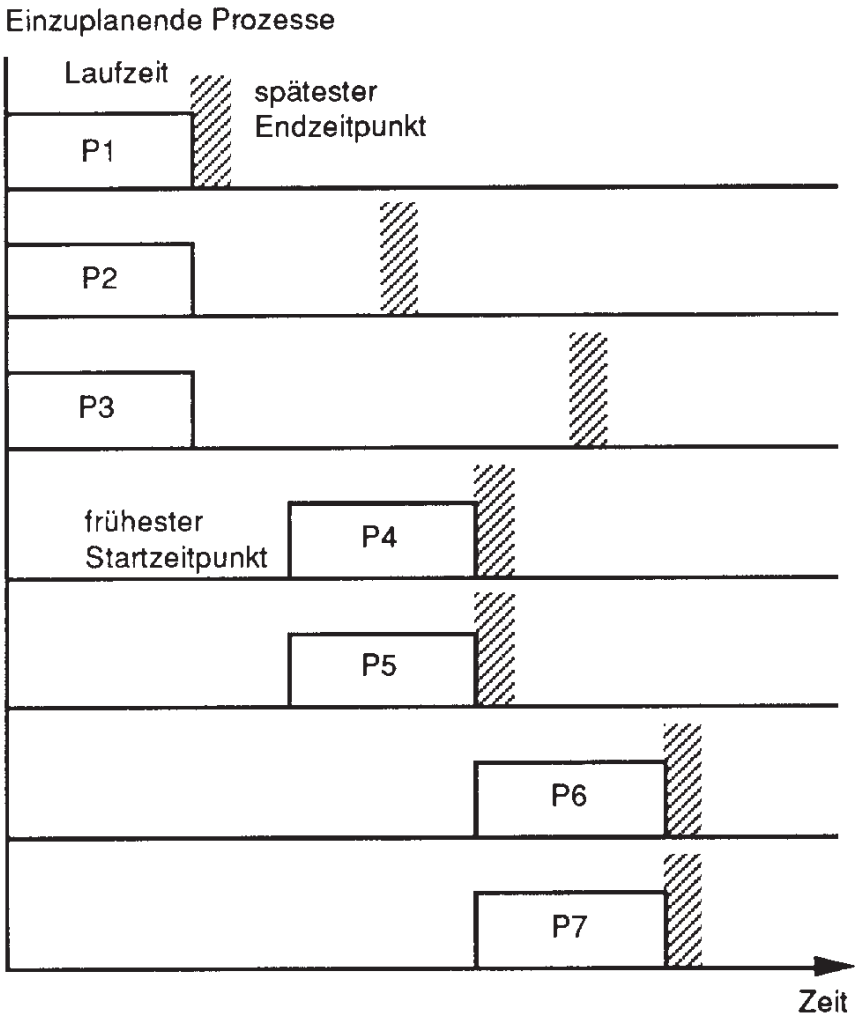
Fristenplanung S1:



