

# Scheduling

---

## Betriebssysteme I WS 2006/2006



**Jörg Kaiser**  
**IVS – EOS**

**Otto-von-Guericke-Universität Magdeburg**

# Scheduling - Beispiele

---

**Tagesplan → Ressource: Zeit**

**Stundenpläne → Ressource: Räume, Dozenten**

**Car Sharing → Ressourcen: Autos**

**Flughafenüberwachung → Ressource: Luftraum, Landebahn(en)**

**Badezimmer in WGs → Ressource - obvious**



# Scheduling - Ablaufplanung

---

Durch **Scheduling** wird eine Reihenfolge für die Ausführung von Prozessen festgelegt. Die Kriterien für die Reihenfolge können an verschiedenen Strategien orientiert sein.

Der **Scheduler** verwaltet die Ressource "Prozessor".

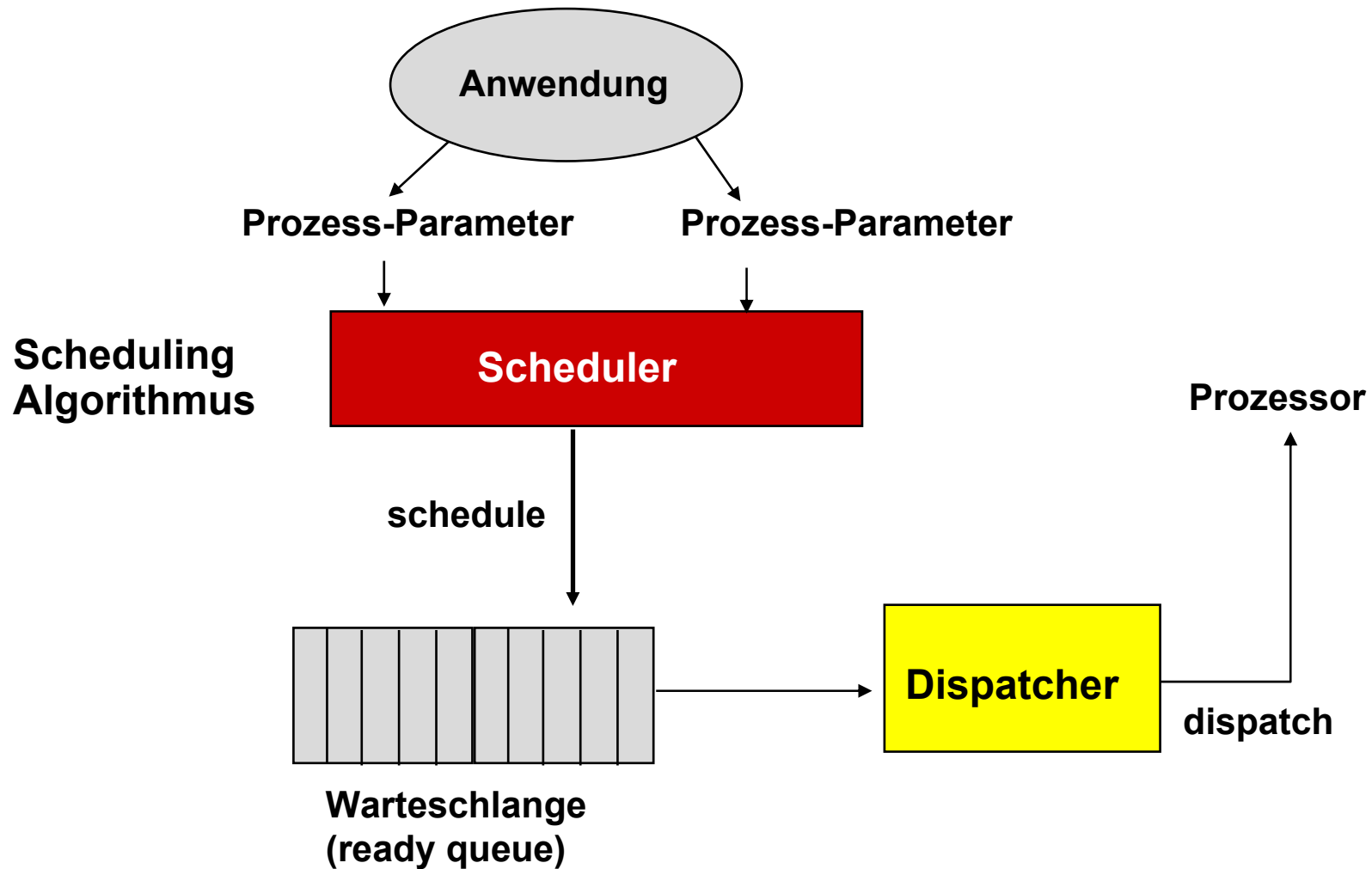
Prozesse werden vom Scheduler in eine oder mehrere **Warteschlangen** eingeordnet.

Der **Dispatcher** teilt den Prozess im Kopf der aktuellen Warteschlange dem Prozessor zu.

**Scheduling folgt einer Strategie, Dispatching ist ein Mechanismus.**



# Scheduling und Dispatching



# Scheduling - Beispiele

---

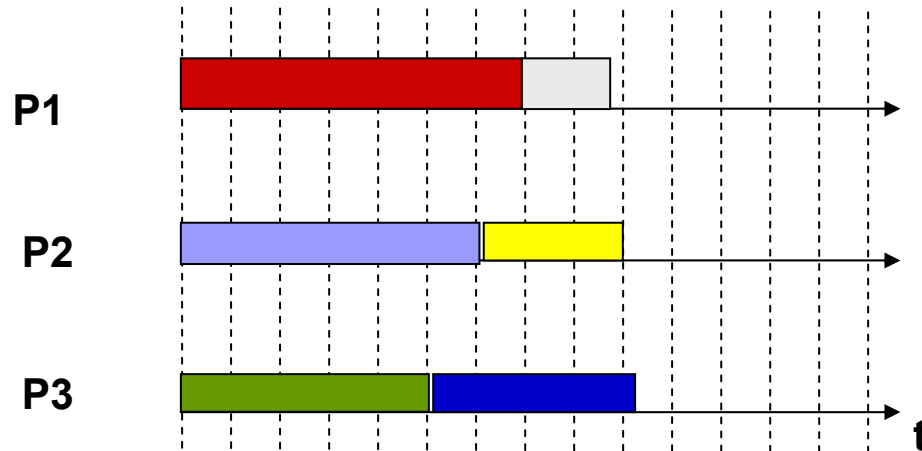
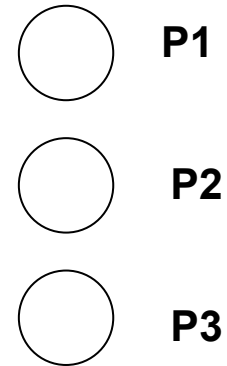
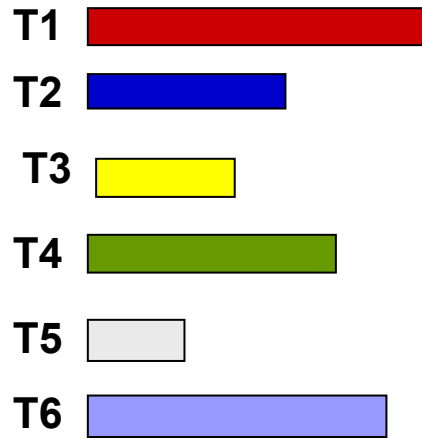
**Mehrbenutzersystem:** Mehrere Prozesse sind ablauffähig in der Bereitliste: Welcher soll als nächster dem Prozessor zugeordnet werden ?

**Einbenutzersystem:** Abspielen eines Videostroms live aus dem Internet. Das Zusammenspiel von Netzwerk-Software, Decodierung, Bildschirmausgabe und Tonausgabe soll zu einer gleichmäßigen, synchronisierten Wiedergabe führen, auch wenn im Hintergrund zusätzlich ein Compiler läuft.

**Steuersystem:** Ein autonomer Roboter muss verschiedene Sensordaten in unterschiedlichen Raten abtasten und darauf reagieren. Z.B. müssen zur Erkennung und Umfahrung von Hindernissen auf einem geplanten Weg zeitaufwendige Berechnungen durchgeführt werden. Gleichzeitig müssen Navigationsaufgaben und sicherheitskritische Aufgaben bearbeitet werden.



# Allgemeines Schedulingproblem



# Entwurfsraum für einen Scheduler

---

- ➔ **Einprozessor / Mehrprozessor?**
- ➔ **Prozessmenge statisch oder dynamisch ?**
- ➔ **Scheduling on-line (zur Laufzeit) oder off-line (vorher)?**
- ➔ **Ausführungszeiten bekannt ?**
- ➔ **Verdrängung möglich?**
- ➔ **Abhängigkeiten zwischen den Prozessen?**
- ➔ **Prioritäten zu berücksichtigen?**
- ➔ **Zeitliche Vorgaben (z.B. Deadlines) zu berücksichtigen?**
- ➔ **Welches Ziel soll erreicht werden?**



# Entwurfsraum für einen Scheduler

---

- ➔ **Kooperativ vs. Unterbrechend**
- ➔ **Deterministisch vs. Probabilistisch**
- ➔ **Offline vs. Online**

## **Betriebsartenabhängigkeit:**

### **General Purpose Betrieb:**

Durchsetzung (der Strategie); Gerechtigkeit, Lastausgleich

### **Batch Betrieb:**

Durchsatz, Durchlaufzeit, Prozessorauslastung

**Interaktiver Betrieb:** Antwortzeit, Proportionalität, Benutzererwartungen

**Echtzeitbetrieb:** Dringlichkeit, Termineinhaltung, Vorhersagbarkeit  
(Widerspruch zu Gerechtigkeit und Lastausgleich)





# Entwurfsraum für einen Scheduler (Ziele)

---

## Benutzerorientierte Ziele

Minimale Länge des Ablaufplans

Minimale maximale Antwortzeit

Minimale mittlere (gewichtete) Antwortzeit

Minimale maximale Verspätung

Minimale Anzahl verspäteter Prozesse

...

## Systemorientierte Ziele

Maximaler Durchsatz

Maximale Prozessorauslastung

....

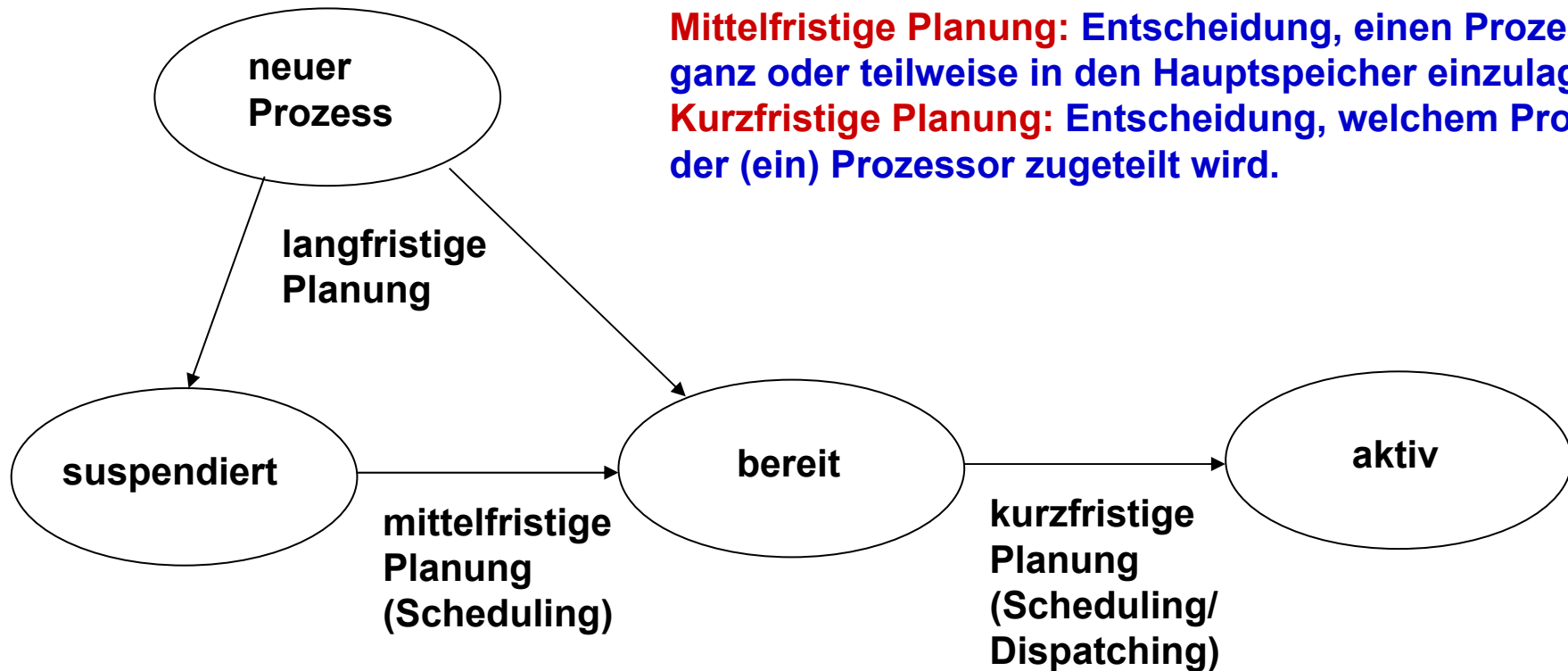


# Arten des Scheduling

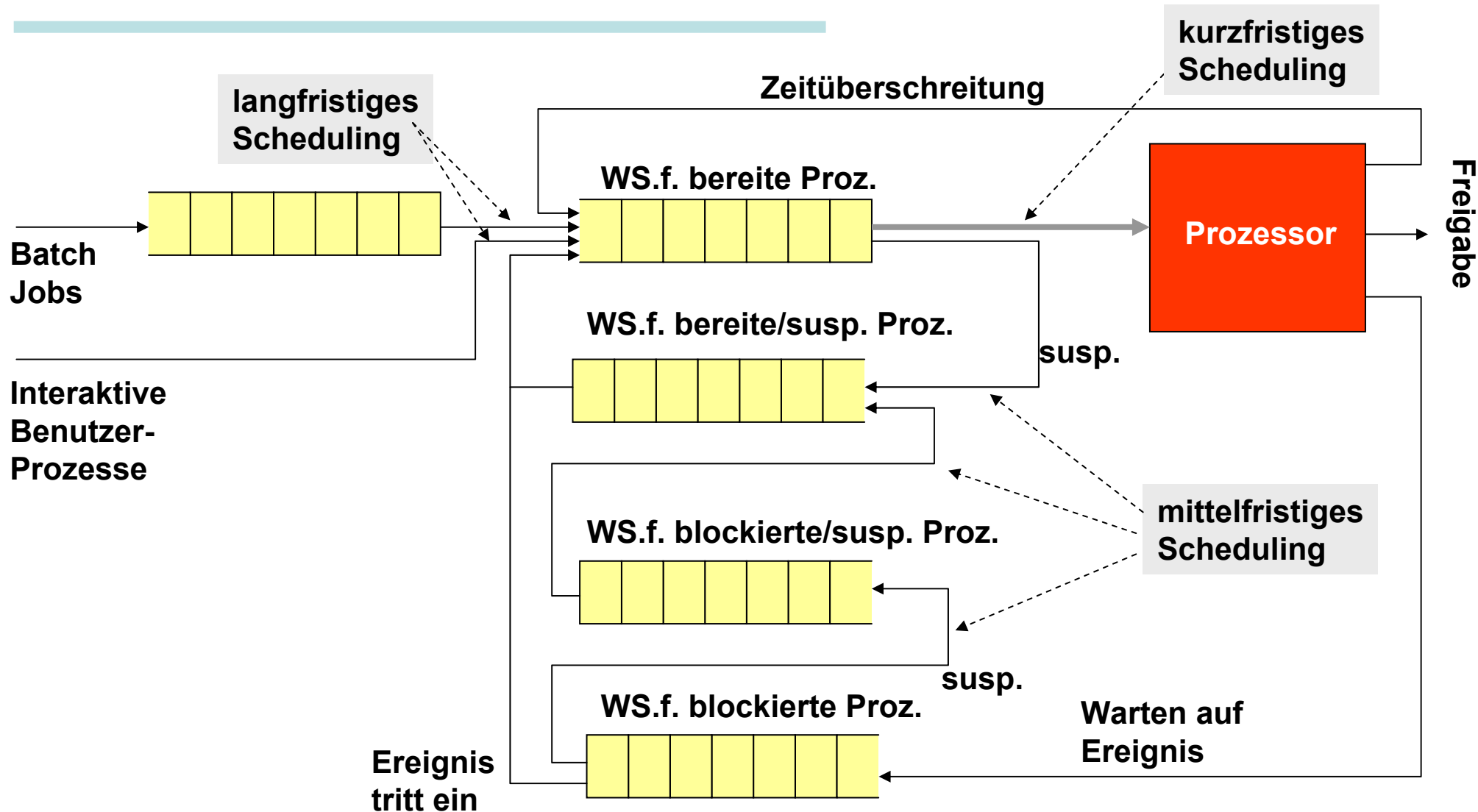
**Langfristige Planung:** Entscheidung, ob ein Prozess in die Menge der auszuführenden Prozesse aufgenommen wird.

