Echtzeitbetriebssysteme

Oliver Jack

Ernst-Abbe-Hochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2025

Ernst-Abbe-Hochschule Jena
University of Applied Sciences

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Lernziele

- Kenntnis elementarer Planungsstrategien für Ein- und Mehrprozessor-Systeme
- Verständnis der Bewertung bzgl. der Optimalität von Planungsstrategien.

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

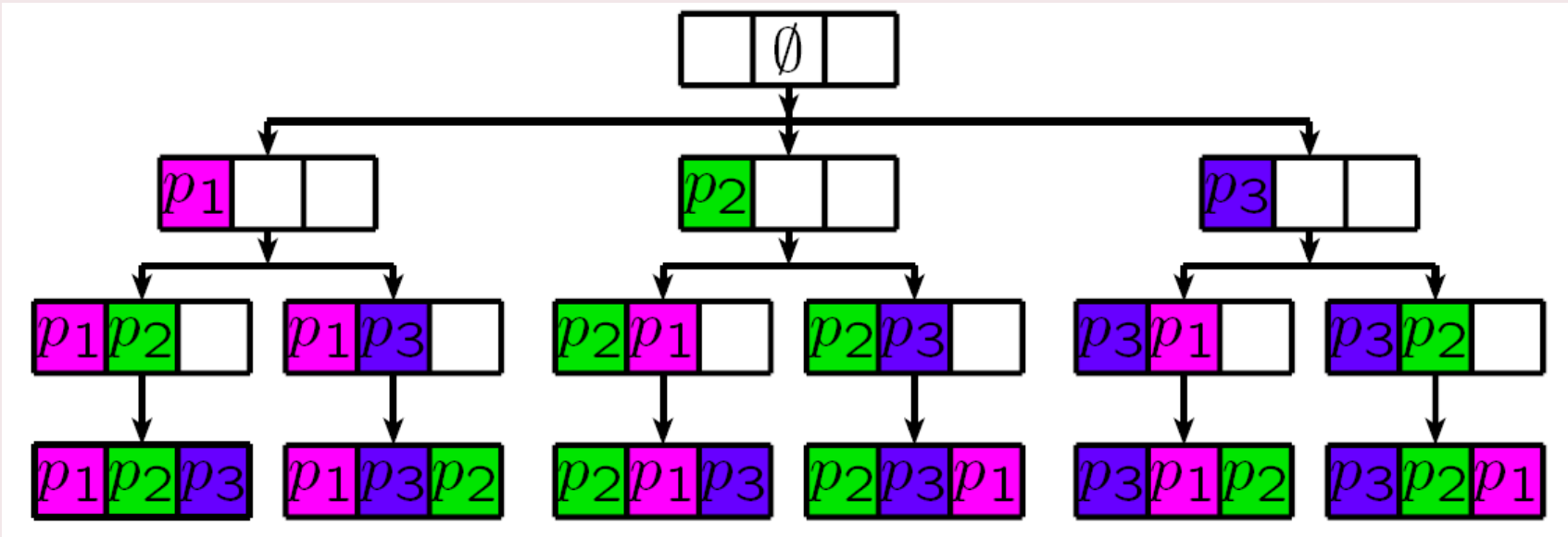
Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Vollständige Suche

- Zunächst nicht-unterbrechbare Aktionen vorausgesetzt
- Exakte Planung über Durchsuchen des Lösungsraums

Beispiel

- $n = 3$ Prozesse p_1, p_2, p_3 und 1 CPU
- Suchbaum:



Vollständige Suche

- $n!$ (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

Vollständige Suche

- Leichte Verbesserung der Komplexität durch
  - Abbrechen der Bildung einer Sequenz, wenn bei einer der Aktionen eine Zeitüberschreitung auftritt
  - Heuristiken, z. B. vorsortieren nach $B(p_i)$
- Es existieren viele Algorithmen z. B. im Buch: J.Blazewicz,et al.: *Scheduling in Computer and Manufacturing Systems.* 2.Auflage, Springer 1994.
- Bei unterbrechbaren Tasks führt Granulariät der Zeitintervalle mit Präzedenzen der Tasks zu System mit nicht-unterbrechbaren Tasks.
- Weitere Probleme treten durch Synchronisation der Aktionen und die Konkurrenz um weitere Betriebsmittel auf.

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Einprozessor-System

- Einziges verplantes Betriebsmittel ist 1 CPU
- Kriterien zur Prüfung auf Einhaltbarkeit aller Zeitbedingungen sind relativ einfach.
- n Prozesse
- Geordnet nach Fristen: $F(p_i) \leq F(p_{i+1})\ (i = 1, \ldots, n-1)$
- Gleiche Bereitzeiten $B(p_i) = 0$
- Es muss notwendig gelten:

$$\forall i : \Delta t(p_i, t) \geq \sum_{k=1}^{i} a(p_k, t)\ (i = 1, \ldots, n)$$