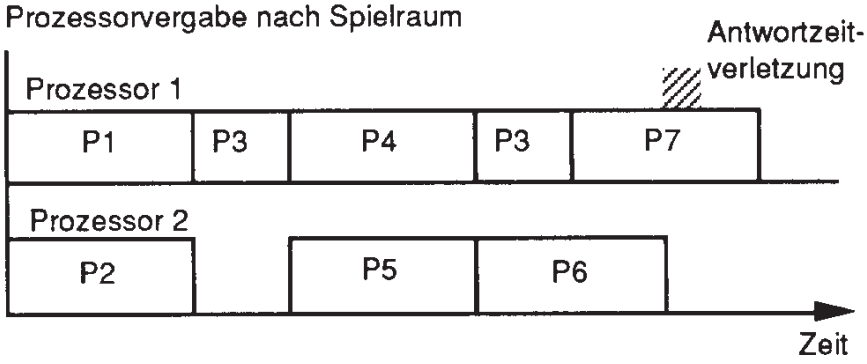
Spielraumplanung S2: $\triangle t = .5$



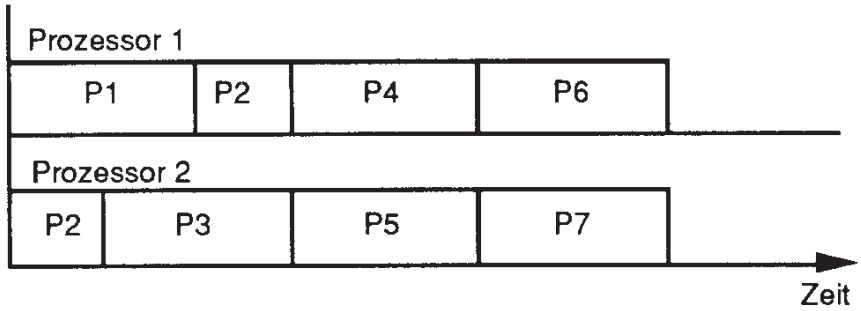
- Nachweis, daß S2 optimal bei gleichen Bereitzeiten aller Prozesse durch Betrachtung der Auslastung. Es wird immer am Prozeß mit geringstem Spielraum gearbeitet, d.h. wenn dort Zeitüberschreitung auftritt, dann auch, falls noch Prozesse mit größerem Spielraum eingeschoben würden.
- Beispiel, daß S2 bei unterschiedlichen Bereitzeiten versagt



(a) Prozessorvergabe nach Spielraum



(b) Zeitgerechte Prozessorvergabe (nicht algorithmisch ermittelt)



- Beweis, daß jede Zuteilungsstrategie versagt, wenn
  - ⋆ unterschiedliche $B_{(pi)}$ und
  - ⋆ die $B_{(pi)}$ nicht vorab bekannt und
  - ⋆ nicht alle Prozesse unter Berücksichtigung ihrer $B_{(pi)}$ zum Zeitpunkt $t = 0$ geplant
  - ⋆ d.h. es versagt auch S2 (kürzester Spielraum) bei unterschiedlichen $B_{(pi)}$
  - ⋆ Vorgehensidee:
    - ◦ n CPU's und n-2 Prozesse ohne Spielraum, diese muessen sofort rechnen, da sonst Fristüberschreitung
    - ◦ damit Problem auf 2 CPU's reduziert
    - ◦ drei weitere Prozesse vorhandenen und einzuplanen
    - ◦ Reihenfolge von Strategie abhängig, alle Reihenfolgen zu betrachten
    - ◦ später treffen dann noch Prozesse ein, die solche Fristen haben, daß auf jeden Fall Fristenüberschreitung auftritt
    - ◦ aber zulässiger Plan existent, wenn alle Prozesse von Anfang an bekannt

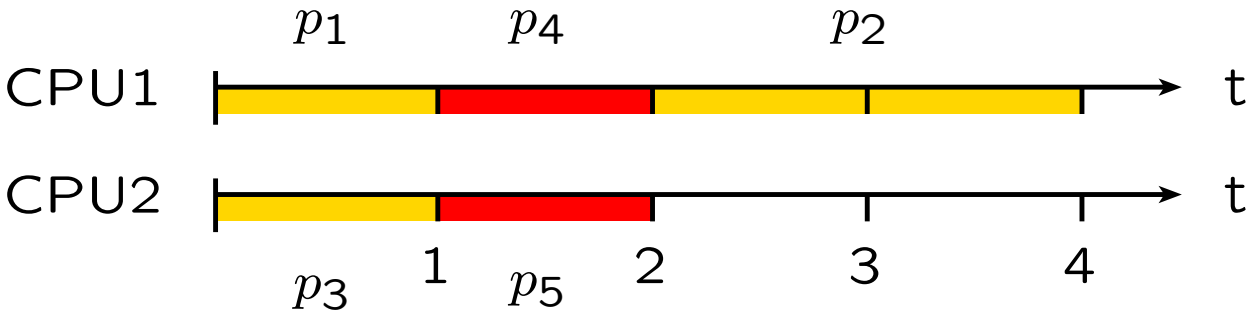
- ⋆ Zahlenwerte:
  - 3 Prozeße p_1, p_2 und p_2 vorhanden

i	$B(P_i)$	$a(P_i)$	$F(P_i)$
1	0	1	1
2	0	2	4
3	0	1	2

  - p_1 ohne Spielraum, rechnet auf CPU1
  - Es gibt jetzt je nach Strategie zwei Fälle zu betrachten: p_2 oder p_3 an CPU2
- ⋆ Fall 1: p_2 an CPU2
  - zur Zeit t=1 muß p_3 begonnen werden
  - zur Zeit t=1 treffen zwei Aufträge ein