**Mittelfristige Planung:** Entscheidung, einen Prozess ganz oder teilweise in den Hauptspeicher einzulagern.

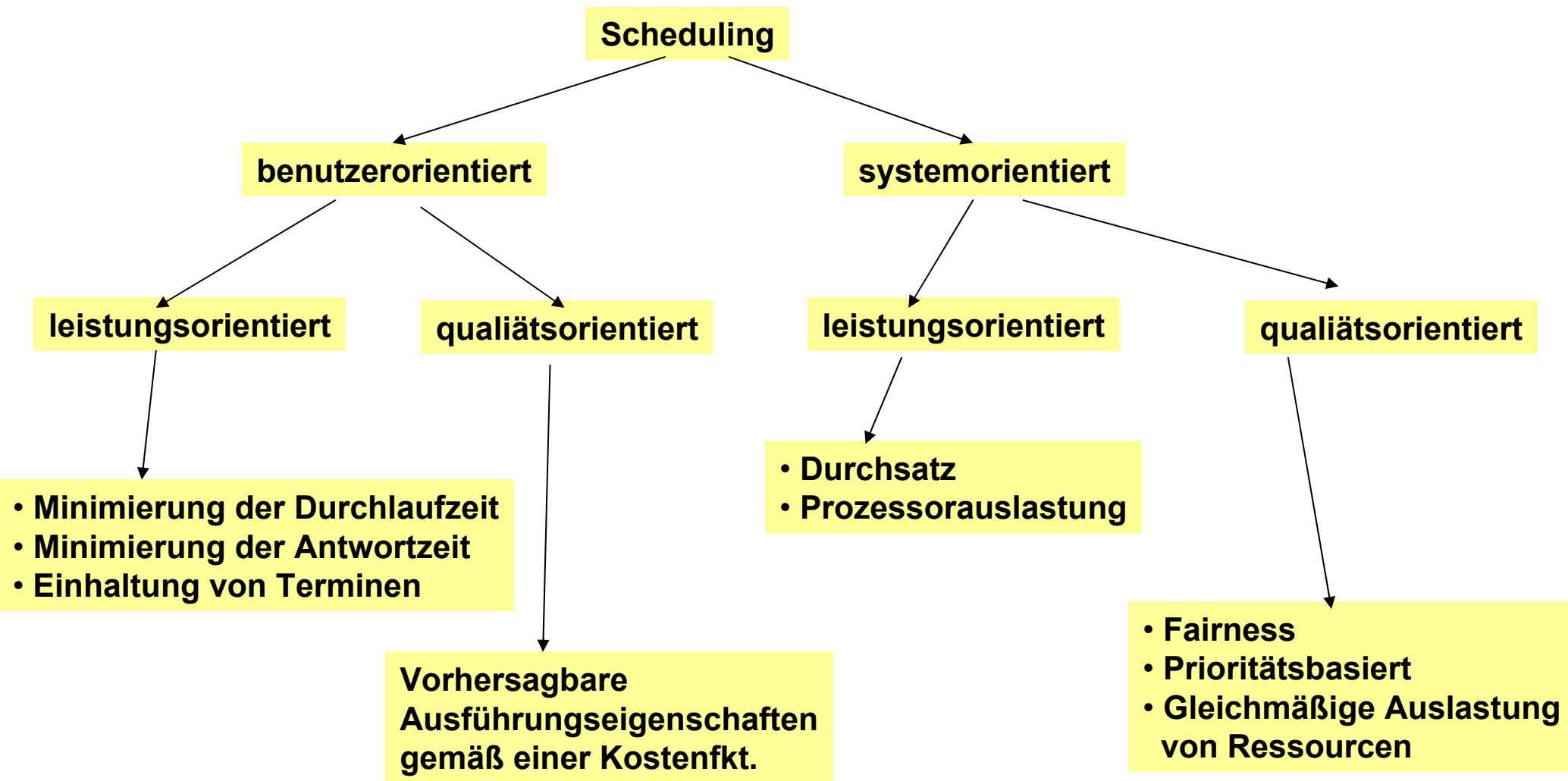
**Kurzfristige Planung:** Entscheidung, welchem Prozess der (ein) Prozessor zugeteilt wird.



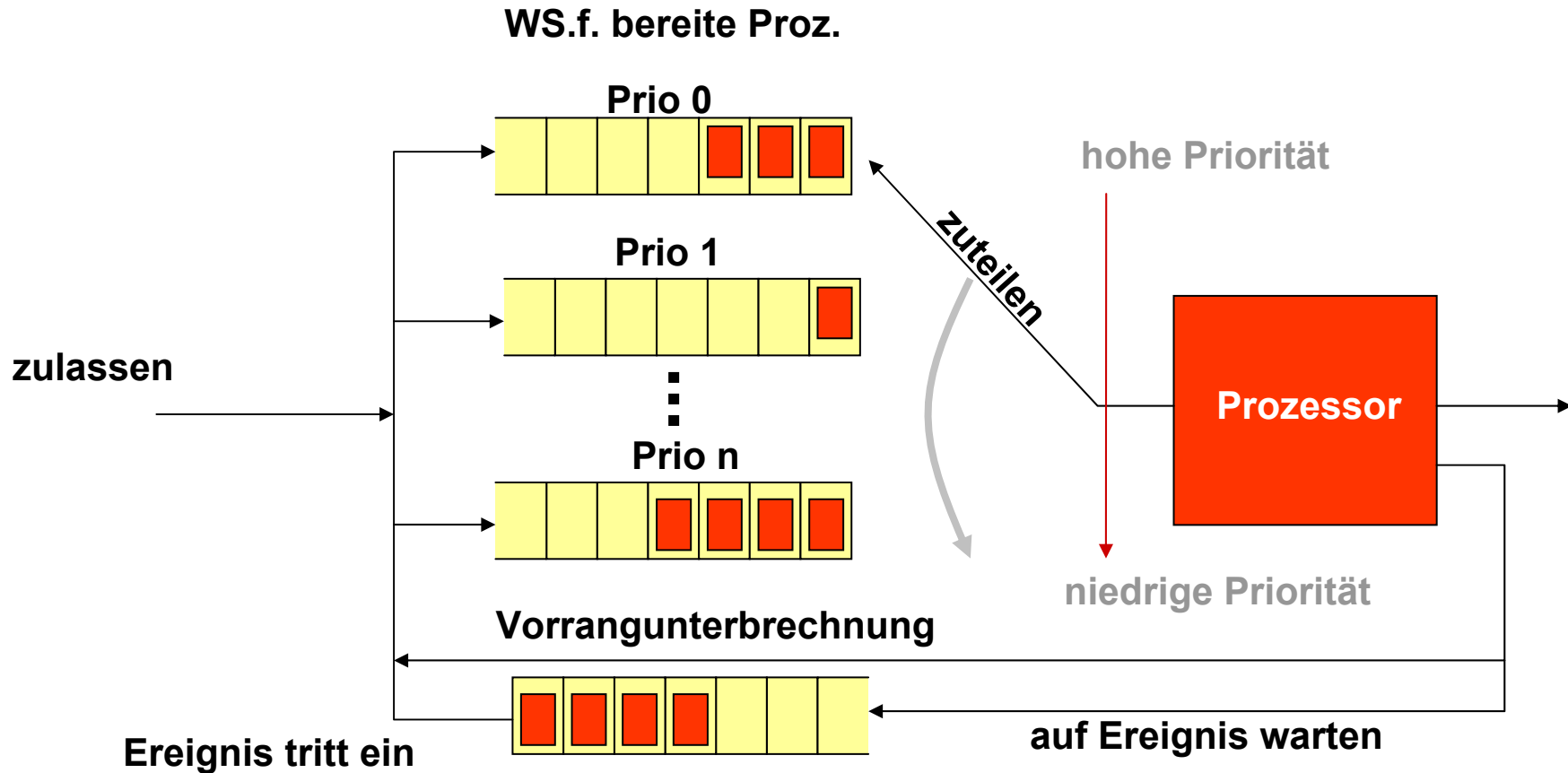
# Warteschlangen (Queues)



# Ziele des Scheduling



# Warteschlangen mit Prioritäten

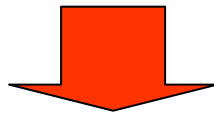


# Wann wird der Scheduler aktiviert?

---

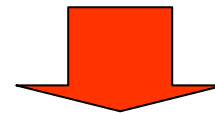
## Aktivierung des Schedulers

Zeitgesteuert



- Ablauf einer Zeitscheibe
- Zeitüberschreitung
- festgelegter Zeitpunkt

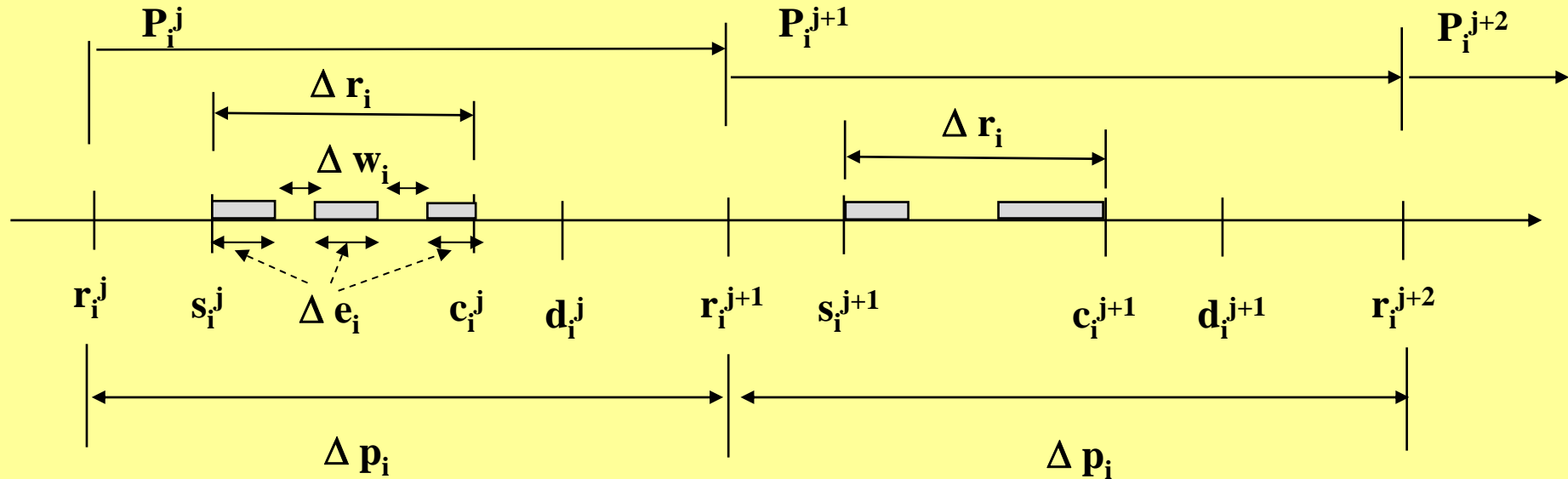
Ereignisgesteuert



- Expliziter Aufruf
- Impliziter Aufruf (Abgabe d. Kontr.)
- Unterbrechung
- Systemaufruf (z.B. E/A)
- Anderer Prozess wird bereit



# Schedulingparameter



## zeitliche Parameter:

$r_i^j$ : Bereitzeit/Ankunftszeit (ready time)

$s_i^j$ : Startzeit (start time)

$c_i^j$ : Abschlusszeit (completion time)

$d_i^j$ : Frist (deadline)

$\Delta e_i$ : Ausführungszeit des Prozesses (execution time)

$\Delta p_i$ : Periode des Prozesses

$\Delta w_i$ : Wartezeit des Prozesses

$\Delta r_i$ : Durchlaufzeit des Prozesses  
(response time) =  $\Delta e_i + \Delta w_i$

## weitere Parameter:

Priorität

Zeitquantum

$P_i^j$ :  $j$ -te Ausführung des Prozesses  $i$



# Nach welchen Kriterien kann geplant werden?

---

- ➔ **Längste Wartezeit**
- ➔ **Feste Nutzungszeit**
- ➔ **Gesamtausführungszeit**
- ➔ **Verbleibende Ausführungszeit**
- ➔ **Mittlere Antwortzeit**
- ➔ **Ausführungsgeschichte**
- ➔ **Dringlichkeit**
- ➔ **Wichtigkeit**



# Standard Schedulingstrategien

---

- ★ **FCFS - First Come First Served:** Der Scheduler wählt den Prozess aus, der die früheste Ankunftszeit hat und demzufolge schon am längsten wartet.
- ★ **RR- Round Robin:** Das Scheduling ist in "Runden" organisiert. In jeder Runde steht jedem Prozess ein festes Zeitintervall (Zeitscheibe, engl. slot) zur Verfügung. Nach Ablauf der entsprechenden Zeit wird zum nächsten Prozess umgeschaltet. Jeder Prozess erhält die gleiche Zuteilung von Prozessorzeit.
- ★ **SPN - Shortest Process Next:** Der Scheduler wählt den Prozess aus, dessen erwartete Ausführungszeit am kürzesten ist. Dieser Prozess wird nicht von höherrangigen Prozessen unterbrochen.
- ★ **SRT - Shortest Remaining Time first:** Es wird der Prozess mit der kürzesten noch verbleibenden Ausführungszeit ausgewählt. Falls ein Prozess mit einer kürzeren Zeit bereit wird, wird der ausführende Prozess unterbrochen und der Prozessor dem neuen Prozess zugeteilt.
- ★ **HRRN - Highest Response Ratio Next:** Der Scheduler wählt Prozesse aufgrund der normalisierten Durchlaufzeit aus.
- ★ **Feedback:** Es werden mehrere Warteschlangen eingerichtet, in die Prozesse aufgrund ihrer Ausführungsgeschichte und anderer Kriterien eingeordnet werden.



