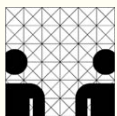


B Betriebssysteme

- B1 Betriebssysteme: Einführung und Motivation
- B2 Prozesse: Scheduling und Betriebsmittel-
zuteilung**
- B3 Speicherverwaltung
- B4 Dateisysteme
- B5 Ein-/Ausgabe

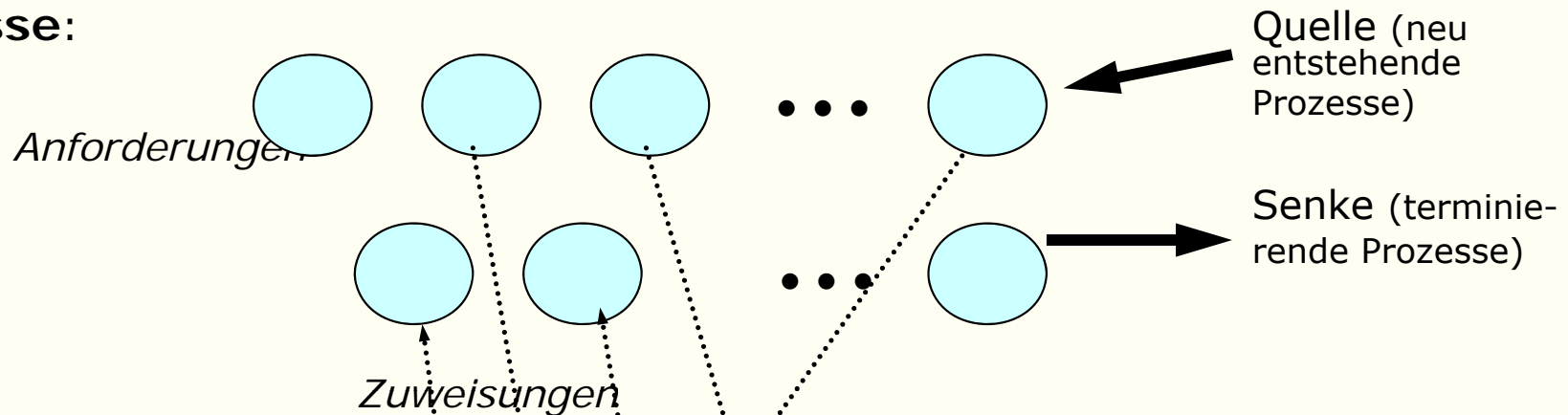


B2 Prozesse: Scheduling & Betriebsmittelzuteilung

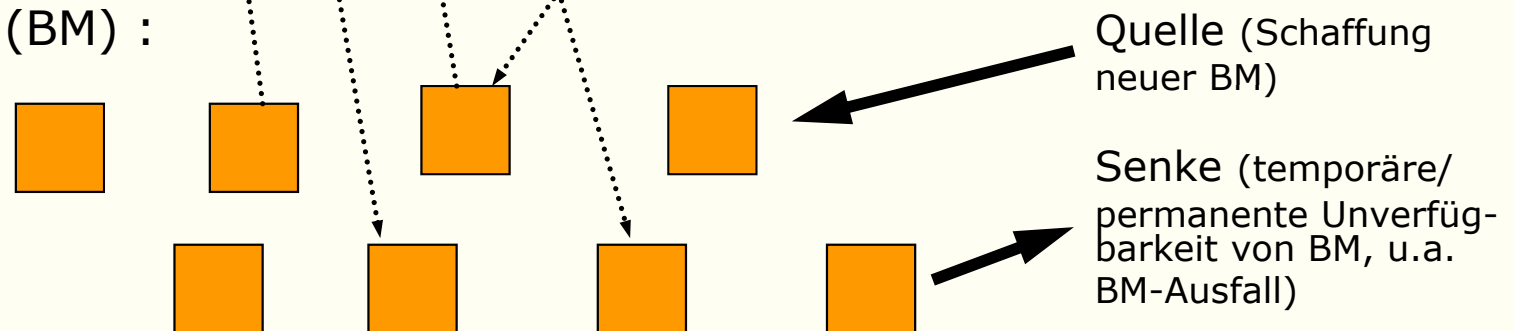
B2.1 Scheduling: Anforderungen und Randbedingungen

Das allgemeine Scheduling-Problem:

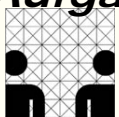
Prozesse:



Betriebsmittel (BM) :



Aufgabe: Erstellung und Aktualisierung eines Arbeitsplans.



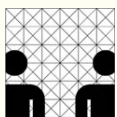
Das Scheduling-Problem:

Situation: die konkurrenten Prozesse P_1, \dots, P_m bewerben sich um die Betriebsmittel B_1, \dots, B_n .

Aufgabe: Löse das Zuteilungsproblem durch Aufstellung einer Rangfolge (=Bearbeitungsreihenfolge) für die Prozesse.

Ziele des Schedulings (Auswahl):

- Maximierung des Durchsatzes eines Rechensystems,
- Minimierung der Antwortzeiten eines Rechensystems,
- Minimierung der Schwankungen der Kosten eines Auftrags,
- Erhaltung der Konsistenz von Datenbeständen.



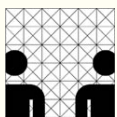
Betriebsmittel: Beispiele und Eigenschaften

Beispiele für Betriebsmittel:

- Speicher,
- Prozessoren (insbes. CPU, d.h. Rechenzeit),
- Geräte,
- Dateien,
- Nachrichten,
- Beschreibungsblöcke,
- Prozessnummern,
- Semaphore.

mögliche **Eigenschaften** von Betriebsmitteln:

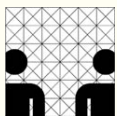
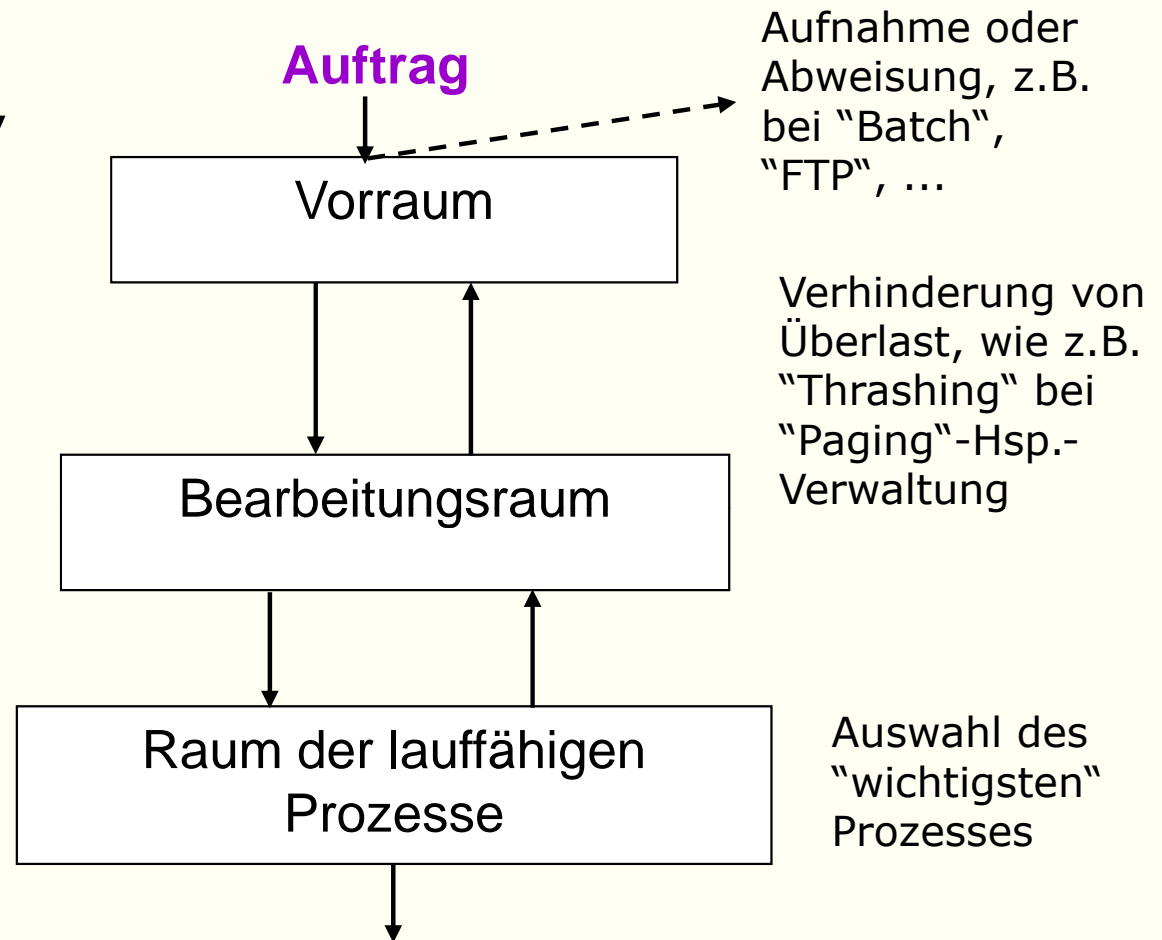
- verbrauchbar,
- wieder verwendbar,
- mehrfach zuteilbar,
- jederzeit entziehbar,
- teilbar.



Planung der Prozessausführungsreihenfolge

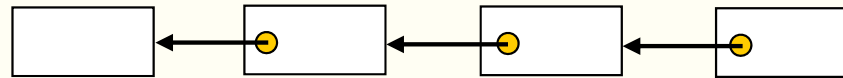
Drei Zeithorizonte der Planung:

- *langfristige* Planung,
- *mittelfristige* Planung,
- *kurzfristige* Planung.

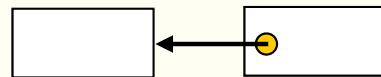


Betriebssystem als Sammlung von Bedienstationen:

- Warteschlange ("Queue") vor BM1:



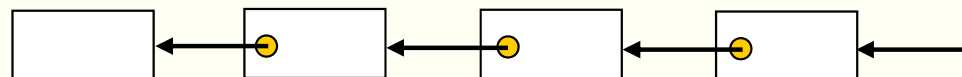
- Warteschlange ("Queue") vor BM2:



- Warteschlange ("Queue") vor BM3:

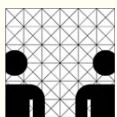


- "*Ready Queue*" für CPU(s), d.h. Warten auf CPU-Zeit:

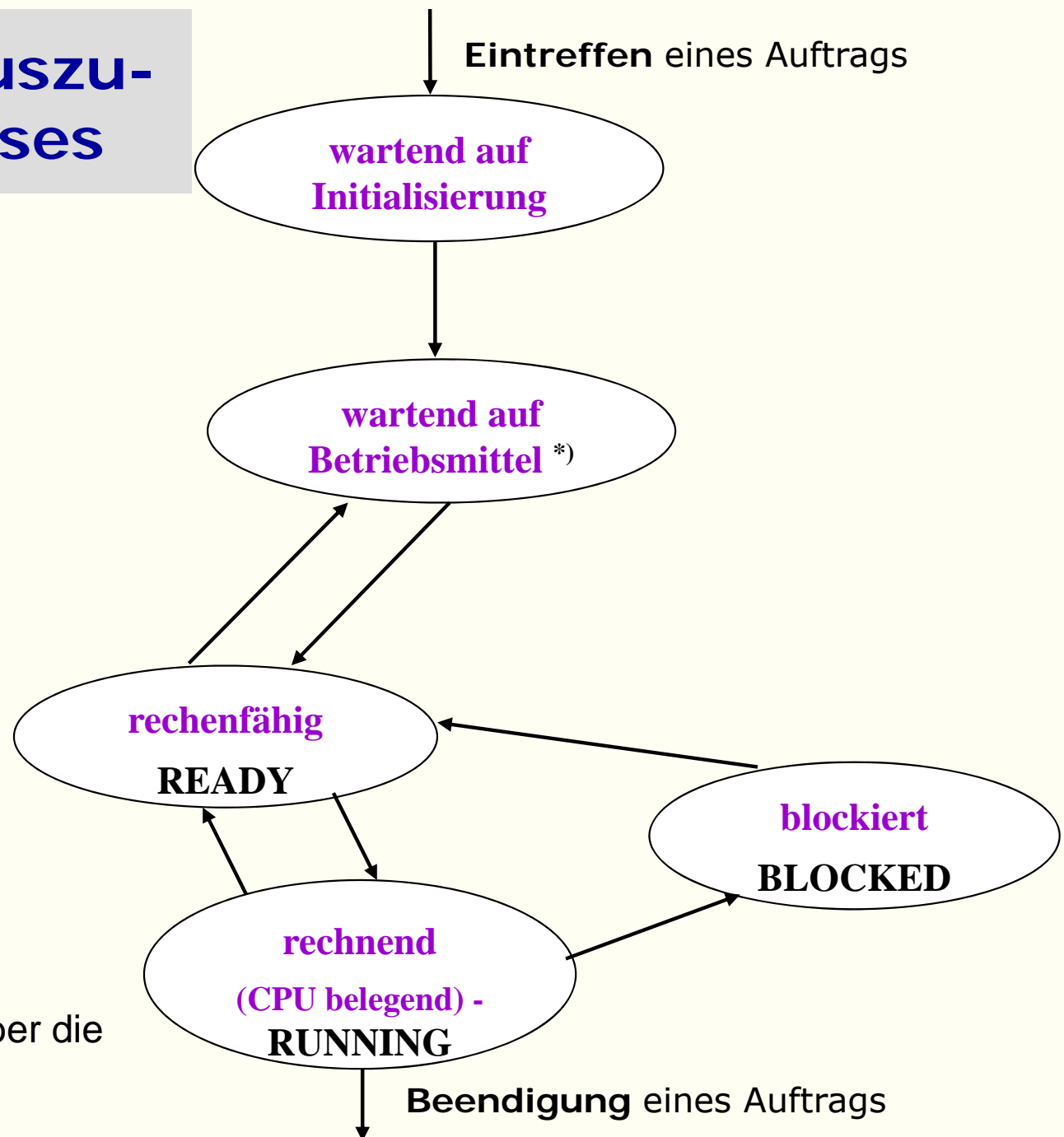


Ausführungsregel:

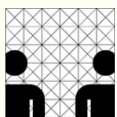
Jeder Prozess erwirbt von den zuständigen Verwaltern alle für den nächsten Arbeitsschritt benötigten Betriebsmittel; dann ordnet er sich in die Warteschlange für die "reine Rechenleistung" (Ready Queue) ein.



Zustände eines auszuführenden Prozesses



*) hier gemeint: Betriebsmittel über die CPU (Rechenzeit) hinaus



Beispiel zur kurzfristigen Planung ("Dispatching"):

Zeit	CPU belegt von	Warteschlange (Prozess, Rest-Rechenzeit)
0	P_1	$(P_1, 3)$
1	P_1	$(P_1, 2)$
2	P_1	$(P_1, 1); (P_2, 6)$
3	P_2	$(P_2, 6);$
4	P_2	$(P_2, 5); (P_3, 4)$
5	P_2	$(P_2, 4); (P_3, 4)$
6	P_2	$(P_2, 3); (P_3, 4); (P_4, 5)$
7	P_2	$(P_2, 2); (P_3, 4); (P_4, 5);$
8	P_2	$(P_2, 1); (P_3, 4); (P_4, 5); (P_5, 2)$
9	$P_{??}$	

Zum Zeitpunkt 9 gibt der Prozess P_2 die CPU frei.

Zur Auswahl des nächsten Prozesses, dem die CPU zugeteilt werden soll, stehen mehrere Kriterien zur Verfügung, z.B.

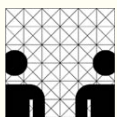
Auswahl des nächsten Prozesses nach

kürzester Restlaufzeit: P_5

längster Wartezeit: P_3

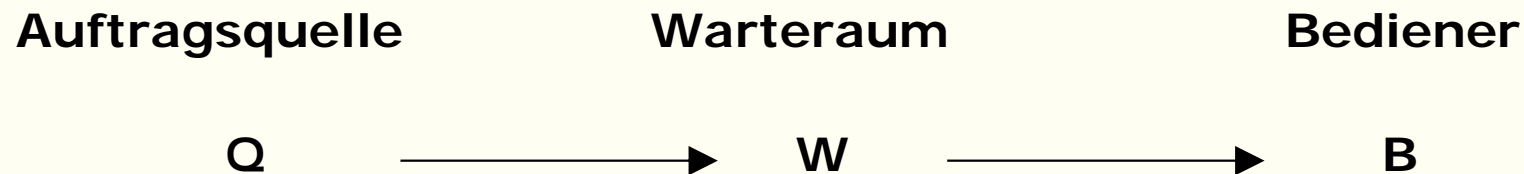
bester Bediengüte: P_3

wobei **Bediengüte** = $(\text{Wartezeit} + \text{Bedienzeit}) / \text{Bedienzeit}$



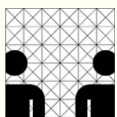
Prozessor-Scheduling

Situation:



Bemerkungen:

- Die Erzeugungszeitpunkte der Aufträge sind zufällig und somit i.d.R. a priori unbekannt.
- Der Bedienaufwand für die Auftragspopulation ist zufällig, wobei evtl. eine max. Bedienzeitanforderung spezifiziert sein kann.
- Scheduling-Entscheidungen werden getroffen
 - bei Einfügen eines Auftrags in den Wartezimmer,
 - bei Herausnahme eines Auftrags aus dem Wartezimmer und
 - bei „sonstigen markanten“ Ereignissen.
- Der Aufwand für das Prozessor-Scheduling soll gering sein.



B2.2 Scheduling ohne Echtzeitanforderungen

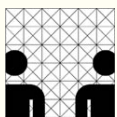
FIFO (First In First Out) - bzw. FCFS (First Come First Serve)-Scheduling:



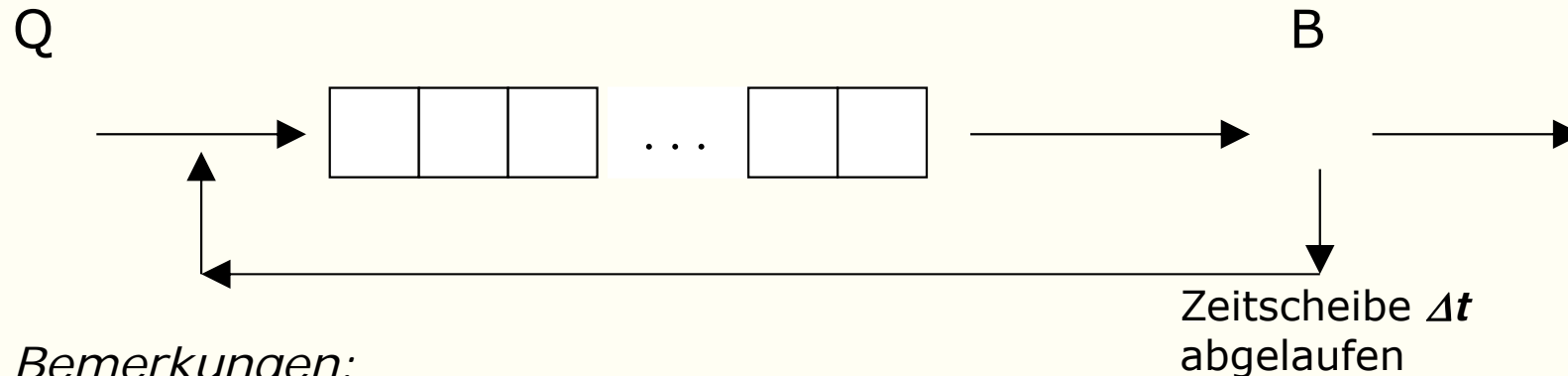
Bemerkungen:

- Alle Aufträge werden gleich behandelt.
- Die Verweilzeit eines Auftrags im System ist weitgehend unabhängig von seiner Bedienzeit, sofern $W \gg B$, d.h. mittl. Wartezeit \gg mittl. Bedienzeit.
- Sie errechnet aus dem „Gesetz von (John D.C.) Little“ *):
*Mittlere Zahl der Aufträge (N) im System = mittlere Verweilzeit eines Auftrags (T) im System * Ankunftsrate (λ) der Aufträge, d.h. $N = \lambda * T$*
- Das FIFO-Scheduling begünstigt den „Konvoi-Effekt“. Dies ist ein klarer Nachteil. Eine zeitweilige Bevorzugung von Kurzläufnern würde die verdeckte Ressourcennutzung mindern.
- Der Verwaltungsaufwand ist gering.

*) gilt unter sehr allgemeinen Randbedingungen für nahezu beliebige Bediensysteme

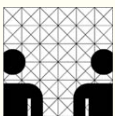


„Round Robin“ (= „Reigen“) - Scheduling



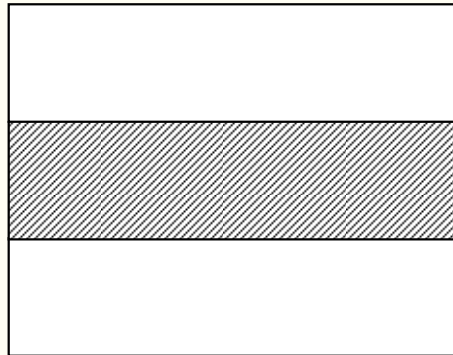
Bemerkungen:

- Die Verweilzeit eines Auftrags ist weitgehend proportional seiner Bedienzeit.
- Die Größe der Zeitscheibe bedingt den Verwaltungsaufwand.
- Kurze Aufträge sollten für ihre Ausführung nur eine einzige Zeitscheibe benötigen (Ziel: geringer Implementations-Overhead).
- Das Hauptproblem des „Round Robin“-Schedulings sind die Kosten des Prozesswechsels, z.B. kann ein Prozesswechsel dazu führen, dass ein Prozess auf den Peripheriespeicher ausgelagert wird und ein anderer von dort geladen wird (siehe „Paging“-Verfahren).
- Bei Wahl einer hinreichend kleinen Zeitscheibe (Δt) erfüllt das „Round Robin“-Scheduling eine kontextabhängige Vorstellung von Gleichbehandlung, wie in der Folge gezeigt wird (s.u.).

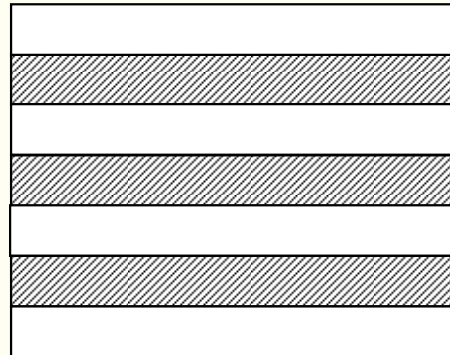


„Round Robin“: Eine Art von Gleichbehandlung

Jeder neu im System eintreffende Auftrag hat einen sofort zu erfüllenden Anspruch auf Rechenleistung.



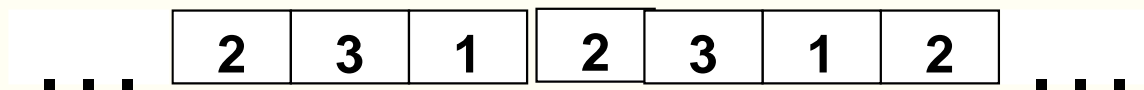
**Drei Aufträge teilen
sich die Rechenkapazität**



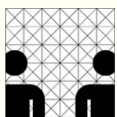
**Sieben Aufträge teilen
sich die Rechenkapazität**

Implementation über Zeitscheibenverfahren:

drei Aufträge :

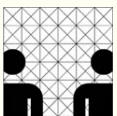


sieben Aufträge :



“Processor Sharing” (PS)

- “Processor Sharing” als Sonderfall von Round Robin, insbesondere für $\lim \Delta t \rightarrow 0$
- *Vorteile* von PS:
 - wäre sehr fair, da CPU hier völlig gleichmäßig auf bereite Prozesse (Zustand “READY”) aufgeteilt wird
 - stellt für manche Leistungs- und Fairnessbewertungen eine sinnvolle “best case“-Abschätzung dar.
- *Nachteile* von PS:
 - ist nicht implementierbar, da Implementations-Overhead für $\Delta t \rightarrow 0$ gegen unendlich strebt
 - ist daher auch nicht (!) praxisrelevant.

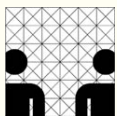
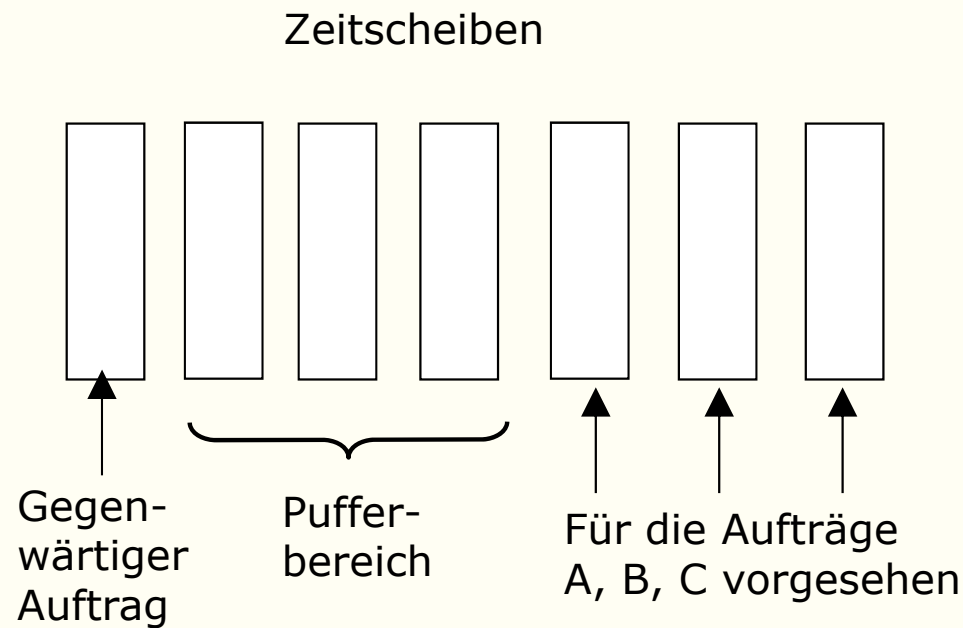