i	$B(P_i)$	$a(P_i)$	$F(P_i)$
4	1	1	2
5	1	1	2

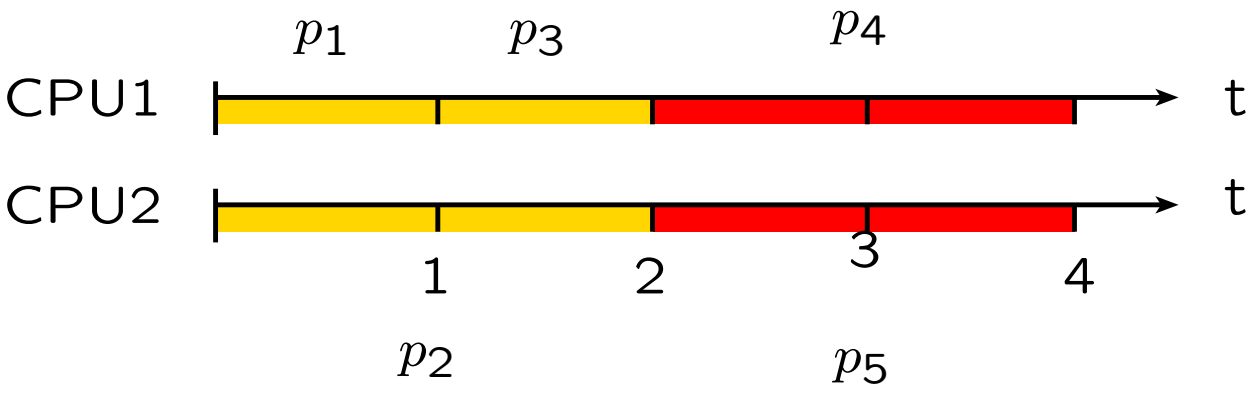
  - damit müssen drei Aufträge ohne Spielraum bearbeitet werden; geht nicht
  - es gibt aber korrekten Plan:



- ⋆ Fall 2: p_3 an CPU2
  - zur Zeit t=1 sind p_1 und p_3 beendet
  - zur Zeit t=1 wird p_2 begonnen
  - zur Zeit t=2 treffen 2 Aufträge ein

i	$B(P_i)$	$a(P_i)$	$F(P_i)$
4	2	2	4
5	2	2	4

  - es müssen noch 5 Zeiteinheiten Rechnerleistung aufgebracht werden
  - es sind aber nur 4 vorhanden bis zu den Fristen bei Zeit 4
  - Plan geht nicht
  - es gibt aber korrekten Plan:

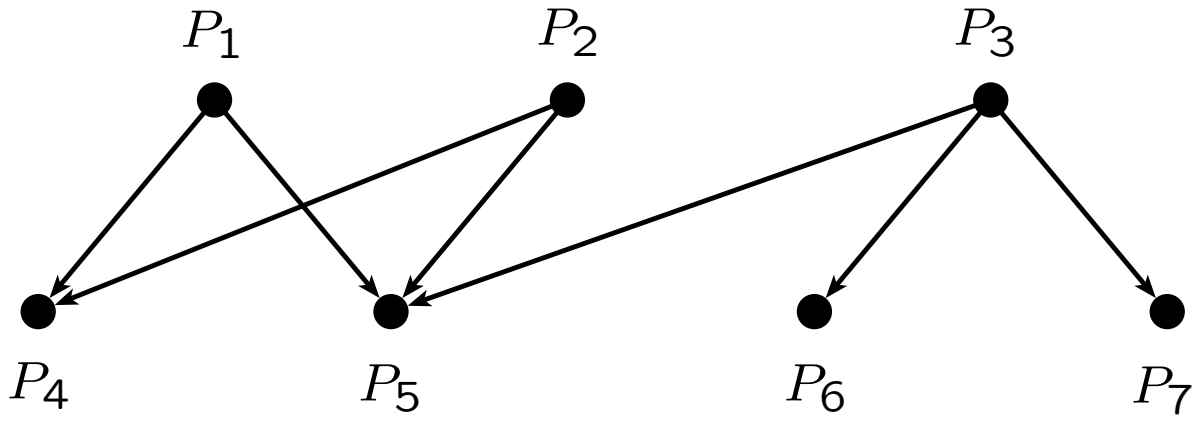


8.11 Anomalien bei nichtpräemptiven Systemen

- Bei den plausiblen Strategien S1 oder S2 können sog. Anomalien auftreten, z.B.:
  - ⋆ zeitweise freiwilliges Ruhenlassen eines Prozessors kann die Gesamtzeit verkürzen
  - ⋆ der Einsatz eines zusätzlichen Prozessors verlängert die Gesamtzeit.
- Beispiel aus Coffman/Denning: Operating Systems Theory
  - ⋆ 3 gleichartige Prozessoren CPU_1, CPU_2, CPU_3
  - ⋆ Präzedenzgraph für 7 Aktionen p_1, $p_2, \ldots$, p_7
  - ⋆ vorgegebene Laufzeiten $a(p_i)$

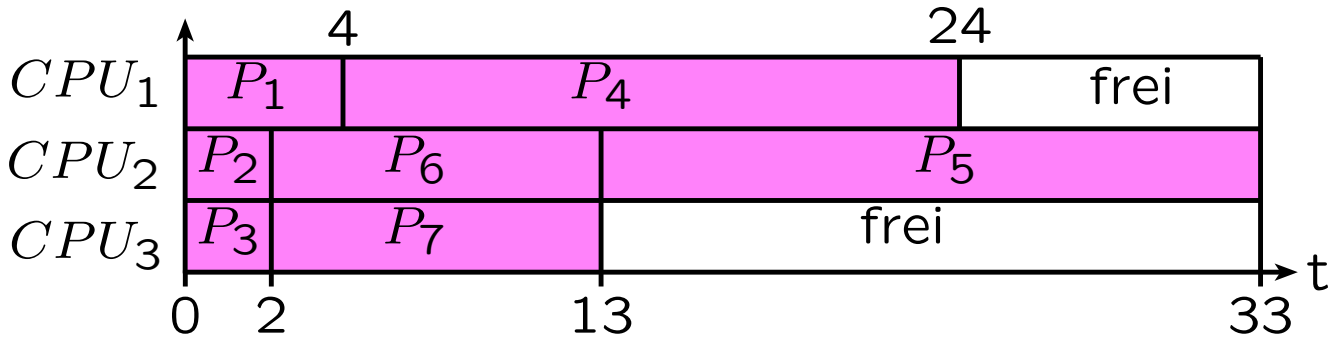
i	$a(p_i)$
1	4
2	2
3	2
4	20
5	20
6	11
7	11

⋆ Präzedenzgraph:

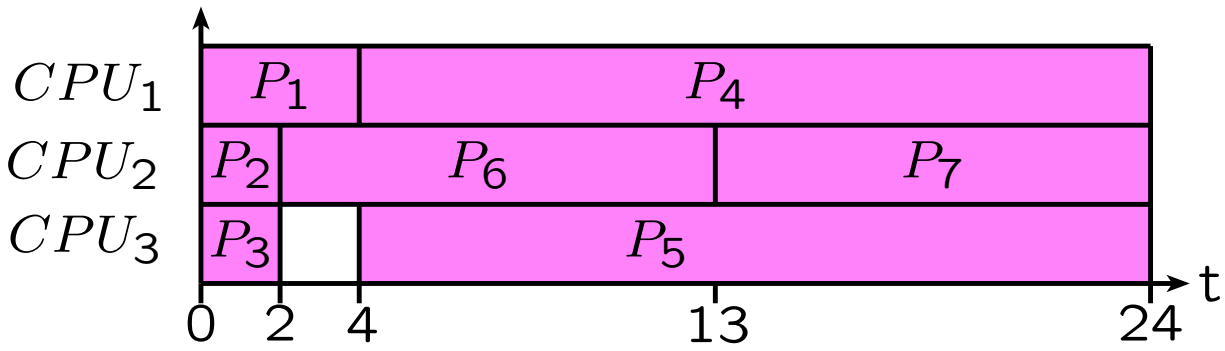


⋆ p_1, p_2, p_3 sofort startbar

⋆ plausible Zuteilungsstrategie: Starten einer Aktion, sobald Vorgänger fertig und eine CPU frei; Gesamtzeit = 33