# Standard Schedulingstrategien

---

**Beispiel:**

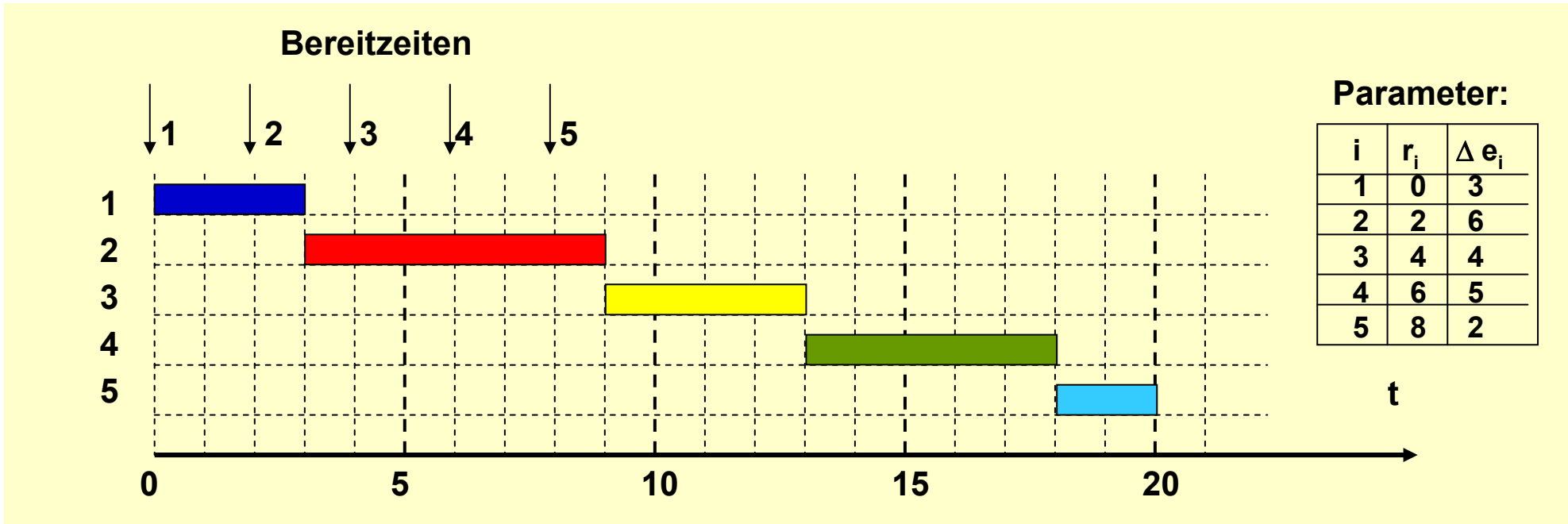
<b>P-Nr.</b>	<b>Bereitzeit <math>r_i</math></b>	<b>Ausführungszeit <math>\Delta e_i</math></b>
<b>1</b>	<b>0</b>	<b>3</b>
<b>2</b>	<b>2</b>	<b>6</b>
<b>3</b>	<b>4</b>	<b>4</b>
<b>4</b>	<b>6</b>	<b>5</b>
<b>5</b>	<b>8</b>	<b>2</b>

**P-Nr.:** Prozessnummer



# FCFS - First Come First Served

Der Scheduler wählt den Prozess aus, der die früheste Ankunftszeit hat und demzufolge schon am längsten wartet.



Prozess #	1	2	3	4	5	Mittelwert:	normalisierte Durchlaufzeit
Abschlusszeit	3	9	13	18	20		
Durchlaufzeit	3	7	9	12	12	8,60	
$\Delta r_i / \Delta e_i$	1,00	1,17	2,25	2,40	6,00	2,56	

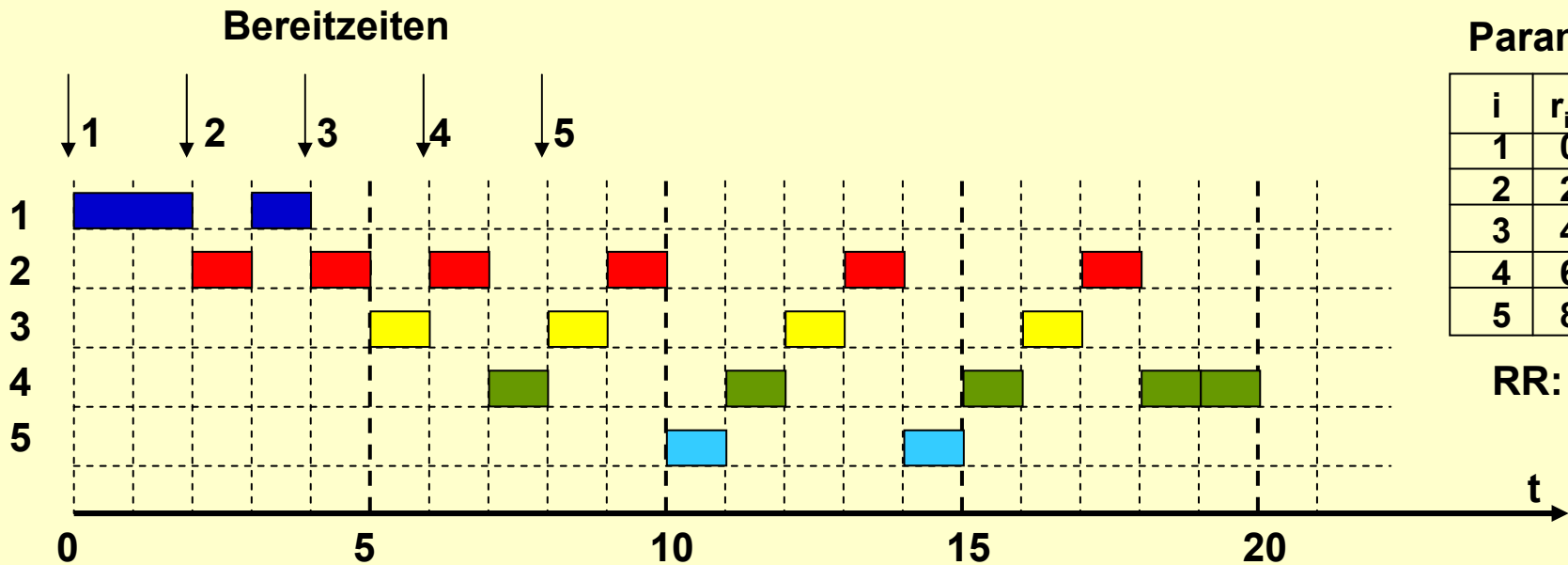


# Beispiel für FCFS

Prozess	Bereitzeit	Bearbeitungszeit	Startzeit	Abschluss-	Durch-	Norm.
	$r_k$	$\Delta e_i$	$s_k$	$c_k$	laufzeit	D-Zeit
					$\Delta r_i$	$\Delta r_i / \Delta e_i$
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1,99
Mittelwert					100	26

# RR - Round Robin

Das Scheduling ist in "Runden" organisiert. In jeder Runde steht jedem Prozess ein festes Zeitintervall  $q$  (Zeitscheibe, engl. slot) zur Verfügung. Nach Ablauf der entsprechenden Zeit wird zum nächsten Prozess umgeschaltet.



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	4	18	17	20	15	
Durchlaufzeit	4	16	13	14	7	10,80
$\Delta r_i / \Delta e_i$	1,33	2,67	3,25	2,80	3,50	2,71

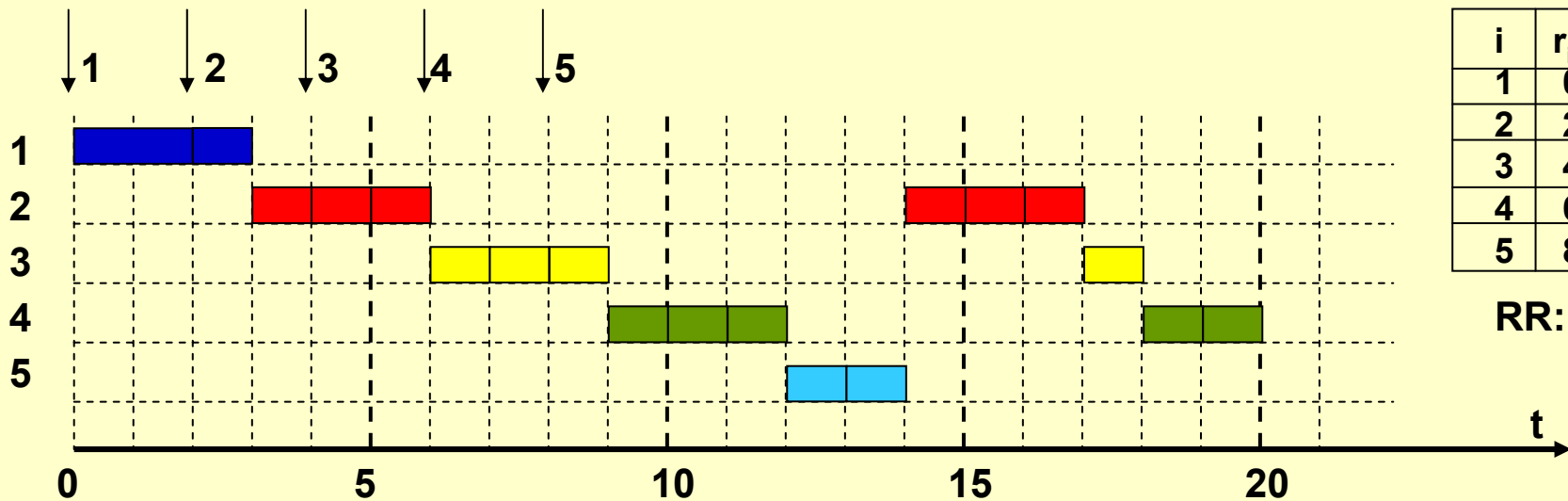
normalisierte Durchlaufzeit



# RR - Round Robin

Das Scheduling ist in "Runden" organisiert. In jeder Runde steht jedem Prozess ein festes Zeitintervall  $q$  (Zeitscheibe, engl. slot) zur Verfügung. Nach Ablauf der entsprechenden Zeit wird zum nächsten Prozess umgeschaltet.

Bereitzeiten



Parameter:

$i$	$r_i$	$\Delta e_i$
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

RR:  $q = 3$

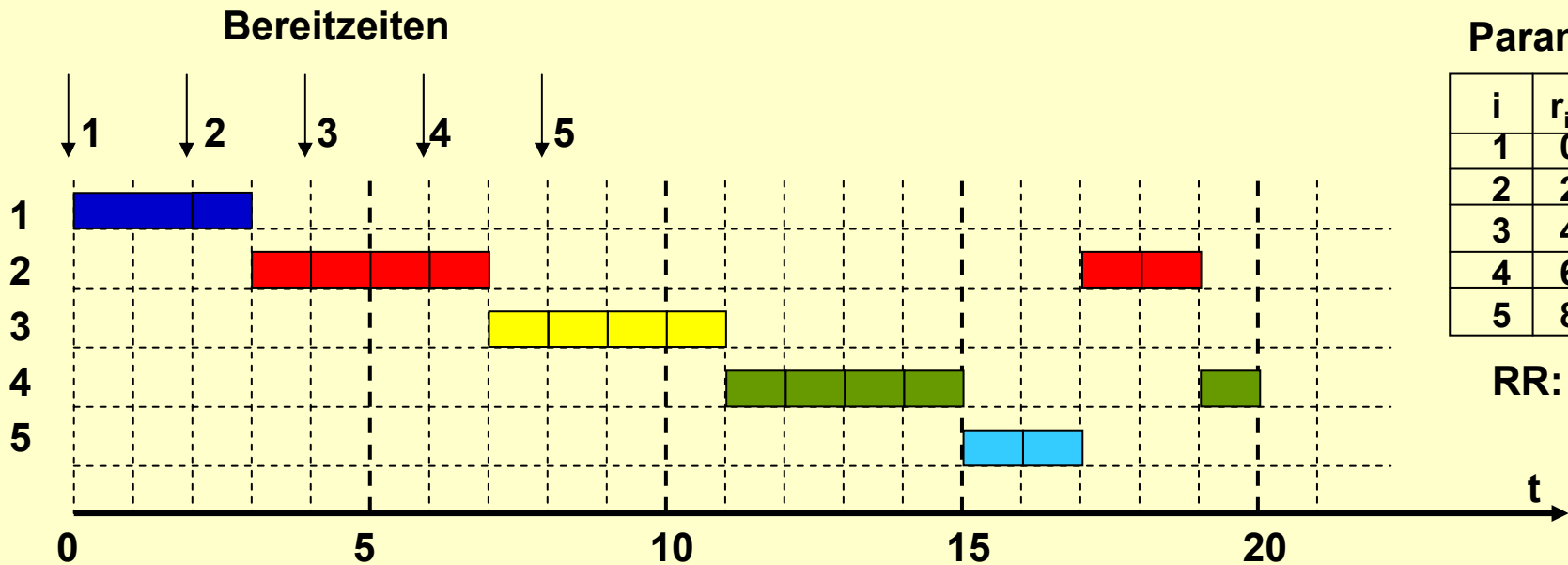
Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	17	18	20	14	
Durchlaufzeit	3	15	14	14	6	10,40
$\Delta r_i / \Delta e_i$	1,00	2,50	3,5	2,80	3,00	2,56

normalisierte Durchlaufzeit



# RR - Round Robin

Das Scheduling ist in "Runden" organisiert. In jeder Runde steht jedem Prozess ein festes Zeitintervall  $q$  (Zeitscheibe, engl. slot) zur Verfügung. Nach Ablauf der entsprechenden Zeit wird zum nächsten Prozess umgeschaltet.