Ein Vorschlag zur Minderung von Prozesswechseln in Teilnehmersystemen:

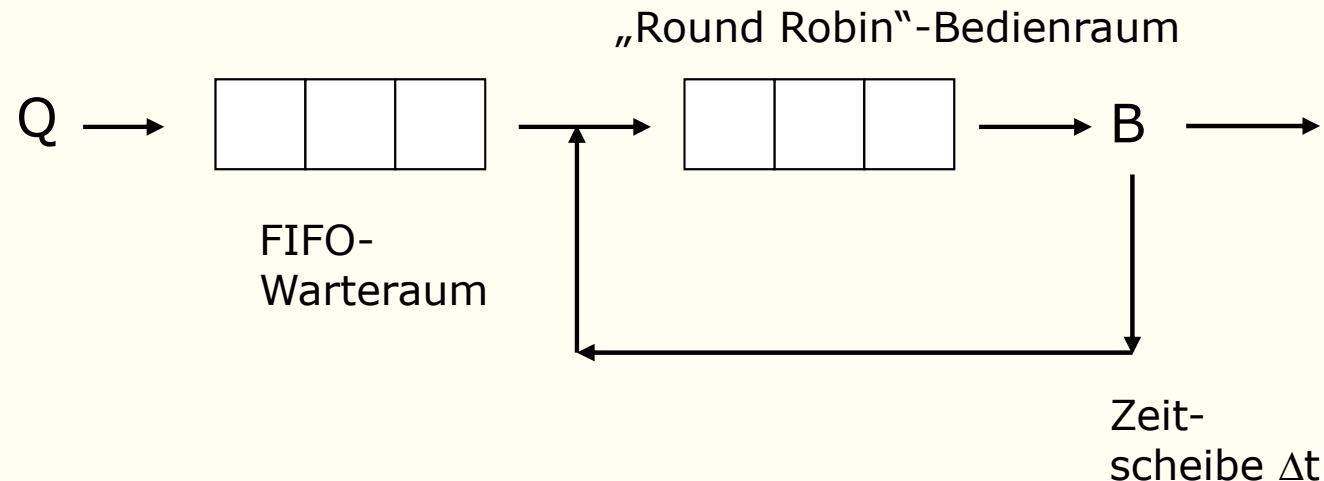
Grundforderung:

Jedem Prozess wird ein Minimum an Rechenzeit innerhalb eines Zeitintervalls garantiert.

Belegung der Zeitscheiben nur nach Minimalerfordernis:

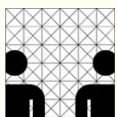


Mischform zwischen „Round Robin“- und FIFO:

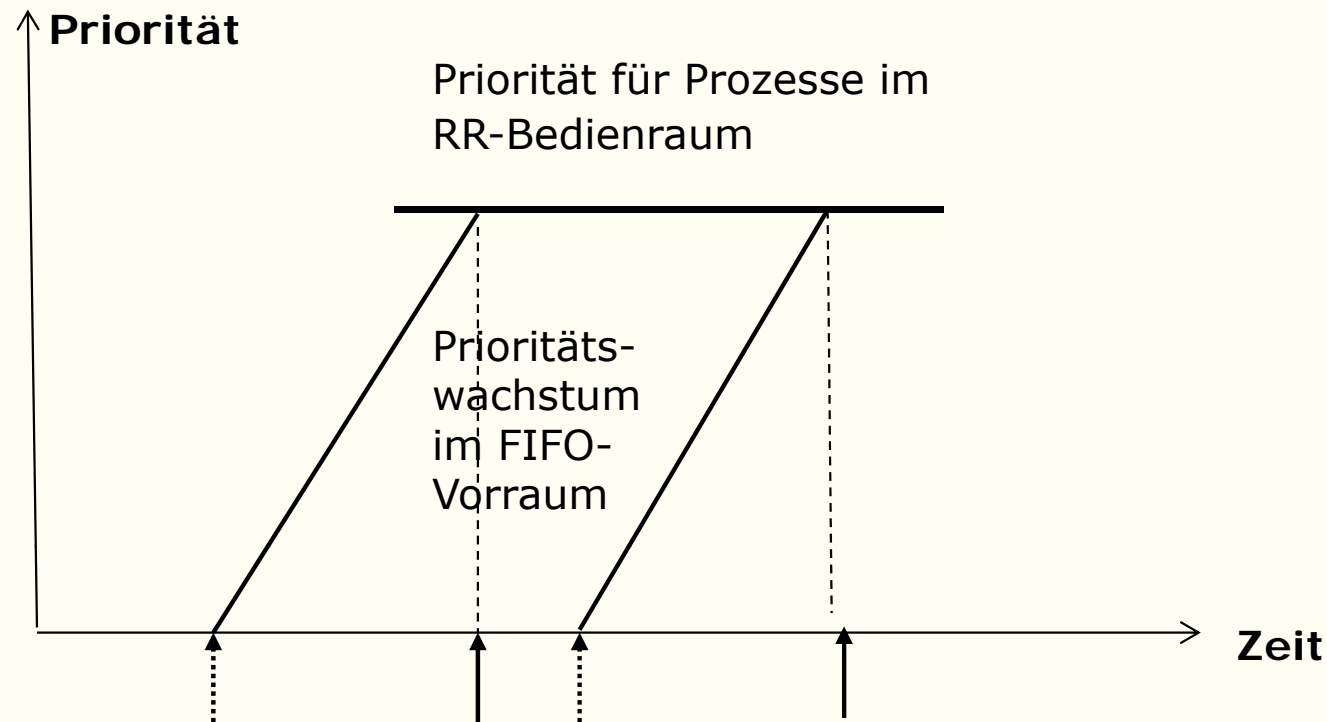


Bemerkungen:

- Jeder Auftrag hat eine Priorität.
- Alle Aufträge im „Round Robin“(RR)-Raum haben die gleiche Priorität.
- Die Prioritäten aller Aufträge im Wartezimmer wachsen mit der Zeit.
- Ein Auftrag tritt vom FIFO-Wartezimmer in den Bedienraum, falls seine Priorität gleich derjenigen der bedienten Aufträge ist.