- Gesucht: Strategien zur dynamisch korrekten Ordnung der Prozesse

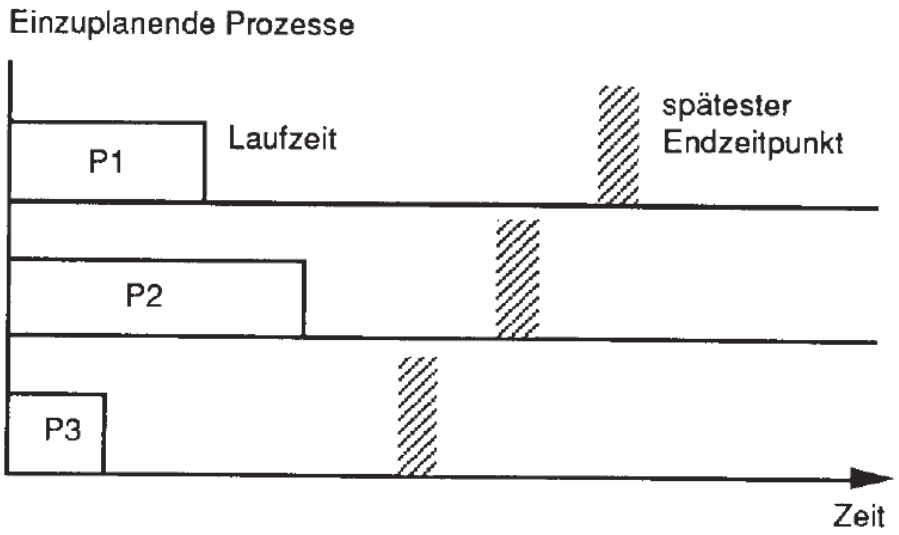
Strategien

- S1: Einplanung nach Fristen
  - next deadline scheduling
  - Der Prozess mit der nächsten Frist F erhält als nächster die CPU
  - Gibt es keinen rechenbereiten Prozess, bleibt der Prozessor untätig (idle).
- 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.

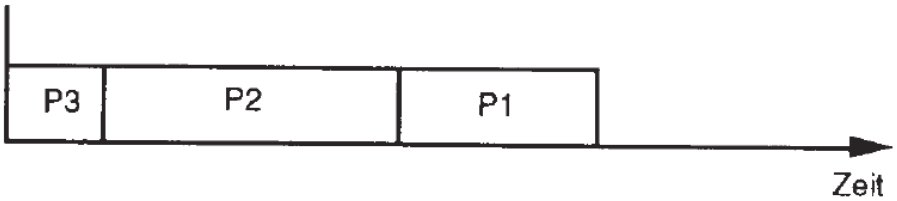
Einplanung nach kleinstem Spielraum

- $l(p_j, t) = F(p_j) - t - a(p_j, t)$
- 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.
- S2 erkennt früher als S1, wenn Zeitbedingungen nicht eingehalten werden können.

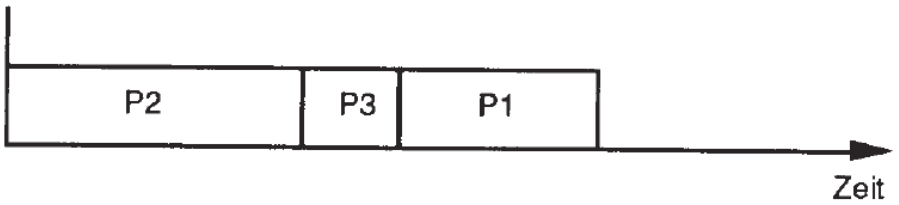
Einplanung nach S1 und S2



(a) Prozessorvergabe nach Antwortzeit



(b) Prozessorvergabe nach Spielraum



Optimalität von S1 und S2

- Voraussetzung: Einprozessorsystem, gleiche Bereitzeiten,
- Satz: Die Zuteilungsstrategien S1 und S2 sind optimal.

Planen nach Fristen (S1)

- Eines der verbreitetsten Scheduling-Verfahren
- Grund ist weniger die erreichte Güte des Plans.
- Grund ist die breite Einsetzbarkeit:
  - auf unterbrechbare Prozesse
  - auf nicht unterbrechbare Prozesse
  - in statischen Planungsverfahren
  - in dynamischen Planungsverfahren
- Optimalität nur bei gleichen Bereitzeiten $B(p_i) = 0$

Beweis der Optmalität 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

Beweis der Optimalität von S1 (Forts.)

- 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:

$$\begin{aligned} i &= \text{pz}(\text{PlanF}, t) \\ j &= \text{pz}(\text{PlanZ}(t), t) \\ i &\neq j \text{ (nach Voraussetzung)} \end{aligned}$$

Beweis der Optimalität von S1 (Forts.)

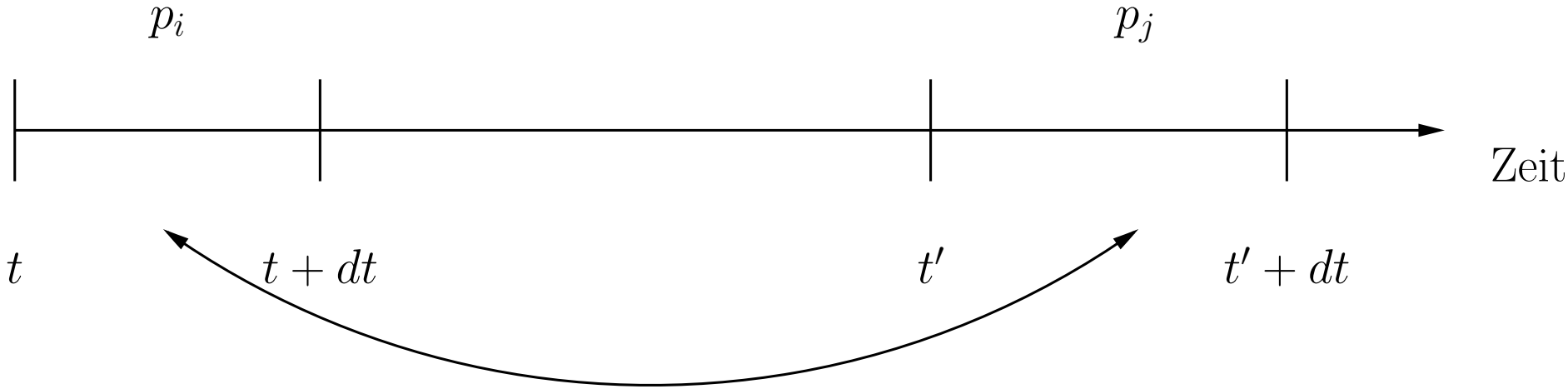
- Betrachte Zeitintervall dt
- Fall $i < j$:
  - $F(p_i) \leq F(p_j)$ wegen Ordnung der Prozesse
  - $t + dt = F(p_j)$ da PlanZ zulässiger Plan
  - da p_i in PlanF im Intervall $(t, t + dt)$ rechnet und die Pläne bis zur Zeit t identisch sind, kann p_i auch in PlanZ noch nicht beendet sein, also gilt:

$$\exists t' \geq t + dt : i = \text{pz}(\text{PlanZ}(t), t') = \text{pz}(\text{PlanZ}(t), t' + dt)$$

$$t' + dt \leq F(pi) \leq F(pj)$$

Beweis der Optimalität von S1 (Forts.)

- 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.



Beweis der Optimalität von S1 (Forts.)

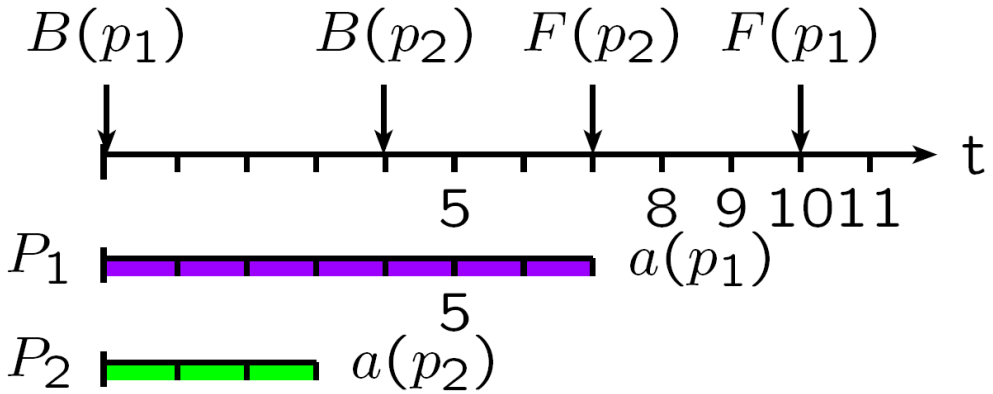
- Fall $i > j$:
  - Rechnet in PlanF zur Zeit t der Prozess p_i, dann sind alle Prozesse p_j mit $j < i$ bereits beendet
  - PlanF und PlanZ(t) stimmen bis zur Zeit t überein
  - der Fall $i > j$ kann also gar nicht auftreten
- Damit ist gezeigt: Jeder zulässige Plan kann ohne Verletzung der Fristen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

Notwendig: Alle Bereitzeiten gleich Null

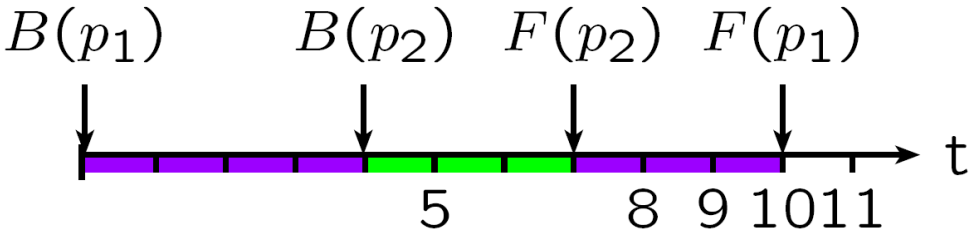
- S1 und S2 liefern unzulässige Pläne, falls dies nicht der Fall ist.
- Allerdings ist eine Modifikation möglich, so dass korrekte Pläne auch in diesem Fall erstellt werden:
  - Präemptive Strategie
  - Neuplanung bei jeder Bereitzeit
  - Einplanung nur derjenigen Prozesse, deren Bereitzeit erreicht ist
  - Entspricht Neuplanung, wenn ein Prozess aktiv wird
  - Evtl. Zeitscheiben für Prozesse mit gleichem Spielraum

Notwendig: Alle Bereitzeiten gleich Null, Beispiel

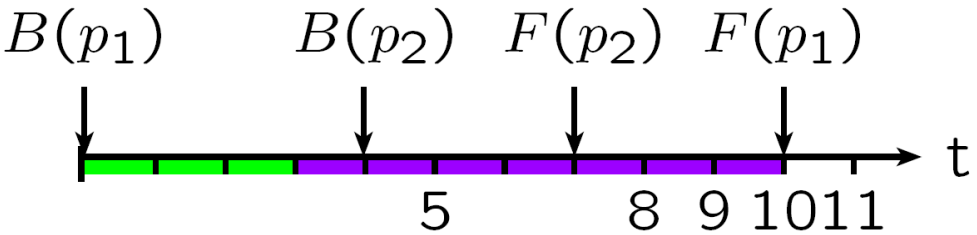
Lastprofil:



korrekter Plan:

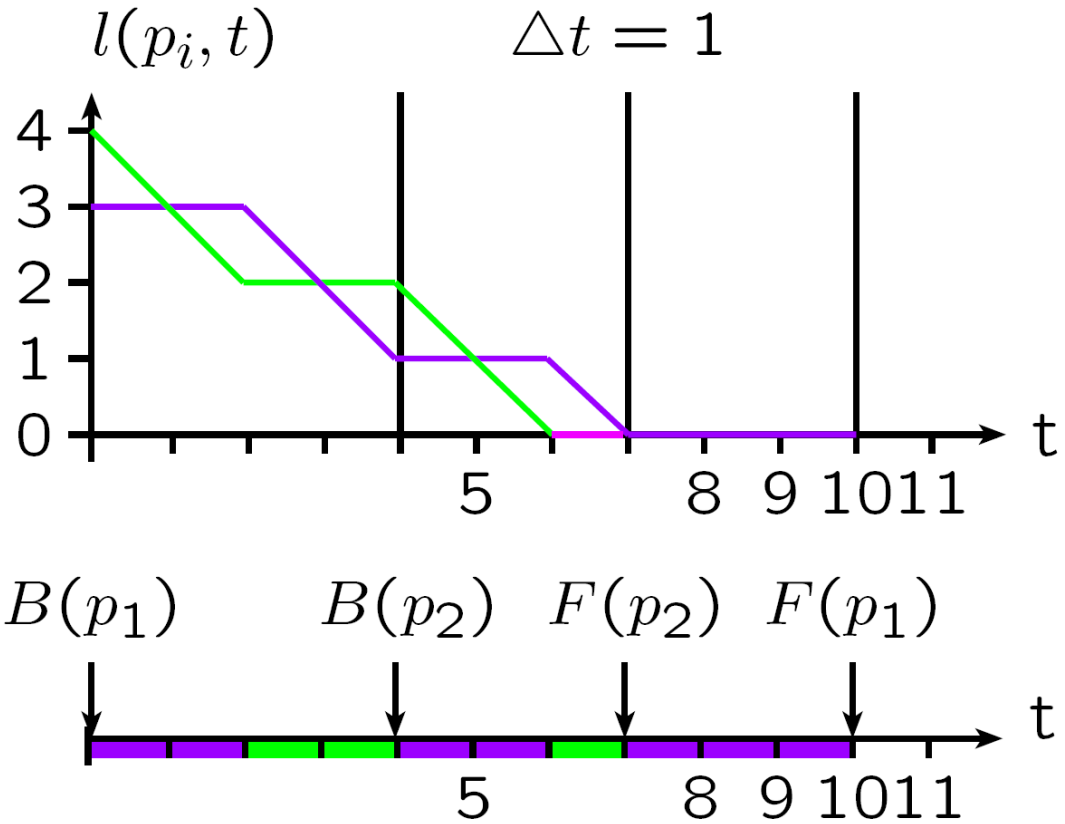


Fristenplanung:



Notwendig: Alle Bereitzeiten gleich Null Beispiel, (Forts.)

kleinster Spielraum:



Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Eigenschaften

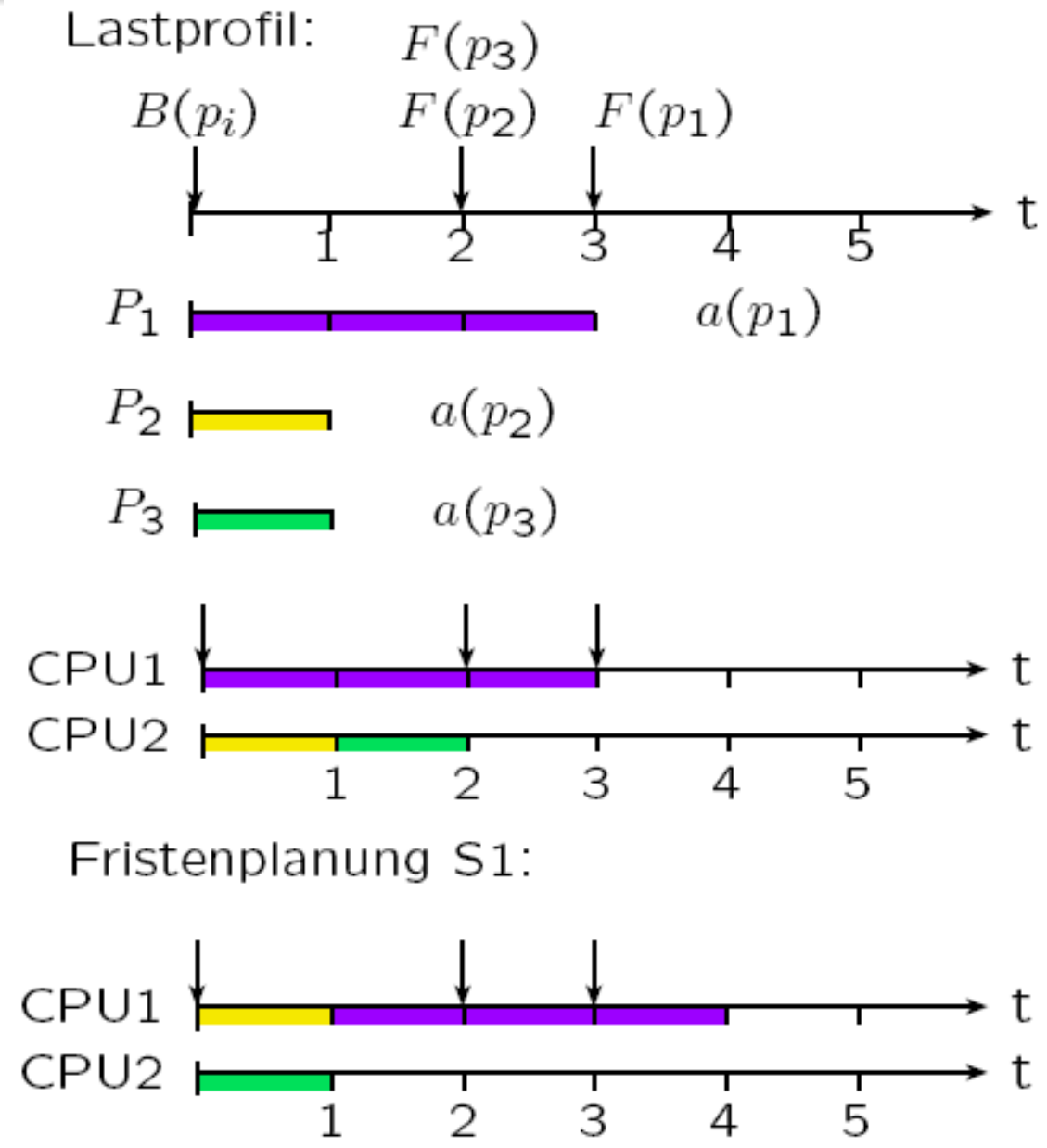
- Gehört zur Klasse der Zuteilungsverfahren für homogene, austauschbare Betriebsmittel.
- S1 ist nicht optimal, egal ob präemptive oder nicht-präemptive Strategie.
- S2 ist nur optimal, falls alle Bereitzeitpunkte $B(p_i)$ gleich sind
- Korrekte Zuteilungsalgorithmen erfordern das Abarbeiten von Suchbäumen mit NP-Aufwand oder geeignete Heuristiken.