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	17	11	20	19	
Durchlaufzeit	3	15	7	14	11	10,00
$\Delta r_i / \Delta e_i$	1,00	2,50	1,75	2,80	5,50	2,71

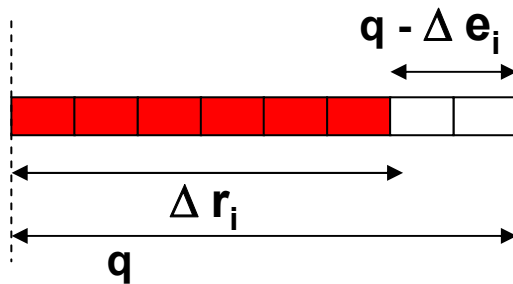
normalisierte Durchlaufzeit



# Probleme mit RR

Verfahren ist abhängig von Zeitscheibenlänge

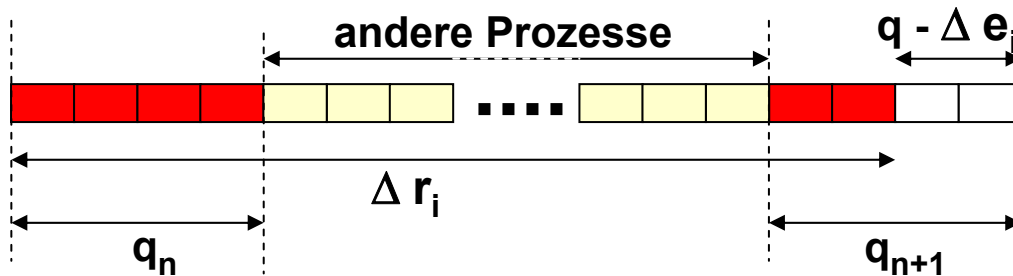
Zeitquantum > durchschnittl. Durchlaufzeit: **Zeit geht verloren**



$q$ : Zeitquantum

$\Delta r_i$ : Durchlaufzeit (Ausführungszeit + E/A Wartezeit)

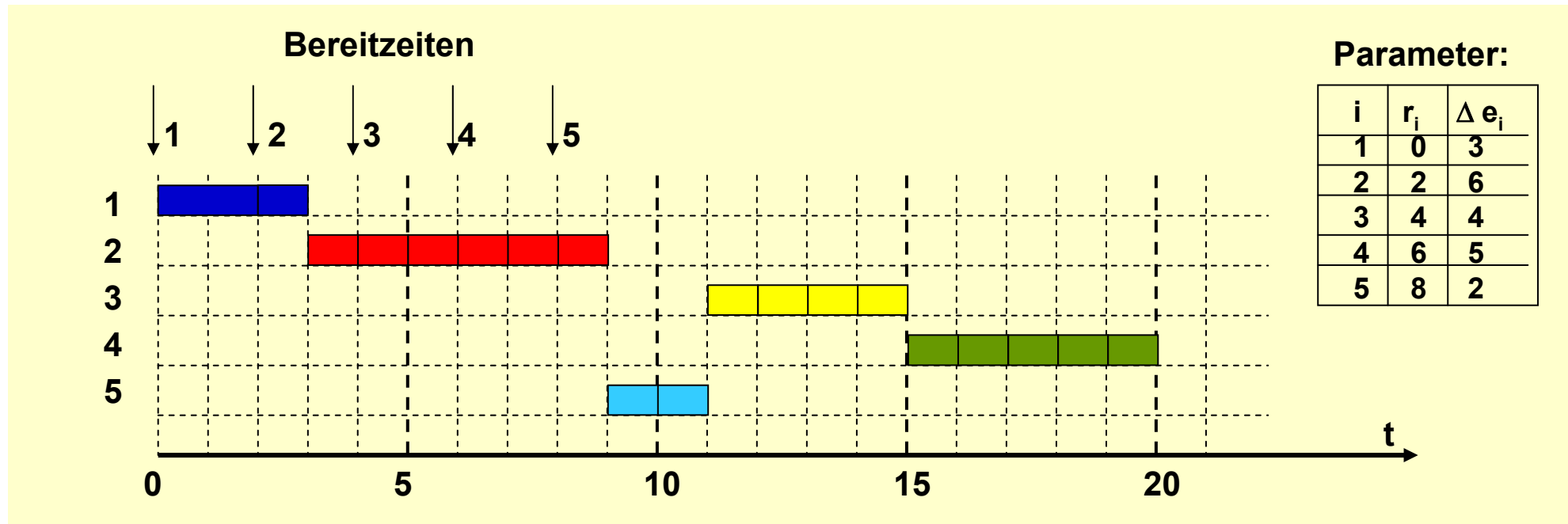
Zeitquantum < durchschnittl. Durchlaufzeit: **Abschlusszeit verschiebt sich**





# SPN - Shortest Process Next

Der Scheduler wählt den Prozess aus, dessen erwartete Ausführungszeit am kürzesten ist. Dieser Prozess wird nicht von höherrangigen Prozessen unterbrochen.



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	9	15	20	11	
Durchlaufzeit	3	7	11	14	3	7,60
$\Delta r_i / \Delta e_i$	1,00	1,17	2,75	2,80	1,50	1,81

normalisierte Durchlaufzeit



# Voraussetzung für SPN

## Verfahren erfordert Kenntnis der Ausführungszeiten:

- Analyse der Ausführungszeiten
- Abschätzung der Ausführungszeiten
- Messung der Ausführungszeiten

## Verfahren: Abschätzung der Ausführungszeiten (fortlaufende Mittelwertbildung)

$$S_{n+1} = (1/n) \sum_{(i=1 \text{ to } n)} \Delta r_i$$

→ tatsächliche Ausführungszeit der i-ten Ausführung

→ Abschätzung der (i+1)ten Ausführung

Lässt sich umstellen zu:

$$S_{n+1} = (1/n) \Delta r_n + ((n-1)/n) S_n$$

**Vorteil:** man muss nicht jedes Mal die gesamte Summe neu berechnen.

**Nachteil:** Die gesamte Vergangenheit geht in die Berechnung ein.

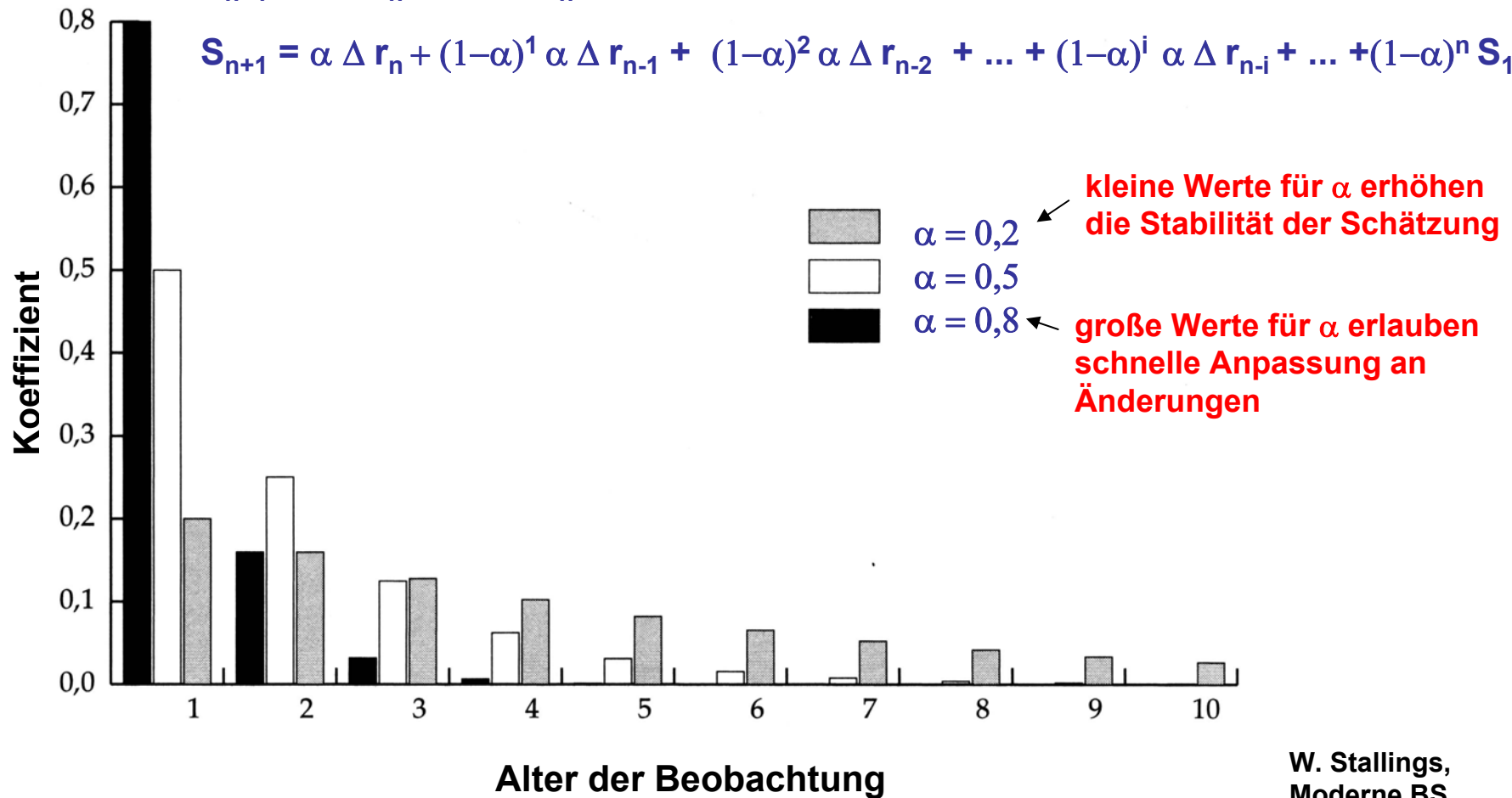
**Ziel:** Neuere Ausführungszeiten sollen stärker gewichtet werden.



# Exponentielle Mittelwertbildung

$$S_{n+1} = \alpha \Delta r_n + (1-\alpha) S_n \quad (0 < \alpha < 1) \quad \text{Erweiterung führt zu:}$$

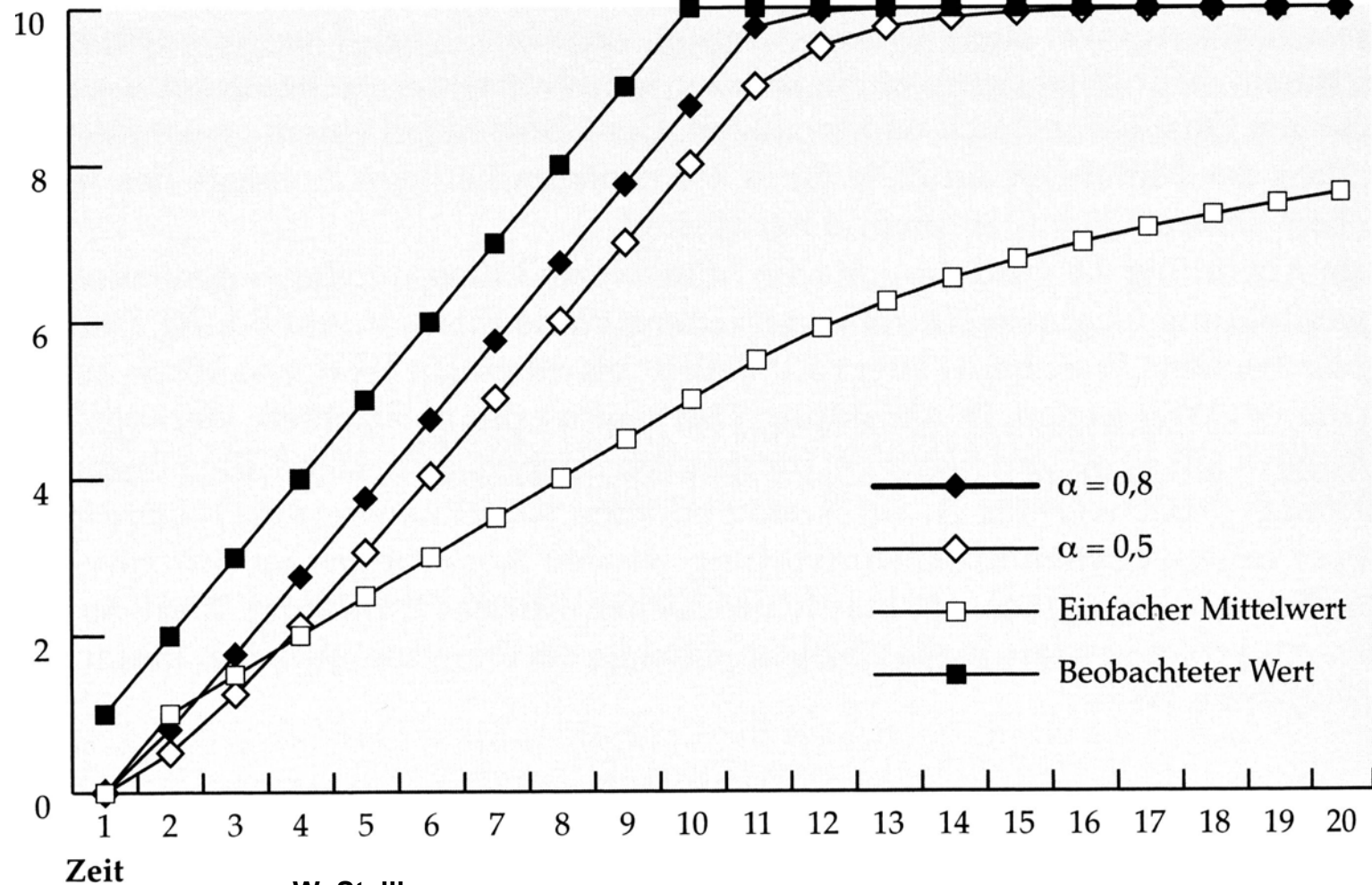
$$S_{n+1} = \alpha \Delta r_n + (1-\alpha)^1 \alpha \Delta r_{n-1} + (1-\alpha)^2 \alpha \Delta r_{n-2} + \dots + (1-\alpha)^i \alpha \Delta r_{n-i} + \dots + (1-\alpha)^n S_1$$



W. Stallings,  
Moderne BS,  
Kap. 9



# Vergleich der Verfahren zur Mittelwertbildung

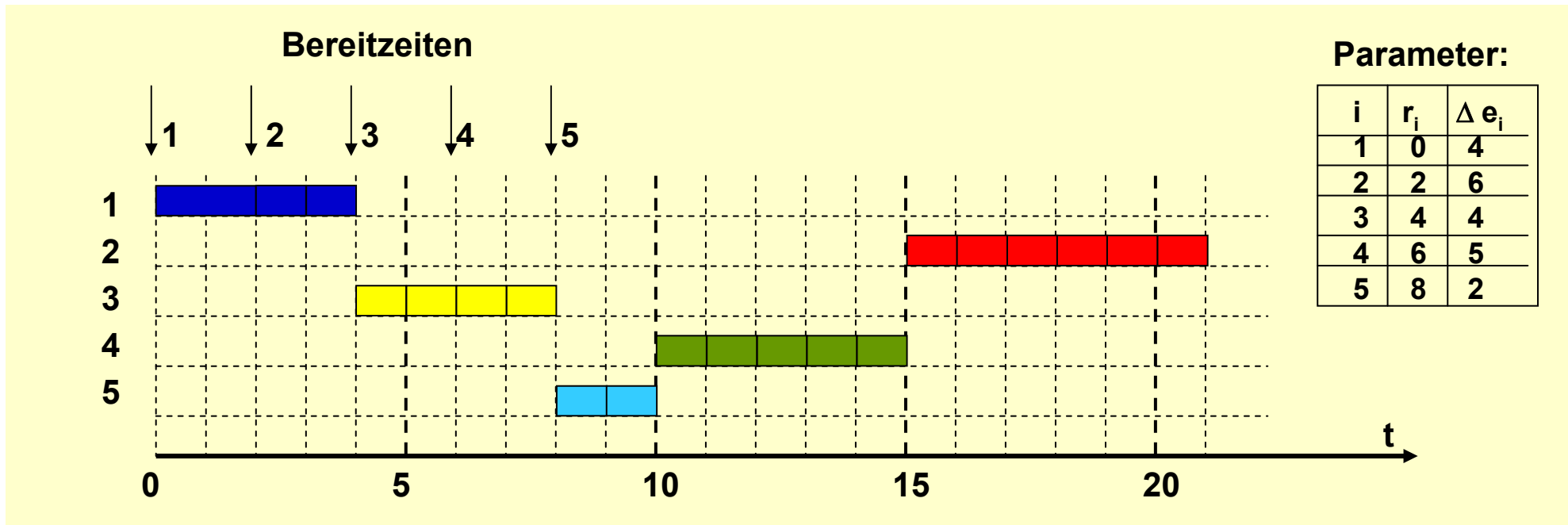


W. Stallings,  
Moderne BS,  
Kap. 9



# Probleme mit SPN

Lange Prozesse werden stark benachteiligt !