⋆ Optimale Zuteilung; Gesamtzeit = 24

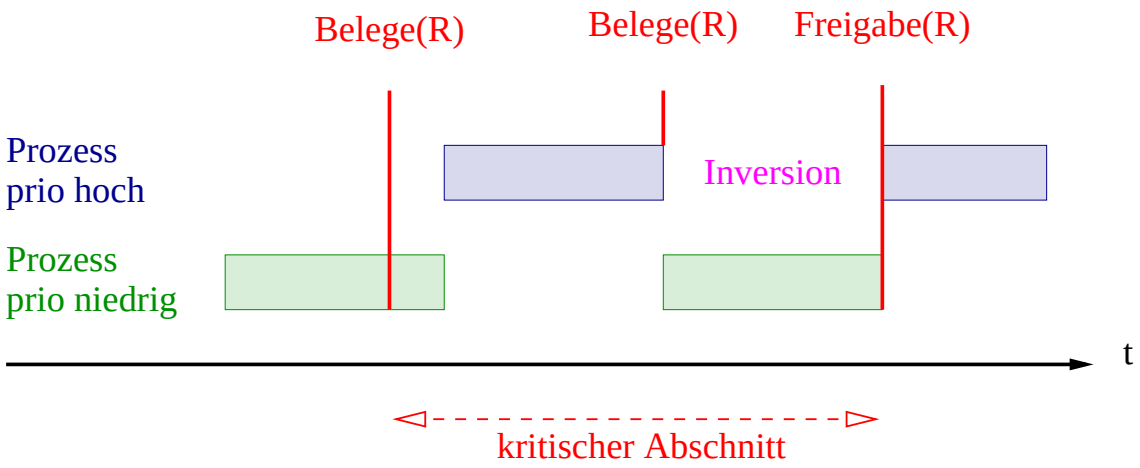


8.12 Zuteilung nach Prioritätszahlen

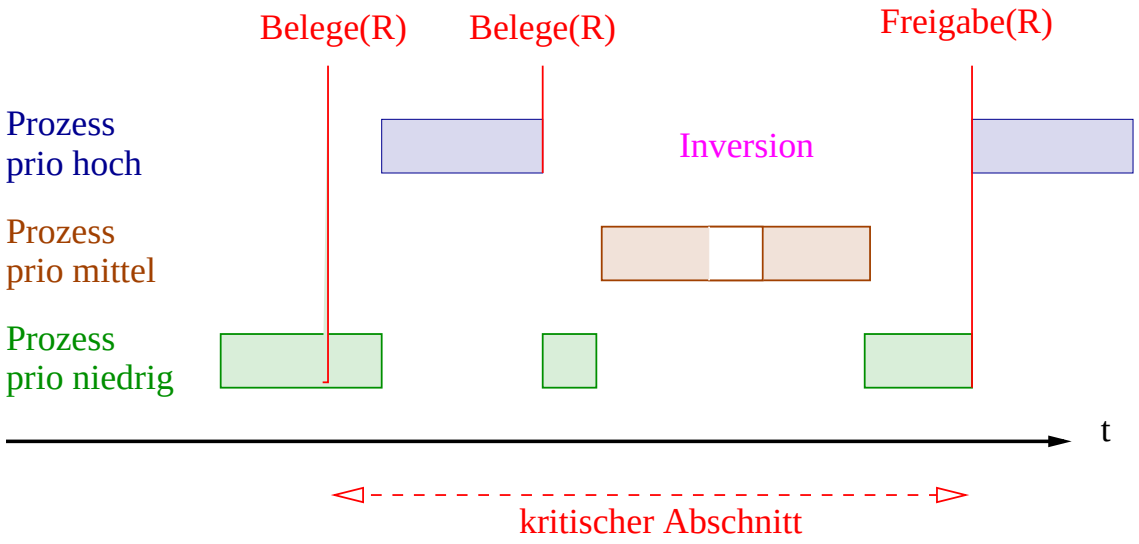
- die einfachen Strategien S1 und S2 werden in der Praxis auch bei Einprozessorsystemen nicht explizit angewandt, da
  - ⋆ kein abgeschlossenes System der Aktionen (Alarme, Interrupts erfordern dynamische Planung)
  - ⋆ Bereitzeitpunkt nur bei zyklischen und Termin–Prozessen bekannt,
  - ⋆ Laufzeiten nicht exakt bekannt, von Daten und Umgebung abhängig
  - ⋆ Synchronisation, Kommunikation und gemeinsame Betriebsmittel verletzen die Forderung nach Unabhängigkeit der Aktionen
- bei großen Systemen Zuteilung nach festen Prioritätszahlen
- Priorität aus Wichtigkeit für den technischen Prozeß und aus Abschätzungen der aktuellen Fristen oder Spielräume
- bei gleicher Priorität meist FIFO–Strategie