S1 ist nicht optimal

Gegenbeispiel zu S1 ist optimal

- präemptiv
- Zwei Prozessoren CPU1 und CPU2
- Drei Prozesse $p1, p2, p3$ mit $B(p_i) = 0$
- S1 genügt nicht! p_1 zu spät beendet
- S2 liefert die korrekte Zuteilung, da alle Bereitzeiten gleich sind.

S1 ist nicht optimal (Forts.)

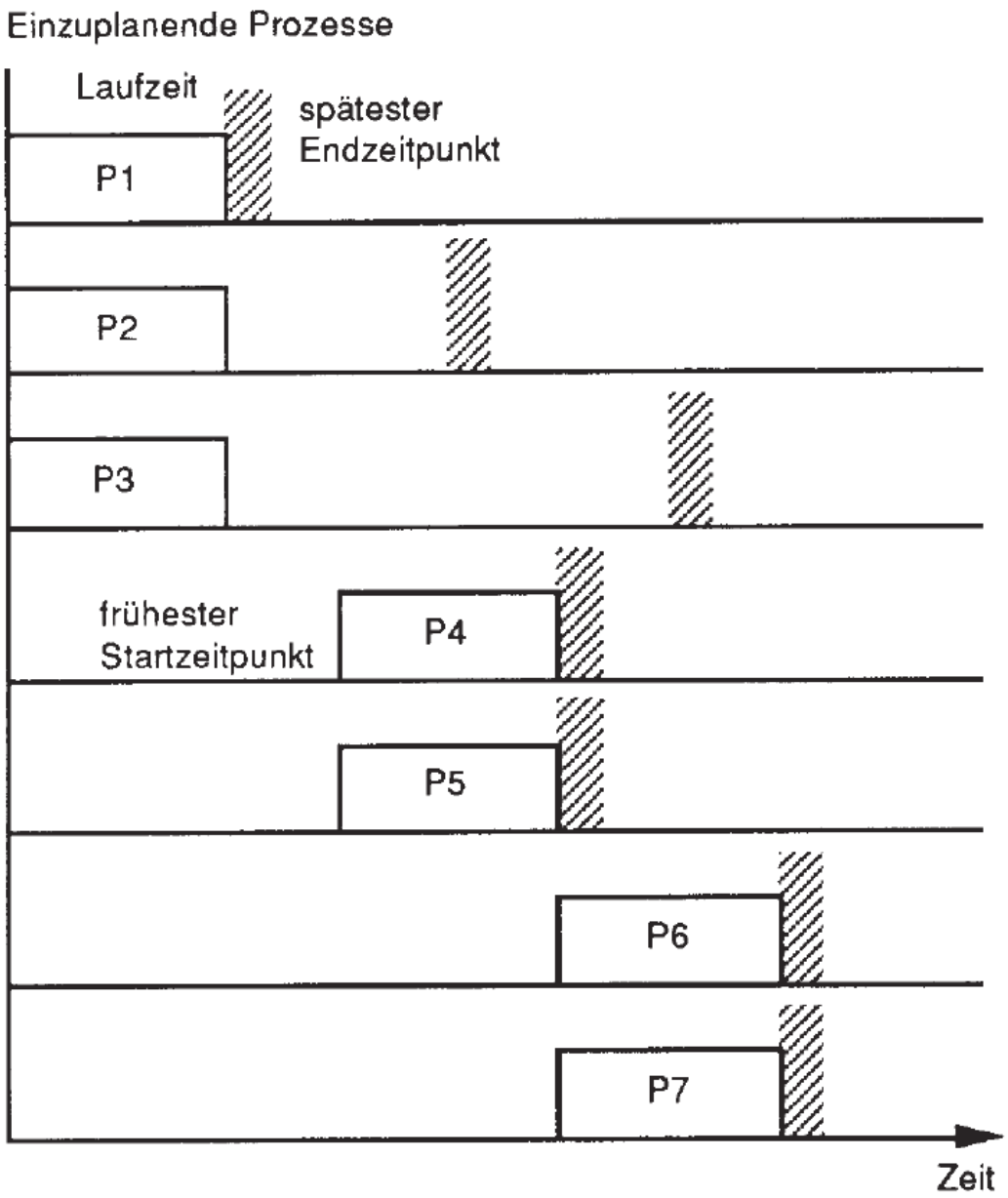


S2 ist optimal bei gleichen Bereitzeiten

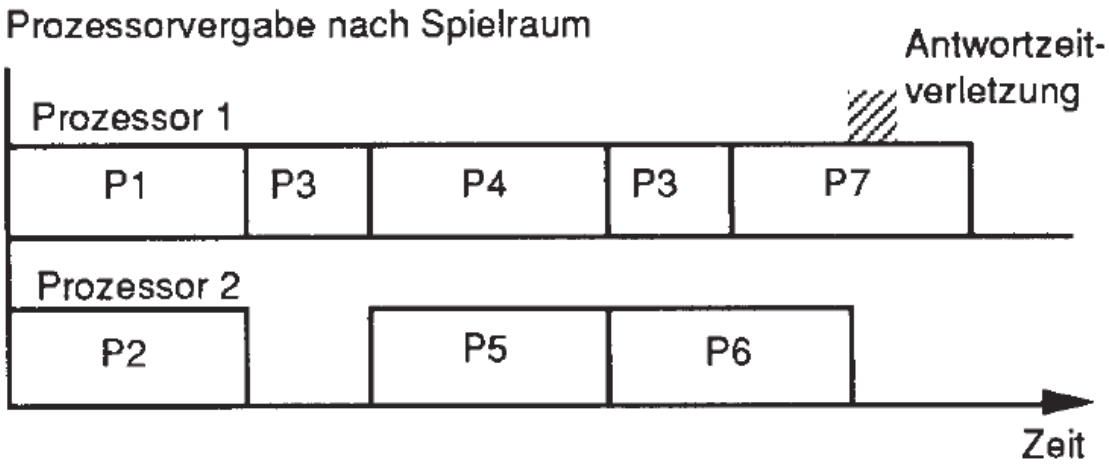