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	4	21	8	15	10	
Durchlaufzeit	4	19	4	9	2	7,60
$\Delta r_i / \Delta e_i$	1,00	3,17	1,00	1,80	1,00	1,70

normalisierte Durchlaufzeit



# Probleme mit SPN

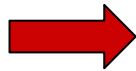
---

**nicht unterbrechbar**



**nicht geeignet für interaktive Anwendungen  
oder Time Sharing Betrieb**

**bevorzugt kurze  
Prozesse**

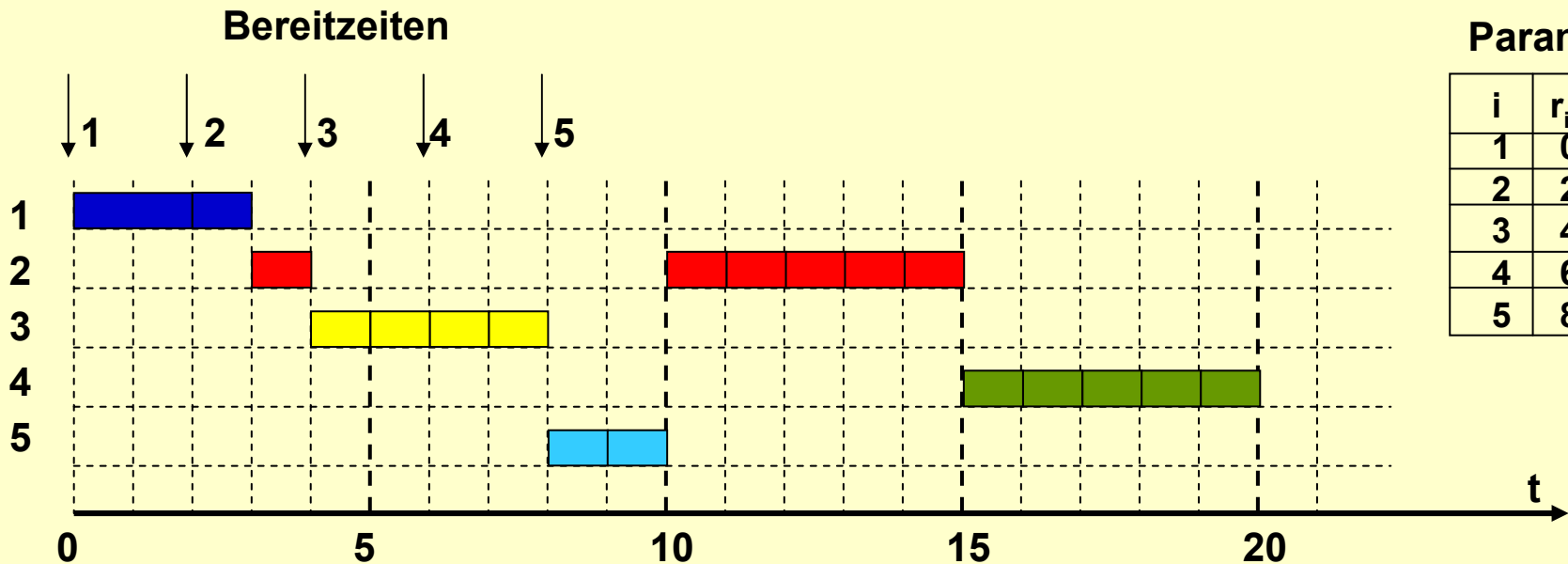


**lange Prozesse können möglicherweise  
verhungern, d.h. kommen nicht zum Zug,  
wenn ständig neue, kurze Prozesse bereit  
werden.**



# SRT - Shortest Remaining Time First

Es wird der Prozess mit der kürzesten noch verbleibenden Ausführungszeit ausgewählt.  
 Falls ein Prozess mit einer kürzeren Zeit bereit wird, wird der ausführende Prozess **unterbrochen** und der Prozessor dem neuen Prozess zugeteilt.



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	15	8	20	10	
Durchlaufzeit	3	13	4	14	2	7,20
$\Delta r_i / \Delta e_i$	1,00	2,17	1,00	2,80	1,00	1,59

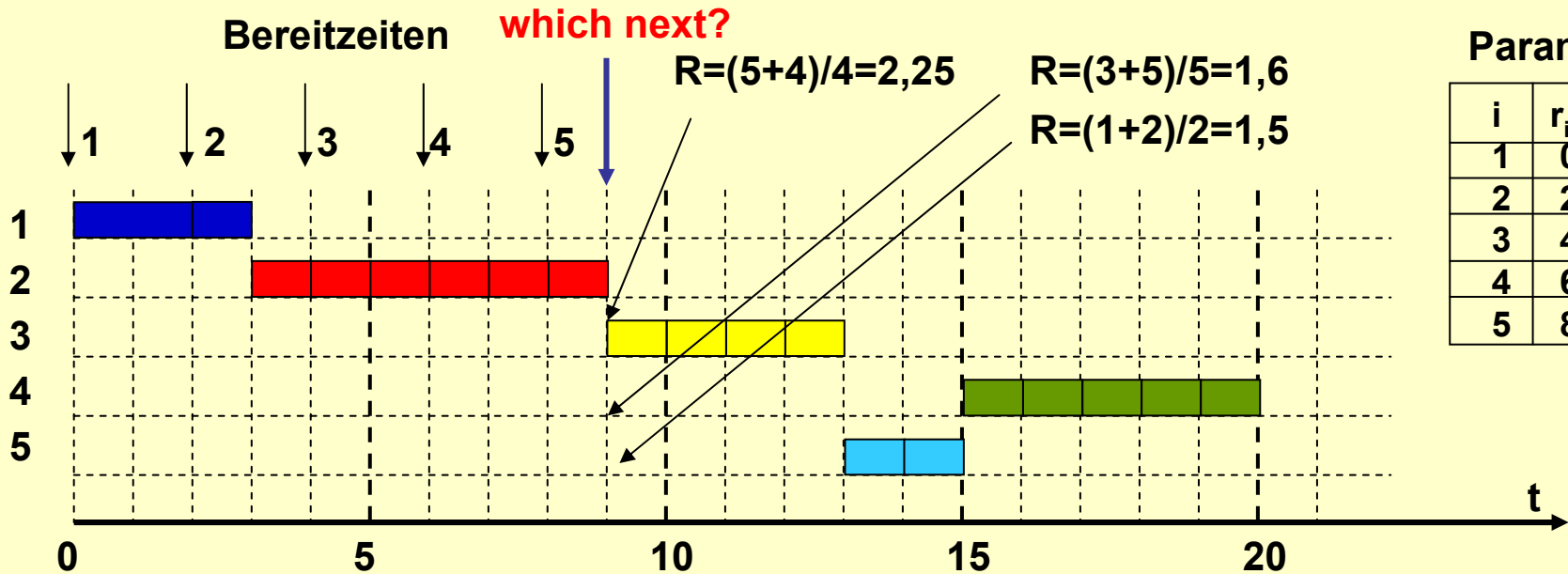
normalisierte Durchlaufzeit



# HRRN - Highest Response Ratio Next

Der Scheduler wählt Prozesse aufgrund der normalisierten Durchlaufzeit (R) aus.

$$R = \Delta r_i / \Delta e_i = (w + \Delta e_i) / \Delta e_i$$



Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	9	13	20	15	
Durchlaufzeit	3	7	9	14	7	8,00
$\Delta r_i / \Delta e_i$	1,00	1,17	2,25	2,80	3,50	2,14

normalisierte Durchlaufzeit





# FB - Feedback Scheduling

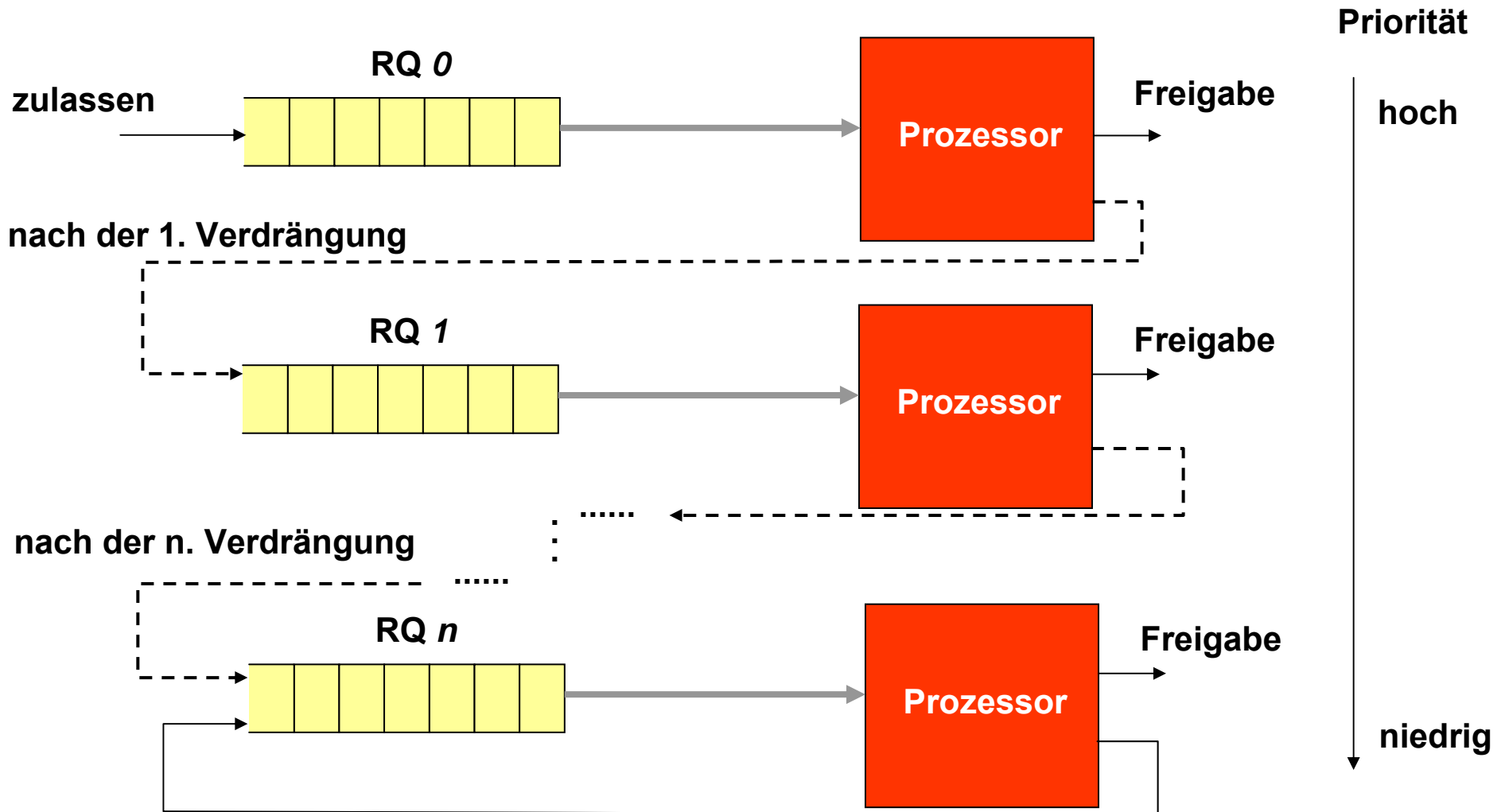
---

**Problem** mit SPN-, SRT-, und HRRN-Strategien:  
Ausführungszeit der Prozesse muss abschätzbar sein.

**Idee:** Nutzung der Kenntnis der bis zu einem bestimmten Zeitpunkt  
verwendeten Ausführungszeit.



# FB - Feedback Scheduling



# FB - Feedback Scheduling

---

## Eigenschaften:

- ➔ in den Warteschlangen  $0, \dots, n-1$  wird das FCFS-Verfahren angewendet,
- ➔ in der letzten Warteschlange wird ein RR-Verfahren angewendet,
- ➔ ein kurzer Prozess ist schnell abgeschlossen und gelangt daher nicht in eine niedrige Prioritätsstufe,
- ➔ lange Prozesse rutschen in RQ  $n$  ; Möglichkeit des Verhungerns (Starvation)
  - ➔ unterschiedliche Zeitquanten (ZQ) für die Warteschlangen:
    - 1 ZQ für RQ 1
    - 2 ZQ für RQ 2
    - ⋮
    - $2^i$  ZQ für RQ  $i$
    - ⋮
    - $2^n$  ZQ für RQ  $n$