- ausreichende Leistungsreserve der CPU mittels worst-case-Betrachtungen
- Prioritätszahlen sind meist vom Typ nat im Bereich [0..255]
- je größer die Zahl, desto höher die Dringlichkeit (z.B. LYNXOS); oft auch umgekehrt (z.B. PEARL, UNIX)
- Einteilung in Prioritätsgruppen, je nach Härte der Zeitbedingungen
- Wann wird umgeplant ?
  - ⋆ Prioritätsinkonsistenz möglichst kurz
  - ⋆ beim Beenden einer Aktion
  - ⋆ beim Übergang einer Aktion in Wartezustand.
  - ⋆ beim Eintreffen einer neuen Anforderung (neuer Prozeß wird aktiv)
  - ⋆ nach bestimmten Zeitintervallen Überprüfung der Situation (z.B. bei Spielraumplanung)

- Behinderung wichtiger Prozesse durch unwichtige darf in Echtzeitsystemen nicht auftreten, also
  - ⋆ Prioritätssreihenfolge bei allen Anforderungen von Betriebsmitteln (CPU, Semaphore, Netzkommunikation, Puffer, Peripherie),
    d.h. Vordrängen in allen Warteschlangen
- Prioritätsinversion (priority inversion) Ein Prozess mit niedriger Priorität blockiert einen Prozess mit höherer Priorität
  - ⋆ begrenzte Inversion (bounded inversion)
    Die Inversion ist durch die Dauer des kritischen Abschnitts beschränkt



- ⋆ unbegrenzte Inversion (unbounded inversion)
  Der kritische Abschnitt wird durch weitere Prozesse auf unbestimmte Zeit blockiert

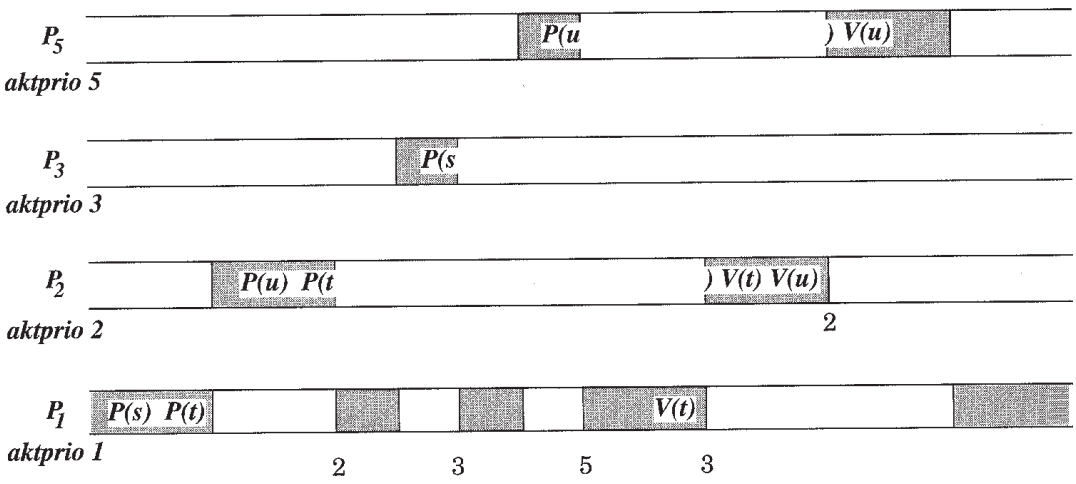


- Prioritätsvererbung (priority inheritance)
  - ⋆ Der Prozess erbt die höhere Priorität des Prozesses, solange dieser das gemeinsame Betriebsmittel blockiert.
  - ⋆ Verhindert unbegrenzte Blockierung
  - ⋆ Die Dauer der Blockierung wird auf die Dauer des kritischen Abschnitts begrenzt
  - ⋆ Die Blockierungen werden hintereinander gereiht (Blockierungsketten)
  - ⋆ Es verhindert keine deadlocks