Nachweis, dass S2 optimal bei gleichen $B(p_i)$ Bereitzeiten aller Prozesse durch Betrachtung der Auslastung.

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.

S2 versagt bei unterschiedlichen Bereitzeiten

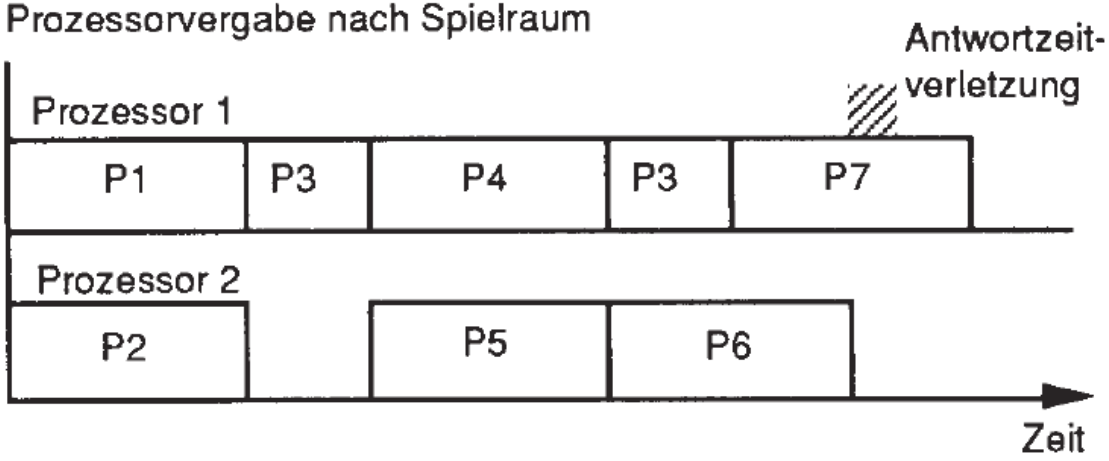


S2 versagt bei unterschiedlichen Bereitzeiten (Forts.)

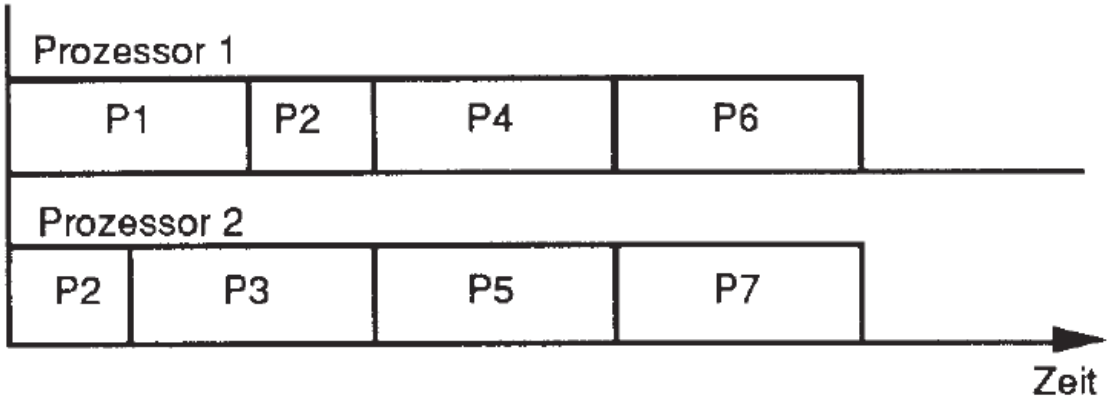


S2 versagt bei unterschiedlichen Bereitzeiten (Forts.)