# Vergleich der Scheduling Strategien

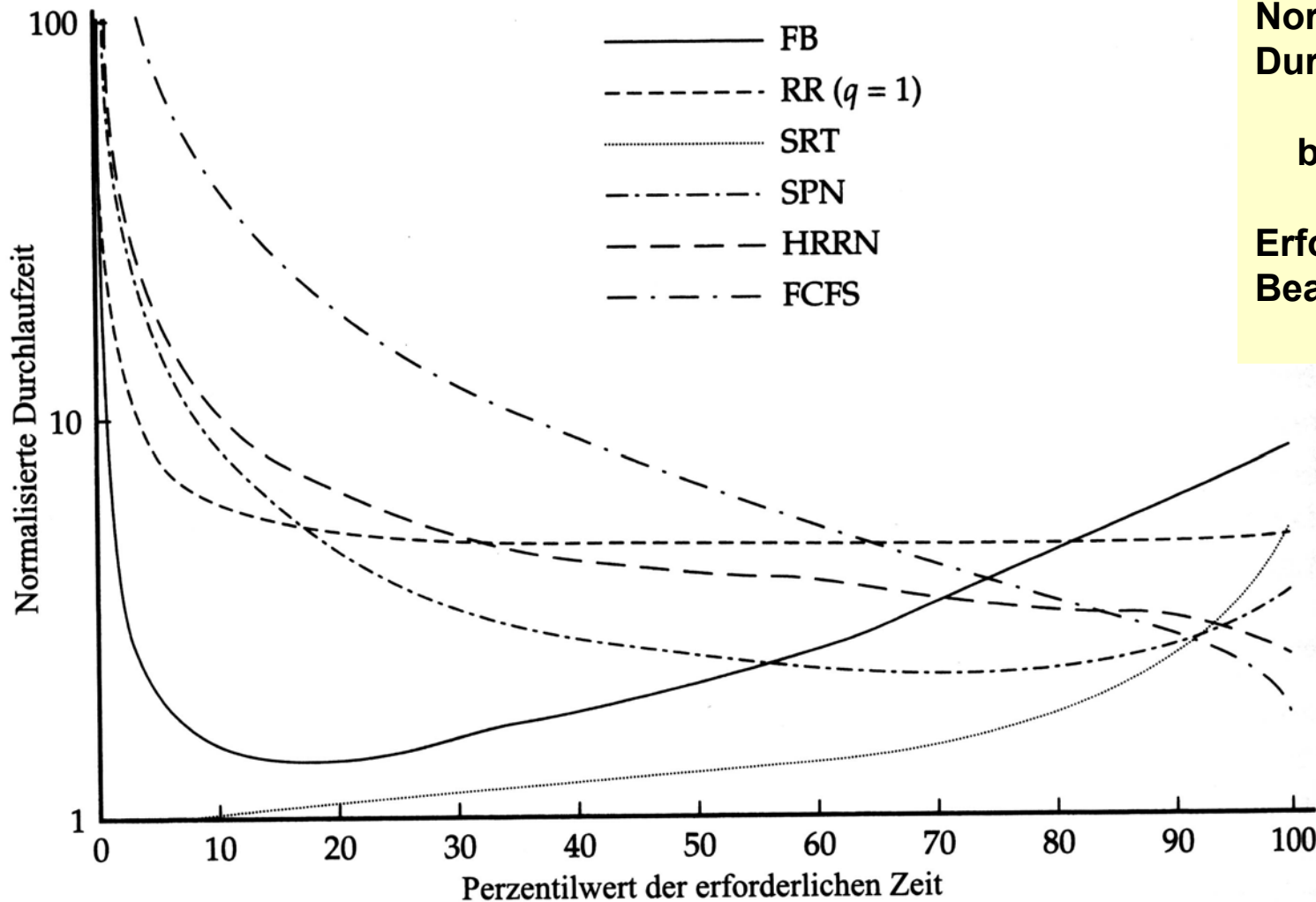
	Auswahl-funktion	Entscheidungsmodus	Durchsatz	Antwortzeit	Overhead	Auswirkung auf Prozesse	Verhungen
FCFS	$\max[w]$	Nicht unterbrechend	Spielt keine Rolle	Kann hoch sein, insbesondere, bei großer Varianz der Prozessausführungszeiten	Minimal	Benachteiligt kurze Prozesse; benachteiligt E/A-lastige Prozesse	Nein
Round Robin	Konstante	Unterbrechend (nach bestimmter Zeitdauer)	Ist gering, wenn Zeitdauer gering ist.	Bietet bei kurzen Prozessen gute Antwortzeit	Minimal	Faire Behandlung	Nein
SPN	$\min [\Delta e]$	Nicht unterbrechend	Hoch	Bietet bei kurzen Prozessen gute Antwortzeit	Kann groß sein	Benachteiligt lange Prozesse	Möglich
SRT	$\min [\Delta R]$	Unterbrechend (bei Eingang)	Hoch	Bietet gute Antwortzeit	Kann groß sein	Benachteiligt lange Prozesse	Möglich
HRRN	$\max \frac{w + [\Delta e]}{[\Delta e]}$	Nicht unterbrechend	Hoch	Bietet gute Antwortzeit	Kann groß sein	Gute Lastverteilung	Nein
Feedback		Unterbrechend (nach bestimmter Zeitdauer)	Spielt keine Rolle	Spielt keine Rolle	Kann groß sein	Bevorzugt u.U. E/A-lastige Prozesse	Möglich

$[\Delta R]$ = Gesamtausführungszeit - bereits verwendete Zeit

W. Stallings,  
Moderne BS,  
Kap. 9



# Vergleich der Schedulingstrategien



Normalisierte Durchlaufzeit

bezogen auf die

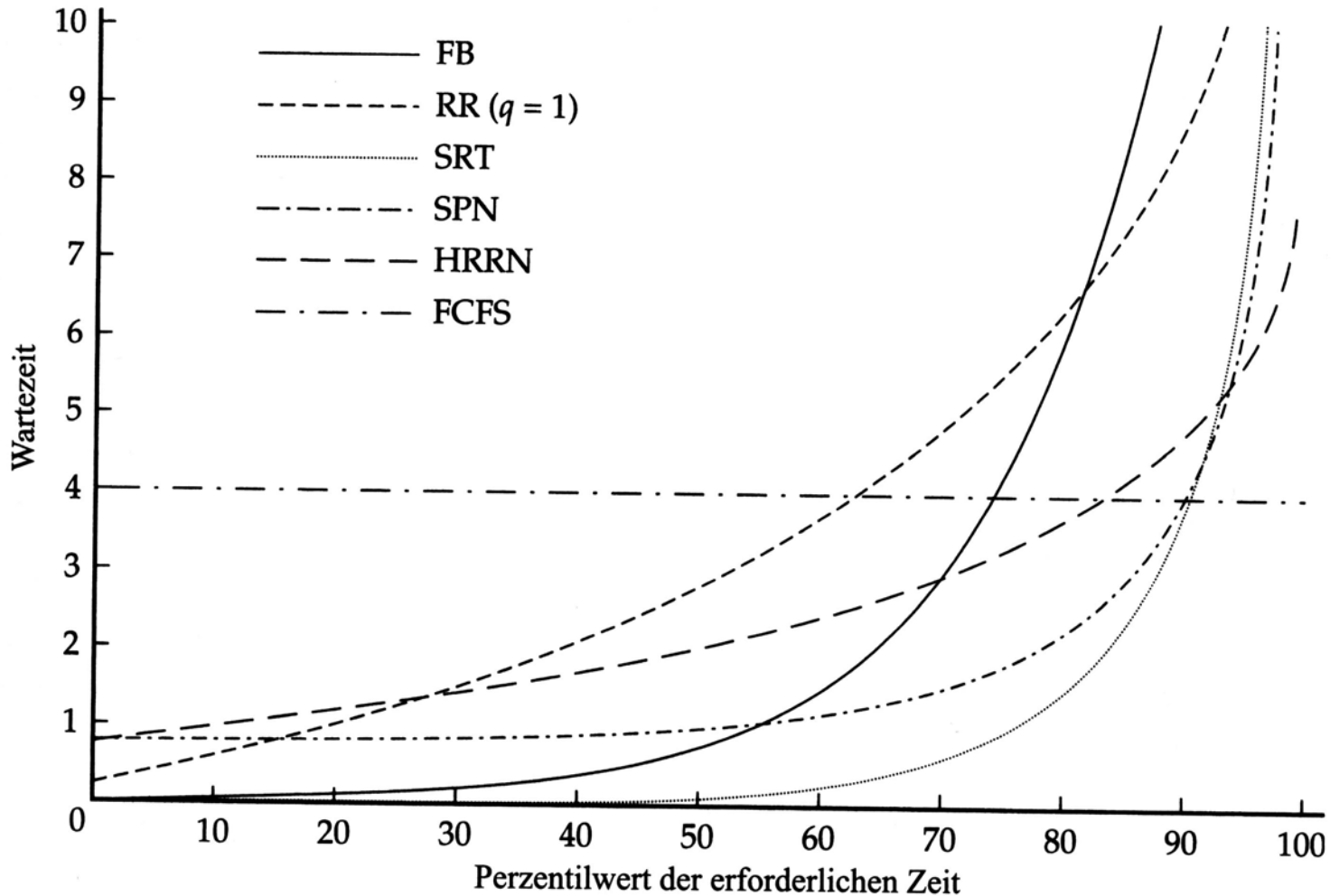
Erforderliche Bearbeitungszeit

50000 Prozesse unterteilt in 100 Gruppen (Perzentil) gemäß ihrer Bearbeitungszeit.

W. Stallings, Moderne BS, Kap. 9



# Vergleich der Schedulingstrategien



**Wartezeit**  
bezogen auf die  
**Erforderliche  
Bearbeitungszeit**

W. Stallings,  
Moderne BS,  
Kap. 9



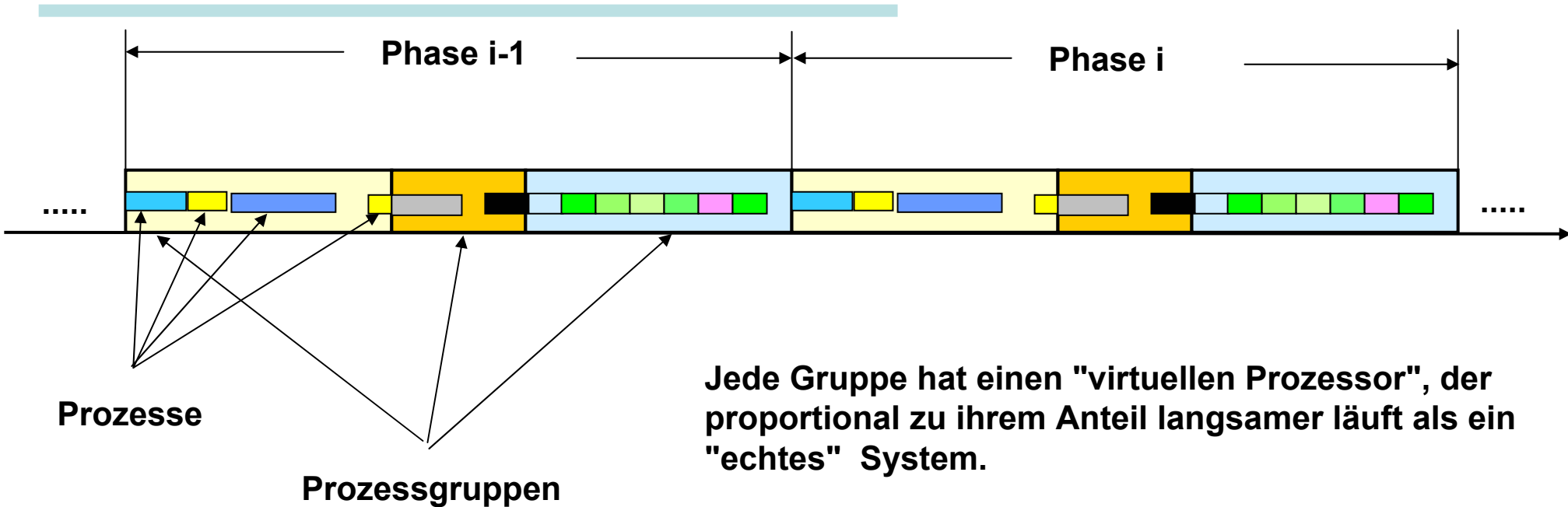
# Fair-Share-Scheduling

---

- Motivation:** Prozesse sind nicht unabhängig, sondern in Gruppen abhängiger Prozesse zusammengefasst.
- Idee:** Scheduler versucht die CPU-Resource(n) zwischen den Gruppen fair zu verteilen. Wächst in einer Gruppe die Anforderung, soll sich das zunächst nicht auf andere Gruppen auswirken.
- Verfahren:** Benutzer werden bezüglich der Gruppe gewichtet und erhalten einen darauf bezogenen Anteil (Share) an den Gruppenressourcen. Scheduler überwacht, dass Benutzer, die ihren Anteil weniger ausgeschöpft haben mehr, andere weniger CPU-Zeit bekommen.



# Fair-Share-Scheduling



$CPU_j(i)$

CPU Auslastung durch den Prozess  $j$  im Intervall  $i$ .

$GCPU_k(i)$

CPU Auslastung durch die Gruppe  $k$  im Intervall  $i$ .

$P_j(i)$

Priorität des Prozesses  $j$  zu Beginn des Intervalls  $i$ ; **niedrigere numerische Werte entsprechen höheren Prioritäten.**

Basis $_j$

Basispriorität von Prozess  $j$ .

$W_k$

Gewicht der Gruppe  $k$  mit  $0 < W_k \leq 1$  und  $\sum W_k = 1$





# Fair-Share-Scheduling

"Fair" beinhaltet:

- ➔ kein einzelner Prozess soll die Prozessorzeit einer Gruppe aufbrauchen,
- ➔ die Priorität eines Prozesses berücksichtigt bereits verbrauchte Prozessorzeit,
- ➔ zu Beginn jeder Phase wird ein Maß für die Prozessorauslastung und die Priorität neu festgelegt.

Für Prozess  $j$  der Gruppe  $k$  gilt dabei die Festlegung:

$$\text{➔ } CPU_j(i) = CPU_j(i-1) / 2$$

CPU Auslastung von Prozess  $j$  im Intervall  $i$ .

$$\text{➔ } GCPU_k(i) = GCPU_k(i-1) / 2$$

CPU Auslastung von Gruppe  $k$  im Intervall  $i$ .

$$\text{➔ } P_j(i) = \text{Basis} + CPU_j(i-1) / 4 + GCPU_k(i-1) / 8 \times W_k$$

Priorität von Prozess  $j$  zu Beginn des Intervalls  $i$ .  $W_k$  ist das Gewicht der Gruppe mit  $0 < W_k \leq 1$  und  $\sum W_k = 1$

**Je mehr CPU-Zeit und Gruppenzeit verbraucht wurde, desto geringer die Priorität!**



# Fair-Share-Scheduling

Beispiel:

3 Prozesse,  
2 Gruppen

W. Stallings,  
Moderne BS,  
Kap. 9

Zeit	Prozess A			Prozess B			Prozess C		
	Priorität	Prozess	Gruppe	Priorität	Prozess	Gruppe	Priorität	Prozess	Gruppe
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0 0 0 0 0 0	0 0 0 0 0 0	60	0 0 0 0 0 0	0 0 0 0 0 0
1	90	30 30 30 30 30 30	30 30 30 30 30 30	60	0 1 2 • • 60	0 1 2 • • 60	60	0 0 0 0 0 0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30 30 30 30 30 30	30 30 30 30 30 30	75	0 0 0 0 0 0	30 30 30 30 30 30
3	96	37 37 37 37 37 37	37 37 37 37 37 37	74	15 15 15 15 15 15	15 15 15 15 15 15	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7 7 7 7 7 7	37 37 37 37 37 37	93	30 30 30 30 30 30	37 37 37 37 37 37
5	98	39 39 39 39 39 39	39 39 39 39 39 39	70	3 3 3 3 3 3	18 18 18 18 18 18	76	15 15 15 15 15 15	18 18 18 18 18 18



Graues Feld = aktiver Prozess



---

# Ein kurzer Ausflug in das Scheduling unter Echtzeitbedingungen

## Bisherige Ziele und Annahmen:

- ➡ Fairness,
- ➡ alle Prozesse besitzen die gleiche Wichtigkeit,
- ➡ kein Prozess soll verhungern.