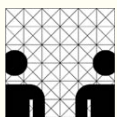
Verwendung zeitlinearer Prioritätsfunktionen



↑ Ankunftszeitpunkte neuer Prozesse

↑ Übertrittszeitpunkte in den RR-Raum; diese sind nur einmal zu berechnen.

Bemerkung: Einfache Adaptionen, wie z.B. für leeren RR-Raum sollten vollzogen werden

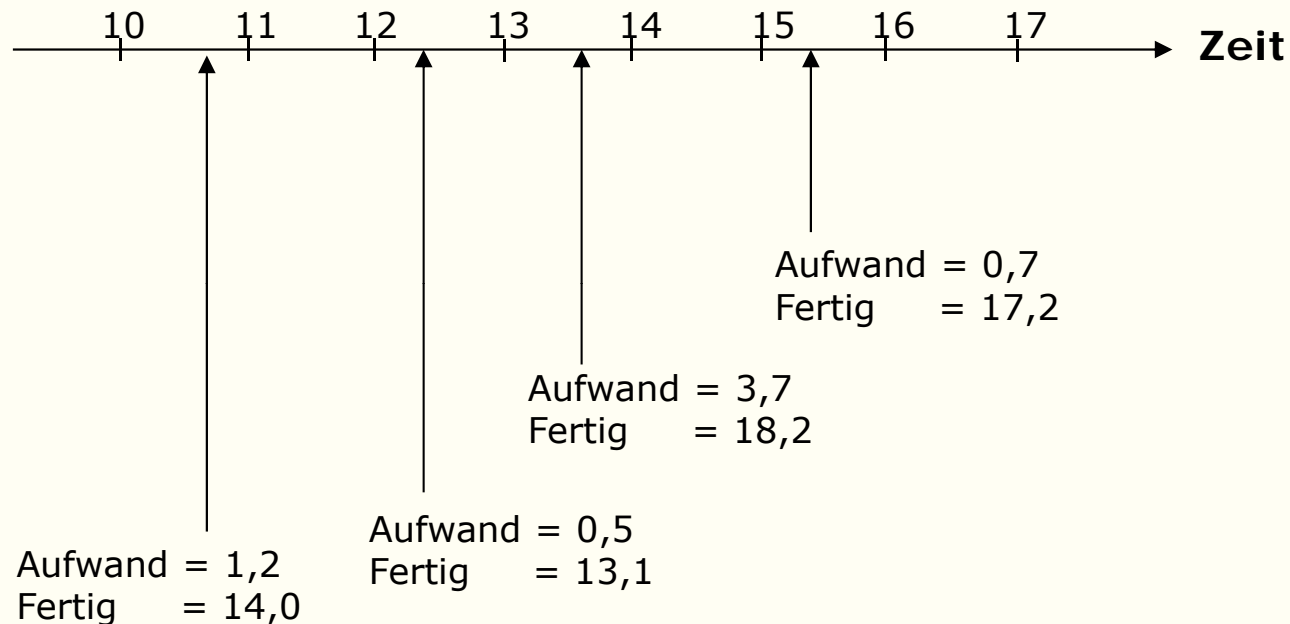


B2.3 Scheduling bei Echtzeitanforderungen

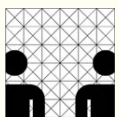
Veranschaulichung des Terminbetriebs:

Situation: Zu unbestimmten Zeitpunkten treffen Aufträge ein, die bis zu einem jeweils vorgegebenen Zeitpunkt ausgeführt sein müssen. Die Rechenanlage ist so dimensioniert, dass sie die Gesamtlast bewältigen kann.

Beispiel:



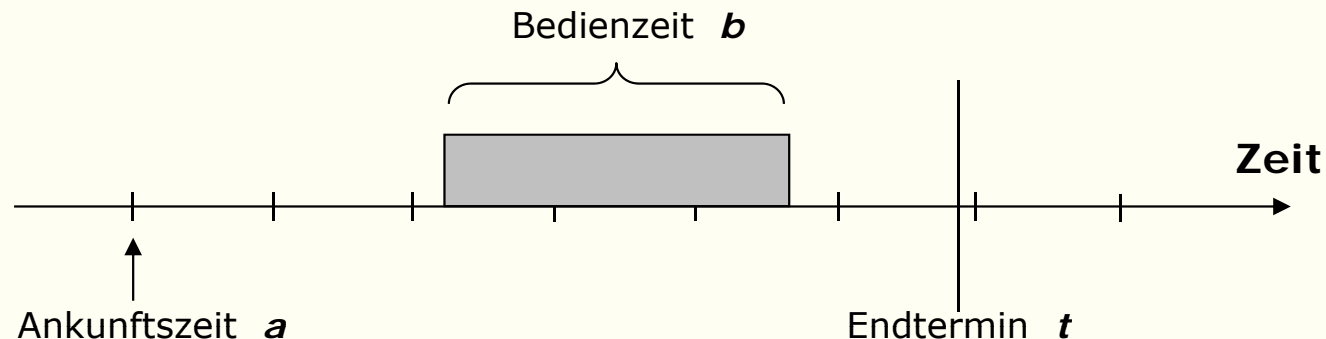
Bemerkung: Um statistische Schwankungen ausgleichen zu können, sollte die Auftragschar nicht 100% der Rechenleistung beanspruchen, d.h. CPU-Auslastung $\rho_{\text{CPU}} < 1$ wird vorausgesetzt.



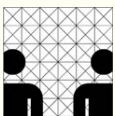
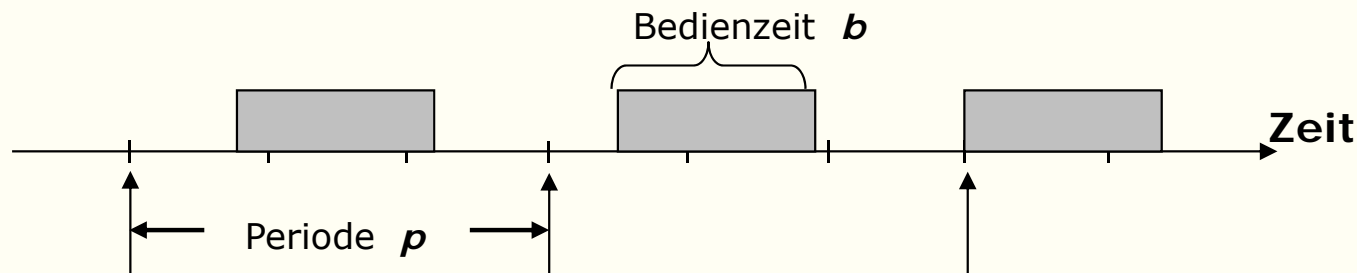
Beschreibung von Aufträgen mit Echtzeitanforderungen

Jeder Prozess ist gekennzeichnet durch drei Angaben :



- den Zeitpunkt der Ankunft a ,
- die Bedienzeit b und
- den Endtermin t (engl.: *Deadline*).