Prozessorvergabe nach Spielraum



Zeitgerechte Prozessorvergabe (nicht algorithmisch ermittelt)



Nichtoptimalität von S1 und S2

Jede Zuteilungsstrategie versagt, wenn

- unterschiedliche $B(p_i)$ und
- die $B(p_i)$ nicht vorab bekannt und
- nicht alle Prozesse unter Berücksichtigung ihrer $B(p_i)$ zum Zeitpunkt $t = 0$ geplant

D. h. auch S2 (kürzester Spielraum) versagt bei unterschiedlichen $B(p_i)$.

Nichtoptimalität von S1 und S2 (Forts.)

Beweisidee:

- n CPUs und $n-2$ Prozesse ohne Spielraum, diese müssen sofort rechnen, da sonst Fristüberschreitung
- damit Problem auf 2 CPUs 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

Nichtoptimalität von S1 und S2 (Forts.)

Daten:

- Drei Prozesse p_1, p_2, p_3 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

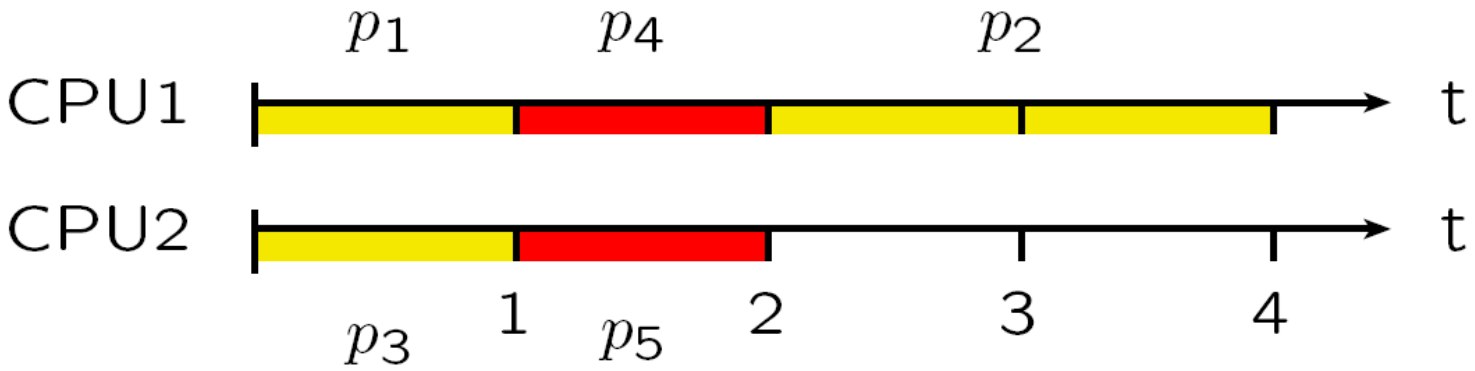
Nichtoptimalität von S1 und S2 (Forts.)

Fall 1: p2 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; nicht möglich.
- Es gibt aber zulässigen Plan



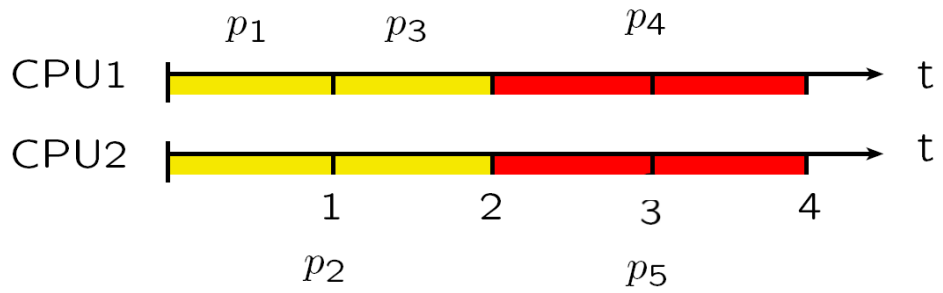
Nichtoptimalität von S1 und S2 (Forts.)

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 fünf Zeiteinheiten Rechnerleistung aufgebracht werden
- Es sind aber nur vier vorhanden bis zu den Fristen bei Zeitpunkt 4; Plan nicht zulässig



- Es gibt aber zulässigen Plan

Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Anomalie bei S1 und S2

Beispiel

Im nicht-präemptiven System kann zeitweise freiwilliges Ruhenlassen eines Prozessors die Gesamtzeit verkürzen.

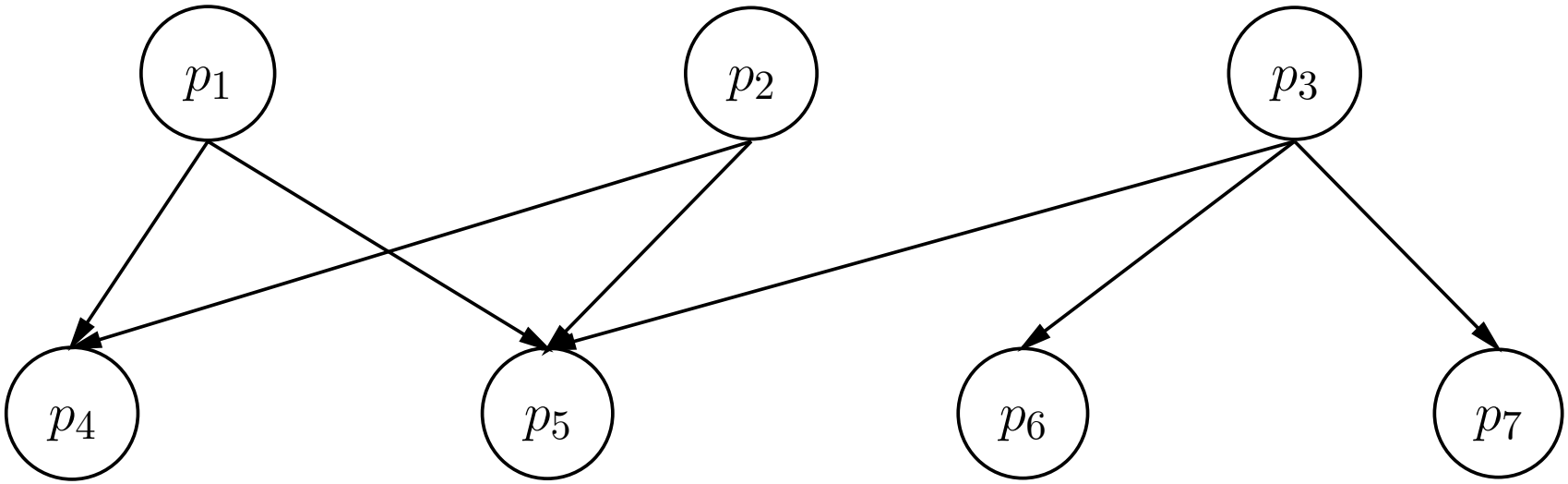
- 3 gleichartige Prozessoren CPU1, CPU2, CPU3
- Präzedenzgraph für 7 Aktionen $p_1, p_2, \ldots, p_7$
- vorgegebene Laufzeiten $a(p_i)$