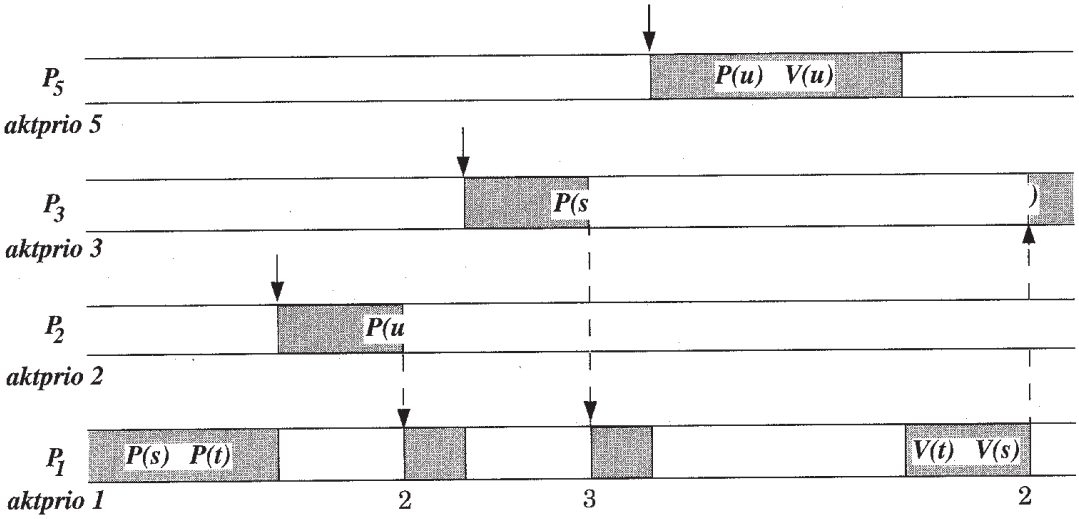
- Prioritätsgrenzen (priority ceiling)
  - ⋆ Jedes Betriebsmittel (Semaphor) s erhält eine Prioritätsgrenze
    $ceil(s)$ =Maximum der Prioritäten der Prozesse, die auf s zugreifen
  - ⋆ Der Prozess p darf ein BM nur blockieren, wenn er von keinem anderen Prozess, der andere BM besitzt, verzögert werden kann
  - ⋆ aktuelle Prioritätsgrenze für Prozess p
    $aktceil(p) = max\{ceil(s)|s \in lockedsem\}$
    $lockedsem$ = Menge aller blockierter BM
  - ⋆ Prozess p darf Betriebsmittel s benutzen, wenn $aktprio(p) > aktceil(p)$
  - ⋆ andernfalls gibt es genau einen Prozeß, der s besitzt. Die Priorität dieses Prozesses wird auf $aktprio(p)$ gesetzt
  - ⋆ Blockierung nur für die Dauer eines kritischen Abschnitts
  - ⋆ Verhindert deadlocks
  - ⋆ schwieriger zu realisieren, zusätzlicher Prozesszustand

- ⋆ vereinfachtes Protokoll:
  Immediate priority ceiling
  Prozesse, die BM s belegen, erhalten die Priorität $ceil(s)$

- Beispiel zur Prioritätsvererbung



- gleiches Beispiel mit Prioritätsgrenzen



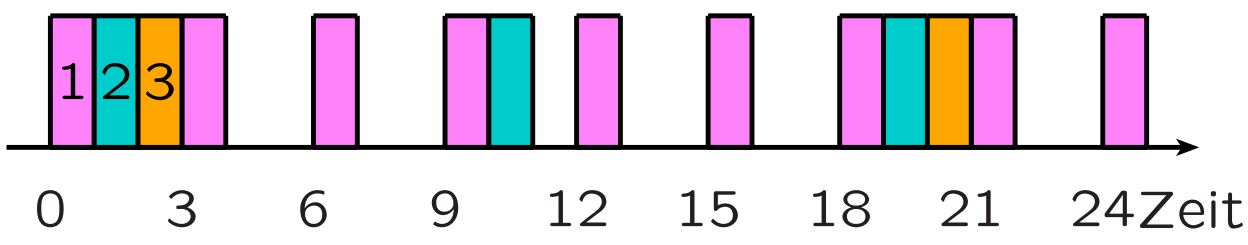
- Ein reales Beispiel für das Inversionsproblem: Mars Pathfinder