## Eigenschaften von Echtzeitprozessen:

- ➡ Wichtigkeit,
- ➡ Zeitliche Vorhersagbarkeit,
- ➡ Kenntnis der Bereitzeiten, Ausführungszeiten und Fristen.

**General Purpose Scheduling und Echtzeitscheduling besitzen gegensätzliche Ziele!**



# Beispiele populärer Kostenfunktionen

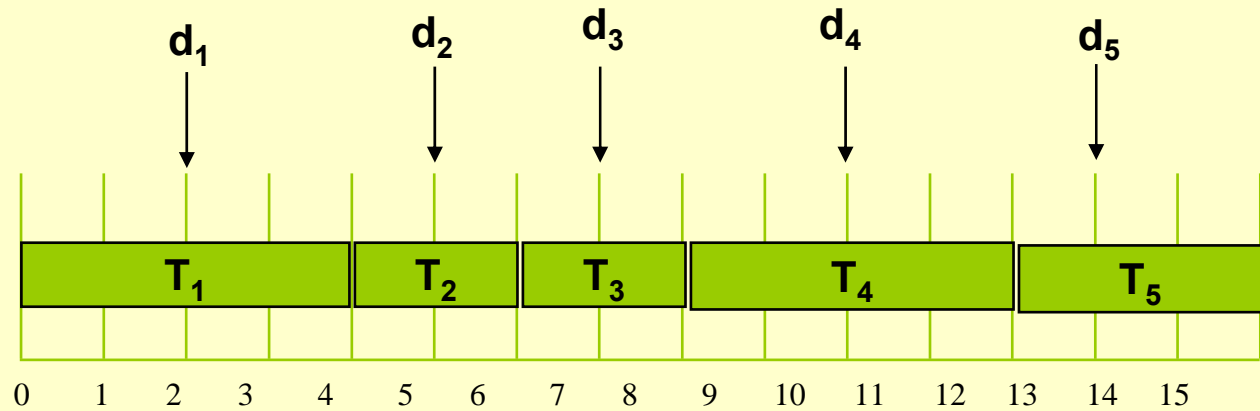
---

- **Mittlere Antwortzeit:**  $t_r = 1/n \sum_{(i=1, \dots, n)} (c_i - r_i)$   
(average response time)
- **Zeit bis zum vollständigen Abschluß (aller Prozesse):**  $t_c = \max_i (c_i) - \min_i (r_i)$   
(total completion time)
- **Maximale Verspätung:**  $L_{max} = \max_i (c_i - d_i)$   
(maximum lateness)
- **Maximale Zahl verspäteter Tasks:**  $N_{late} = \sum_{(i=1, \dots, n)} miss(c_i)$   
mit:  $miss(c_i) = \begin{cases} 0 & \text{if } c_i \leq d_i \\ 1 & \text{sonst} \end{cases}$



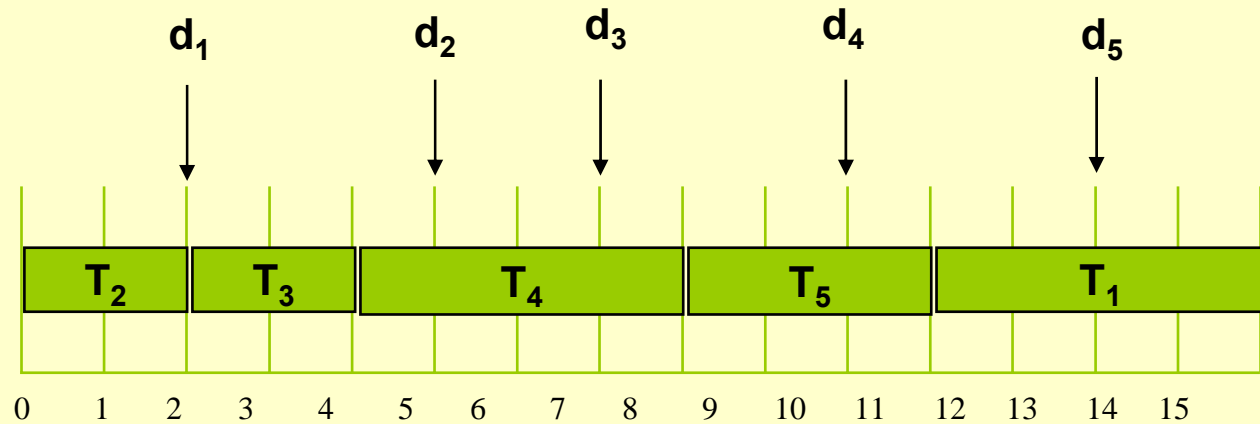
## Beispiel für Kostenfkt.

Minimierung von  $L_{max}$



$$L_{max} = L_1 = L_4 = L_5 = 2, N_{late} = 5$$

Minimierung von  $N_{late}$



$$L_{max} = L_1 = 13, N_{late} = 1$$



---

## Wesentliches Kriterium des klassischen Echtzeitschedulings:

- **Maximale Verspätung:**  $L_{max} = \max_i (c_i - d_i)$   
(maximum lateness)

Für Prozesse mit harten Zeitbedingungen gilt dabei:

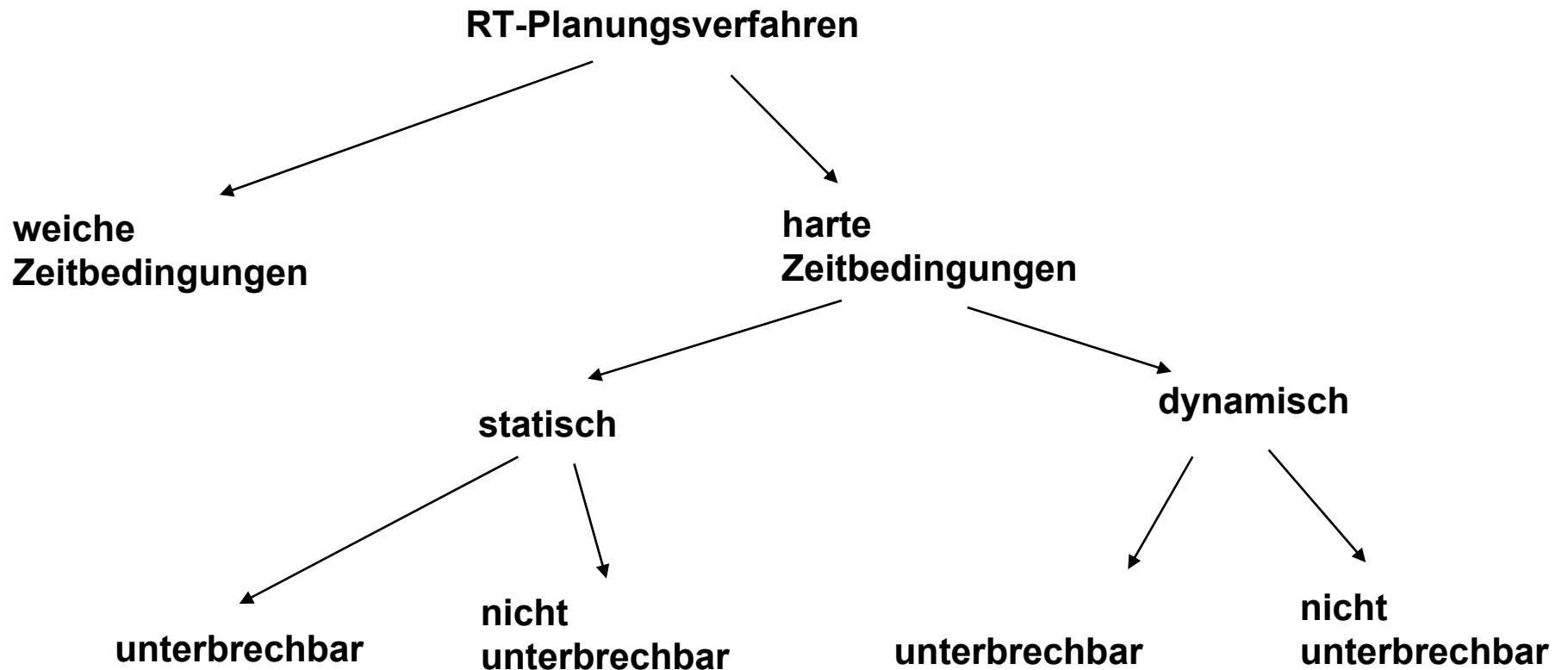
$$L_{max} \leq 0$$

**Die vorgegebene Deadline muss immer eingehalten werden.**

# Taxonomie der RT-Planungsverfahren

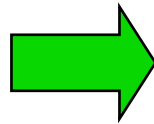
(Cheng, Stankovic, Ramamritham)

---

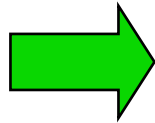


# Klassifizierung der Schedulingalgorithmen

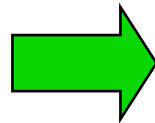
---



**Annahmen über die Systemumgebung:  
Einprozessor-System,  
Mehrprozessor-System,  
verteiltes System, .....**



**Annahmen über die Task-Menge:  
Unterbrechbarkeit,  
Unabhängigkeit,  
Aktivierung, .....**



**Planungs(Optimalitäts)kriterium  
bezogen auf eine Kostenfunktion**



# EDF : Earliest Deadline First

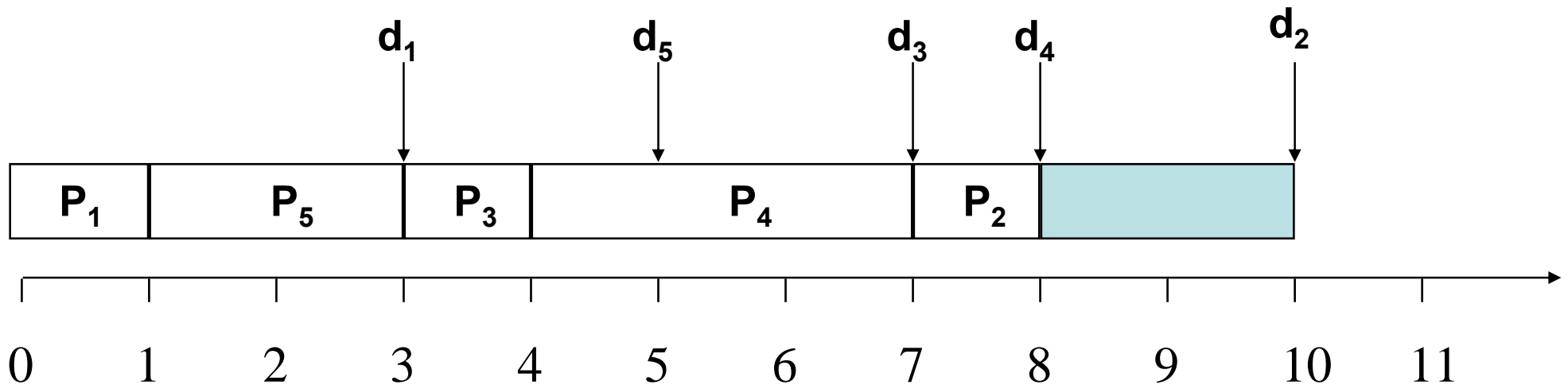
---

- ➔ **Die Ausführungszeiten und Deadlines von Prozessen sind bekannt (z.B. stehen als Parameter im Prozess-Kontrollblock).**
- ➔ **Prozesse, die ausführbereit sind liegen in einer Liste vor, die nach Deadlines geordnet ist.**
- ➔ **Prozesse, die bereit werden, werden dynamisch in diese Liste an die entsprechende Position eingefügt.**
- ➔ **Der Prozessor wird jeweils dem Prozess zugeteilt, der die zeitlich am nächsten liegende Deadline besitzt.**
- ➔ **Der Scheduler trifft Entscheidungen, wenn ein Prozess neu bereit wird, oder ein Prozess blockiert oder beendet wird.**



# Beispiel für EDF

	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$
$\Delta e_i$	1	1	1	3	2
$d_i$	3	10	7	8	5



$$L_{\max} = L_4 = -1$$



# EDF : Earliest Deadline First

**EDF ist eine optimale Schedulingstrategie im Hinblick auf die Minimierung der maximalen Verspätung.**

**Für ein Echtzeitsystem, in dem alle Deadlines eingehalten werden müssen (harte Echtzeitbedingungen) bedeutet das:**

**Wenn EDF keinen Schedule findet, in dem alle Deadlines eingehalten werden, findet auch kein anderes Verfahren einen Plan (inklusive des allwissenden Schedulers)**

- **Eine der am weitesten verbreiteten zeitbasierten Planungsstrategien.**
- **EDF ist eine dynamische Planungsstrategie, d.h. die Priorität der Prozesse ändern sich in der Zeit.**
- **Anwendbar auf unterbrechbare und (unter der Voraussetzung, dass alle Prozesse gleichzeitig bereit sind) nicht unterbrechbare Prozesse.**
- **Anwendbar in off-line und on-line Planungsverfahren.**