Anomalie bei S1 und S2 (Forts.)

- Laufzeiten

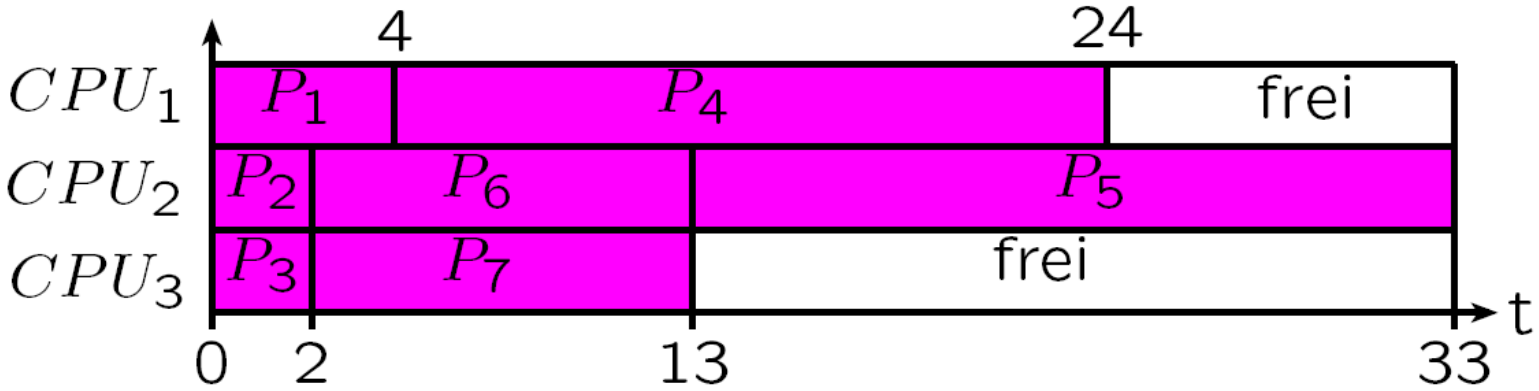
i	1	2	3	4	5	6	7
$a(p_i)$	4	2	2	20	20	11	11

- Präzedenzgraph

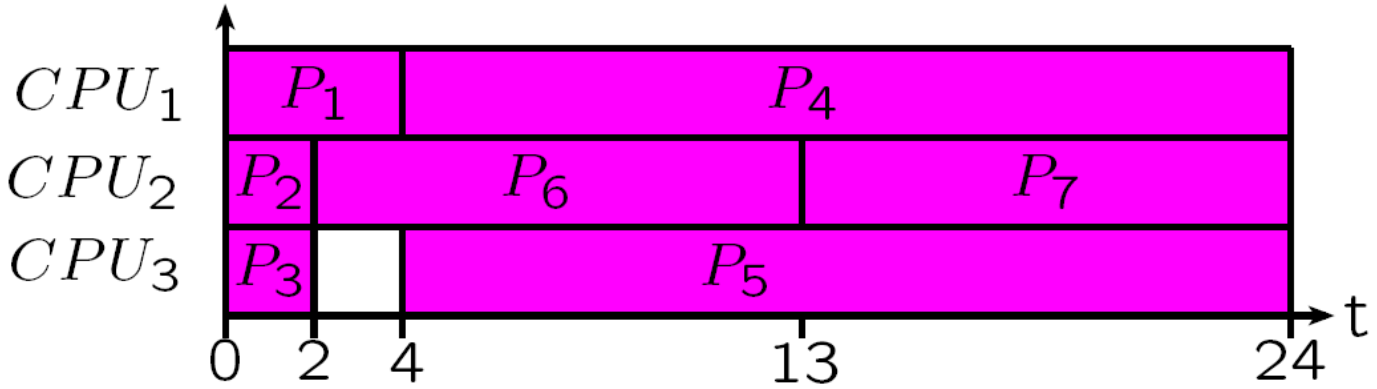


Anomalie bei S1 und S2 (Forts.)

- 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



Lerneinheit 5. Scheduling-Verfahren (Teil 1)

Zusammenfassung

- Planen nach Fristen und Planen nach kleinstem Spielraum sind grundlegende Strategien.
- Bei Einprozessorsystemen sind beide Strategien optimal.
- Bei Mehrprozessorsystemen sind beide Strategien nicht optimal.