`http://mcs.une.edu.au/ iam/Data/threads/node12.html`

8.13 Zeitplanung periodischer Prozesse

- Bezeichnungen
  - ⋆ Prozesse werden periodisch aktiviert
  - ⋆ n Prozesse p_i, mit $1 \leq i \leq n$
  - ⋆ Periode $T(p_i)$
  - ⋆ Ausführungszeit $a(p_i)$
  - ⋆ $B(p_i)$ Bereitzeit relativ zum Beginn einer Periode
  - ⋆ $F(p_i)$ Frist relativ zum Beginn einer Periode
- Beispiel: Meßwerterfassung
  - ⋆ 3 Prozesse p_i
  - ⋆ jeweils Werte einlesen, umrechnen, skalieren, auf Platte speichern
  - ⋆ Kenngrößen der Prozesse:

i	$T(p_i)$	$B(p_i)$	$a(p_i)$	$F(p_i)$
1	3	0	1	3
2	9	0	1	9
3	18	0	1	18



- Einplanbarkeit aufgrund Last
  - ⋆ Auslastung durch Prozeß P_i:

$$\varrho_i = \frac{a(p_i)}{T(p_i)}$$

  - ⋆ Gesamte Auslastung :

$$\varrho = \sum_{i=1}^{n} \varrho_i = \sum_{i=1}^{n} \frac{a(p_i)}{T(p_i)}$$

  - ⋆ Bei m Einheiten eines BM gilt: $\varrho \leq m$ notwendig für Zeiteinhaltung, aber nicht hinreichend!
- Einplanung nach Fristen
  - ⋆ bei 1 BM, $\varrho \leq 1$ und $T(p_i) = F(p_i)$ für alle i ist Einplanung nach Fristen optimal
  - ⋆ Beweisidee: vor dem Verletzen einer Frist ist das BM nie unbeschäftigt

- Einplanung nach Raten
  rate–monotonic scheduling,
  smallest period–first
  - ⋆ Rate $R_i = 1/T(p_i)$
  - ⋆ hohe Rate = hohe Priorität
  - ⋆ Prozesse mit hohen Raten zuerst bedient
  - ⋆ Ratenplanung benutzt also statische Prioritäten
  - ⋆ optimal, falls eine Lösung mit statischen Prioritäten existiert
  - ⋆ Verfahren mit dynamischen Prioritäten können aber evtl. bessere Ergebnisse liefern
  - ⋆ Einplanbarkeitstest 1 für Ratenplanung
    - ◦ Bei n Aktionen ist Ratenplanung sicher erfolgreich,falls

$$\varrho \leq \varrho_{max} = n * \left(2^{1/n} - 1\right)$$

$$\lim_{n \to \infty} \varrho_{max} = ln2 \approx 0.69$$

    - ◦ Beweis (Liu, Layland 1973) durch "worstcase"–Analyse

- Achtung!
  Das heißt aber nicht, daß Ratenplanung für Systeme mit $\varrho > \varrho_{max}$ nicht doch Lösungen findet, nur ist dies nicht garantiert.
- Für Spezialfälle (Perioden gleich oder Perioden Teiler einer größeren) gibt es günstigere Abschätzungen

⋆ Einplanbarkeitstest 2 für Ratenplanung

- Für Satz periodischer Aktionen P_i gibt Ratenplanung einen erfolgreichen Plan, falls unter der Annahme gleichzeitigen Starts (= worst case) alle Aktionen innerhalb ihrer ersten Periodenzeit $T(p_i)$ beendet werden
- Test ist auch bei grosser Anzahl Prozesse durch Simulation effizient ausführbar, falls die Perioden nicht teilerfremd
- Beweisidee: Gleichzeitiger Start ist worst case-Situation, da alle höherprioren Aktionen eine Ausführung von P_i verzögern

8.14 Vergleich der Planungsverfahren

- Einordnung der eingeführten Verfahren
- für Einprozesorsystem (Ausnahme angegeben)
- und statische Planung

Strategie	präemptiv	nichtpräemptiv
Suchen		optimale Pläne O(n!), NP-vollständig
Fristen Spielraum	optimal	optimal bei gleicher Bereitzeit
Spielraum (Mehrpr.)	optimal bei gleicher Bereitzeit	NP-vollständig Anomalien
Monotone Raten	optimal bei beschränkter Auslastung	