# RMS: Echtzeitschedulingverfahren für periodische Prozesse

---

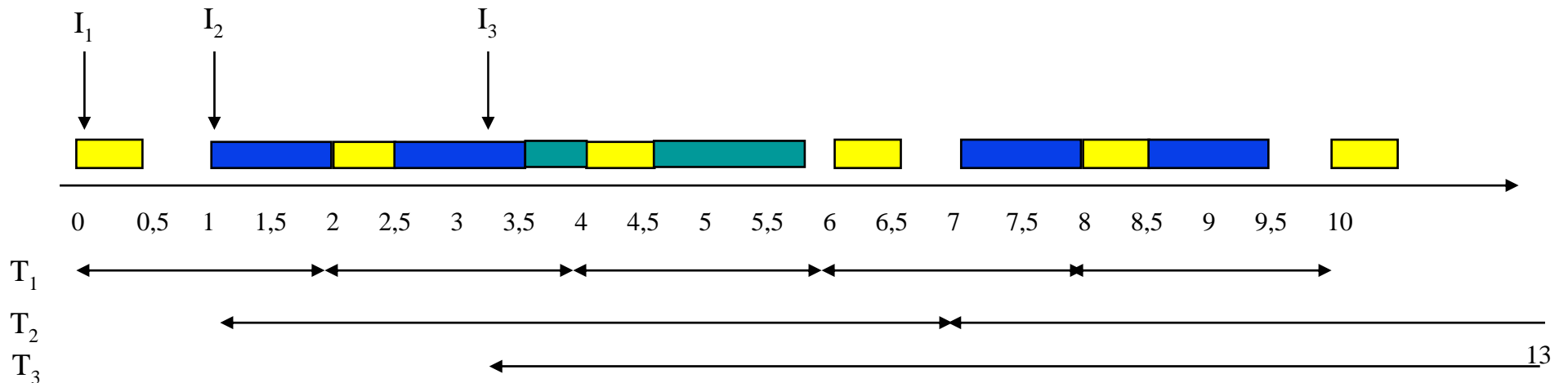
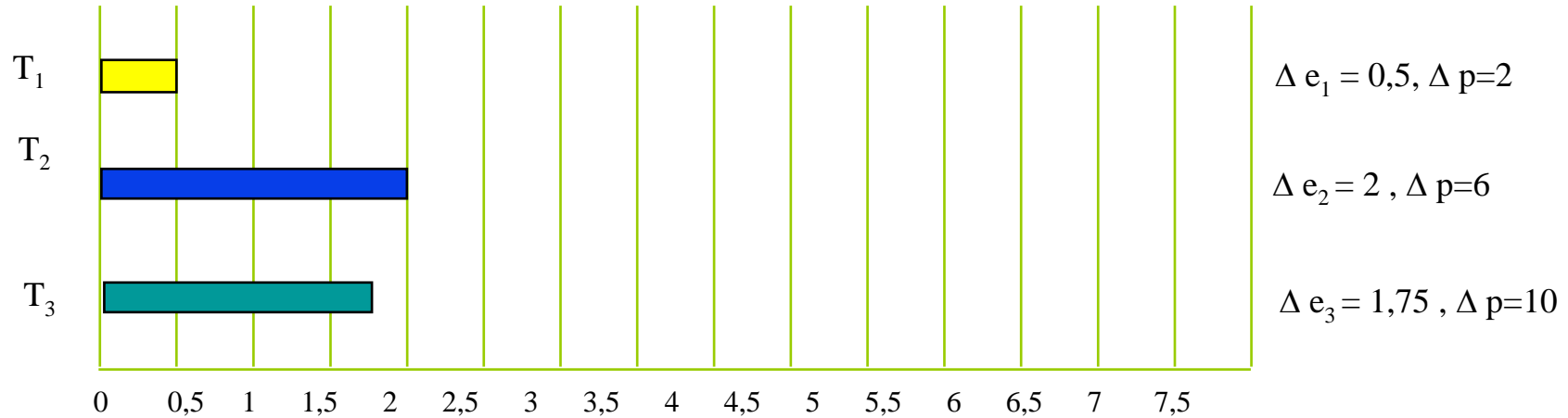
**Rate Monotonic Scheduling (RMS):** Für einen periodischen Prozess wird die Priorität als invers zur Länge seiner Periode definiert, d.h. je kürzer die Periode, desto höher die Priorität.

- ➔ Statisches Planungsverfahren, d.h. die Priorität eines Prozesses ist fest.
- ➔ Optimales statisches Planungsverfahren, d.h. wenn RMS keinen Plan findet, wird auch kein anderes statisches Verfahren mit festen Prioritäten einen Plan finden.

C.L. Liu, J.W. Layland: "Scheduling algorithms for multiprogramming in a hard- real-time environment", Journal of the ACM 20(1), January 1973, pp.46-61



# Rate Monotonic Scheduling (RMS): Beispiel



# Rate Monotonic Scheduling (RMS): Eigenschaften

Für bestimmte Auslastungsgrenzen der CPU kann garantiert werden, dass alle periodischen Prozesse ihre Deadlines einhalten.

$U_{lub}$  ist die kleinste obere Schranke (least upper bound) der Prozessorauslastung.

Für  $n$  Tasks gilt:  $U_{lub} = n ( \sqrt[n]{2} - 1 ) .$

Für  $n = 1$  :  $U_{lub} = 1$

Für  $n = 2$  :  $U_{lub} = 0,828$

Für  $n \rightarrow \infty$  :  $\lim U_{lub} (n) = \ln (2) = 0,693$



# Beispiele:

---

- **Klassisches Unix Scheduling**
- **Linux**
- **Unix SVR4**
- **W2K**



# Klassisches Unix Scheduling

---

Eingesetzt in SVR3 und 4.3 BSD. Eine modifizierte Version wird in Linux verwendet.

Optimiert für interaktive Systeme, d.h. gute Antwortzeiten und faire Zuordnung.

Kombination von Round Robin und Prioritätsbasiertem Scheduling.

Mehrere Warteschlangen, die nicht überlappenden Prioritätsbereichen zugeordnet sind.

Die CPU-Nutzung wird durch einen Zähler gemessen, der alle 100ms (typ.) incrementiert wird.

Jede Sekunde wird die Priorität der Prozesse neu berechnet analog zum Fair-Share Verfahren.

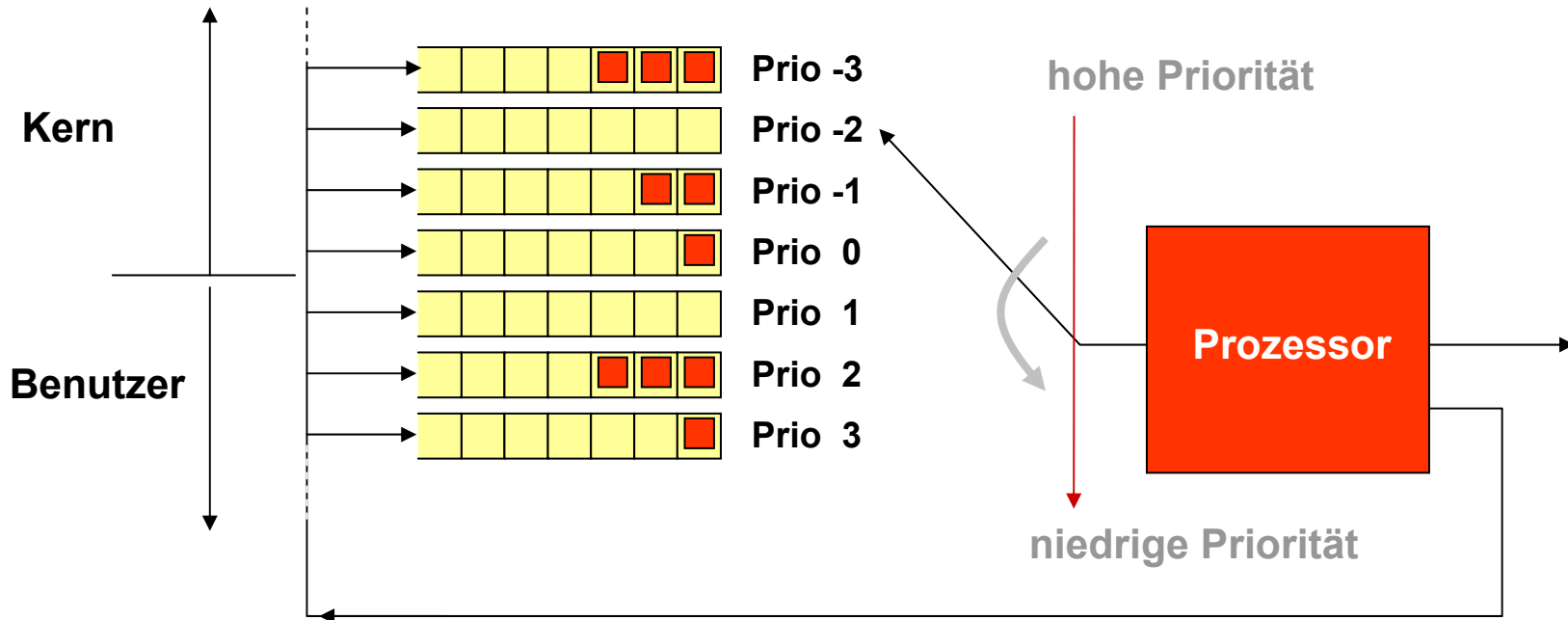




kann durch Systemaufruf von Benutzer gesetzt werden.

$$P_j(i) = \text{Basis} + \text{CPU}_j(i-1) / 4 + \text{nice}$$

Priorität von  
Prozess j im  
Intervall i



Ziel ist es, Prozesse schnell aus dem K.-Bereich in den B.-Bereich zu verschieben. Deshalb werden Prozesse, die auf ein E/A Ereignis gewartet haben und dann wieder bereit werden, in eine W.-Schlange hoher Priorität eingeordnet.

Prioritätsordnung:

- Swapper
- E/A Kontrolle (blockorientierte Geräte)
- Dateiänderungen
- E/A Kontrolle (zeichenorientierte Geräte)
- Benutzerprozesse



# Linux Scheduling

---

Die Einheiten des Scheduling in Linux sind Kernel-Threads.  
Klassisches Unix-Scheduling wird um 3 Zuteilungsklassen erweitert:

- ➔ **SCHED\_FIFO**: Fifo-"Echtzeit"-Threads
- ➔ **SCHED\_RR**: RR-"Echtzeit"-Threads
- ➔ **SCHED\_OTHER**: Threads ohne "Echtzeit"

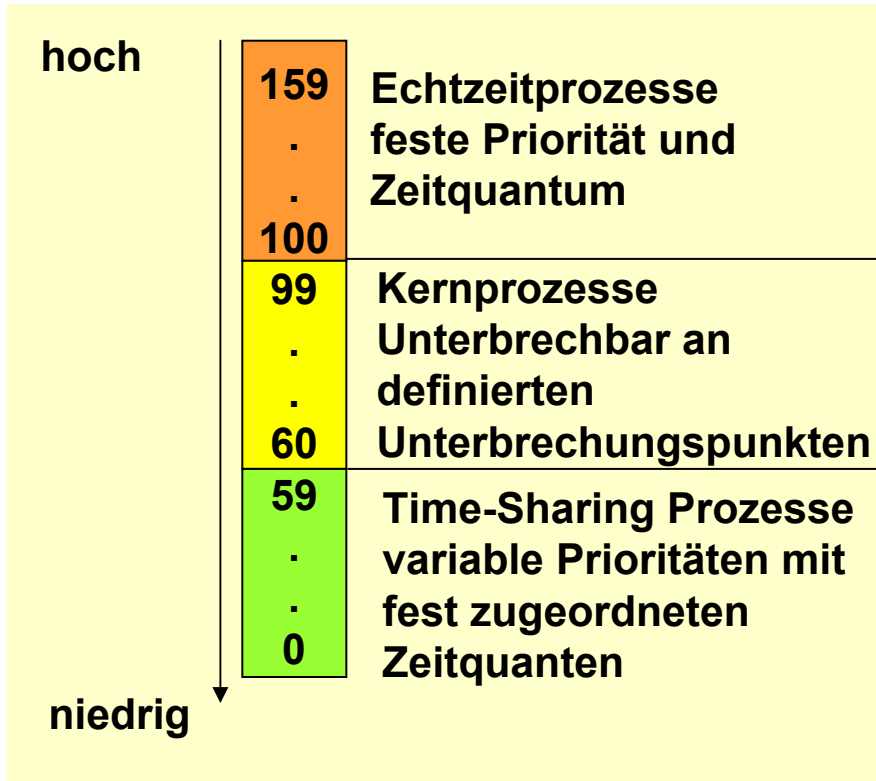
Echtzeit bedeutet hier: FIFO-Threads haben Prioritäten und können nur unterbrochen werden wenn:

- ➔ ein anderer FIFO-Echtzeit-Thread mit höherer Priorität bereit wird,
  - ➔ der aktuelle FIFO-Echtzeit-Thread blockiert,
  - ➔ der aktive FIFO-Echtzeit-Thread explizit den Prozessor freigibt (*sched\_yield*)
- 
- ➔ für RR-Echtzeit-Threads gilt zusätzlich, dass sie bei abgelaufenem Zeitquantum unterbrochen werden, wenn es einen anderen bereiten Prozess derselben Prioritätsstufe gibt.

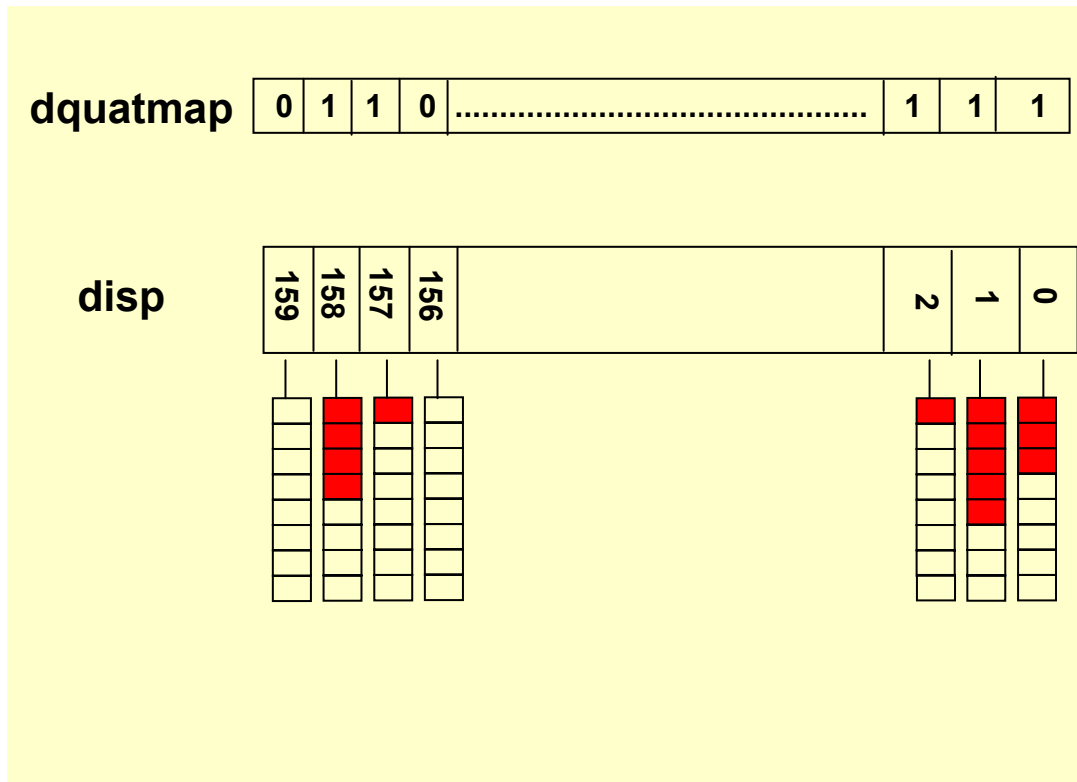


# Scheduling in Unix SVR4

## Prioritätsklassen und Prioritätsstufen



## Organisation der Warteschlange



Innerhalb der Time-Sharing Klasse variable Prioritäten und Zeitquanten. Das Zeitquantum hängt von der Prioritätsstufe ab.



# Scheduling in Unix W2K

## Festlegung der Basis-Prioritätsklassen

**SetPriorityClass**

**Prozessklassen- Prioritäten**

	Echtzeit	hoch	über normal	normal	unter normal	idle
zeitkritisch	31	15	15	15	15	15
höchste	26	15	12	10	8	6
über normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
unter normal	23	12	9	7	5	3
niedrigste	22	11	8	6	4	2
idle	16	1	1	1	1	1

**Thread-Prioritäten**

**SetThreadPriority**



# Scheduling in W2K

## Einheiten des Scheduling: Threads

Priorität =  
Basispriorität des Prozesse  
+ Priorität des Threads

Dyn. Priorität kann Basispriorität erhöhen, aber nicht von einer variablen Priorität in eine Echtzeit-priorität.