Zur theoretischen Behandlung wählt man zyklische Prozesse mit Ankunftsperiode p (auch: Zwischenankunftszeit). Damit reduziert man die Zahl der Auftragsparameter auf zwei, die *Periode* p und die *Bedienzeit* b .

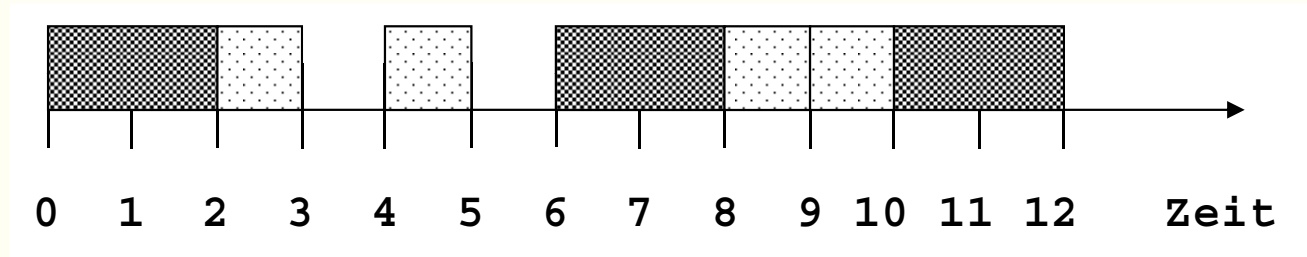


Beispiel: 2 unabhängige, periodische Aufträge

	Periode	Bedienzeit	Symbol
A1	4	2	
A2	3	1	

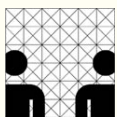
Bemerkung: Alle Aufträge sind zum Zeitpunkt 0 lauffähig, dies ist der ungünstigste Fall.

Ein Schedule heißt **gültig**, falls alle Endtermine eingehalten werden.
Der folgende Schedule ist gültig :



Bemerkungen:

- Da 12 das kleinste gemeinsame Vielfache von 3 und 4 ist, ist der obige Schedule für beliebige Zeiträume gültig.
- Der Beispiel-Schedule zeigt auch die großen Freiheiten, die ein Schedule-Ersteller hat.




Terminbetrieb: Randbedingungen – Fall 1

Randbedingungen bei der Behandlung des Terminbetriebs:

- Es existiert eine feste Zahl unabhängiger Aufträge.
- Die Bedienzeiten der einzelnen Aufträge sind konstant.
- Die einzelnen Aufträge werden periodisch aufgerufen.
- Die Aufrufintervalle sind fest.
- Jeder Auftrag muss innerhalb seines Aufrufintervalls abgearbeitet sein.

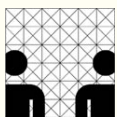
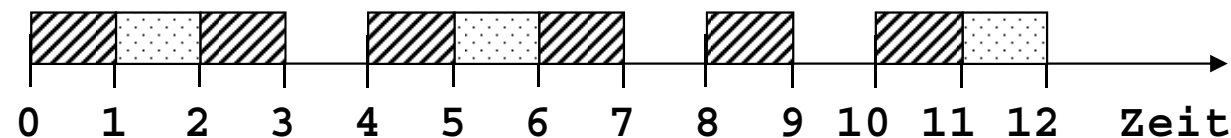
Beispiel zum Scheduling nach festen **Prioritäten**:

Gegeben: 2 periodische Aufträge A1, A2

	Periode	Bedienzeit	Symbol
A1	2	1	
A2	5	1	

Fall 1: Priorität (A1) > Priorität (A2) (A1 "vor" A2)

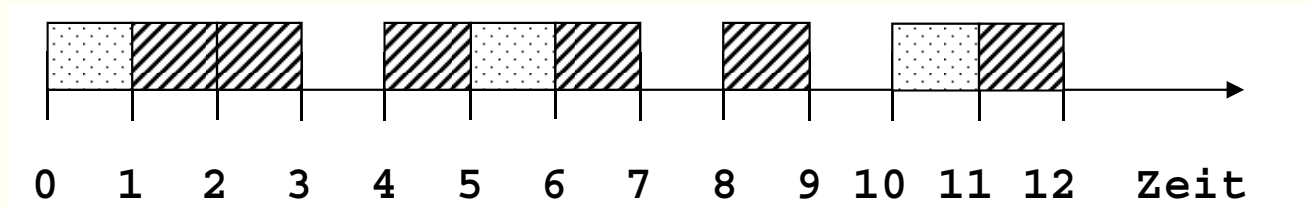
Schedule :



Beispiel zum Scheduling nach festen Prioritäten: Fall 2

Fall 2: Priorität (A2) > Priorität (A1) (A2 "vor" A1)

Schedule :



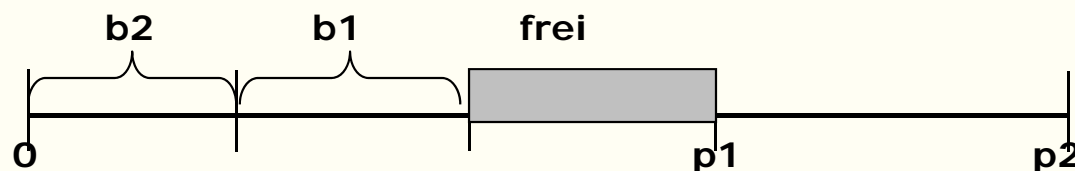
Bemerkungen:

- Bei Scheduling nach festen Prioritäten erhält man einen optimalen Schedule, falls man die Prioritäten nach fallenden Perioden vergibt.

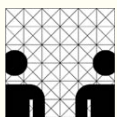
Graphische Illustration:

Gegeben seien 2 zum Zeitpunkt 0 lauffähige Aufträge A1(p_1 , b_1) und A2(p_2 , b_2) mit $p_1 < p_2$;

A2 habe die größere Priorität. Es muss gelten: $b_2 + b_1 \leq p_1$.



Eine andere Anordnung der Prozesse im Intervall $[0, p_1)$ ändert nichts Wesentliches!



Bemerkungen zu den Fallbeispielen (Forts.)

Bemerkungen:

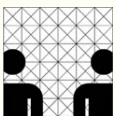
- In ihrer Arbeit "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", J. ACM 20, 1 (1973) bewiesen Liu und Layland die folgende Aussage: *Die kleinste obere Schranke der Prozessor-(CPU-) Auslastung U bei Scheduling nach festen Prioritäten ist bei m Aufträgen:*

- $$U = m * (2^{1/m} - 1)$$

$$\text{CPU-Auslastung } U = \sum_{i=1}^m \frac{b_i}{p_i} \leq 1$$

$$\lim_{m \rightarrow \infty} m * (2^{1/m} - 1) = \ln 2$$

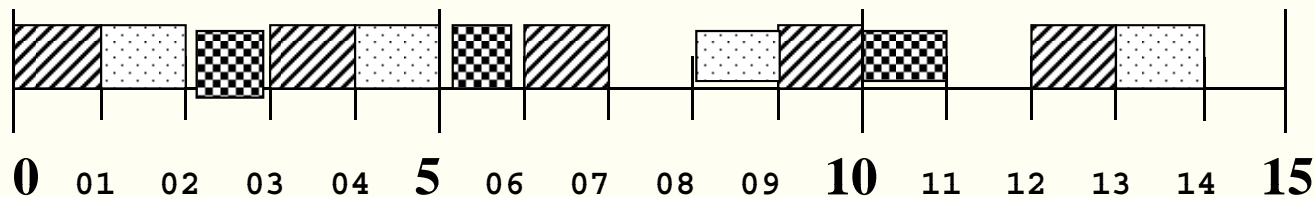
- Da $\ln 2 \approx 0,693$, muss in Extremfällen bei Scheduling nach festen Prioritäten auf etwa 30% der Rechenleistung verzichtet werden. Andererseits erfordert ein Scheduling nach festen Prioritäten kaum Verwaltungsaufwand.



Beispiel: Echtzeit-Scheduling für 3 Aufträge („rate monotonic“)

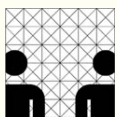
	Periode	Bedienzeit	Symbol
A1	3	1	
A2	4	1	
A3	5	×	

Schaubild eines „rate monotonic“ Schedules:



Beobachtungen:

- Der maximale Wert für die Bedienzeit des Auftrags 3 ist 1. Dies entnimmt man dem Zeitintervall von 2 bis 3. Die Zeitintervalle 7 bis 8, 11 bis 12 und 14 bis 15 sind wegen der Monotonie-Forderung nicht nutzbar.
- Setzt man die Bedienzeit für Auftrag 3 auf 1, dann erhält man eine Nutzung von $1/3 + 1/4 + 1/5 \approx 0,78$, dies entspricht in etwa $3 \cdot (2^{1/3} - 1) \approx 0,78$.
- Wählt man als Scheduling-Kriterium die Nähe zum Endtermin, dann lässt sich die Bedienzeit des Auftrags 3 auf 2 erhöhen.



A1, A2, A3, A3, A1, A2, A1, A3, A3, A1, A2, A3, A3, A1, A2,...

Scheduling nach dem nächsten Termin (*"earliest deadline"*)


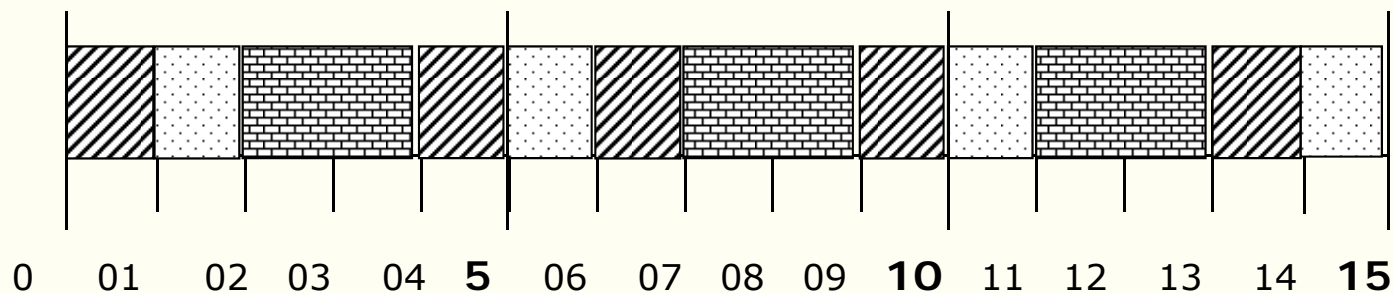
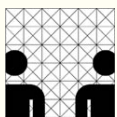
	Periode	Bedienzeit	Symbol
A1	3	1	
A2	4	1	
A3	5	2 !!	

Schaubild eines *"earliest deadline"* Schedules:



Bemerkungen:


- Im Beispiel beträgt die Rechnernutzung $1/3 + 1/4 + 2/5 \approx 0,98$.
- Der Beispiel-Schedule kommt ohne Verdrängung aus.
- Es lässt sich zeigen, dass für "earliest deadline"-Algorithmen die Zahl der Verdrängungen sich durch die Zahl der Aufträge beschränken lässt. Die Kontext-Wechsel-Zeiten können somit in den Laufzeiten der Aufträge berücksichtigt werden .



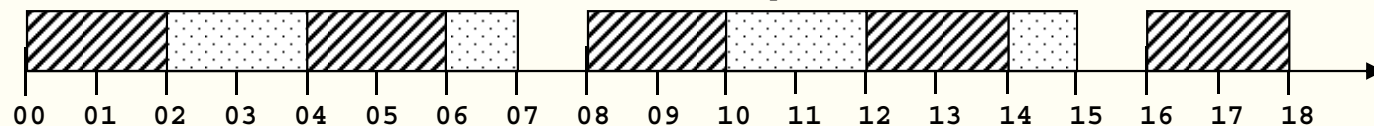
Bemerkungen zu den Fallbeispielen (Forts.)

- Es existiert genau dann ein "earliest deadline" Schedule, falls für die CPU-Auslastung gilt:
$$U = \sum_{i=1}^m \frac{b_i}{p_i} \leq 1$$
- Scheduling nach dem nächsten Termin ist für Monoprozessoren optimal.

Das folgende **Beispiel** demonstriert, dass "earliest deadline" Schedules tendenziell weniger Verdrängungen aufweisen als "rate monotonic" Schedules.

	Periode	Laufzeit	Symbol
A1	4	2	
A2	8	3	

- **"rate monotonic"-Schedule:** pro 8 Zeiteinheiten eine Verdrängung.



- **"earliest deadline"-Schedule:** keine Verdrängung notwendig.

