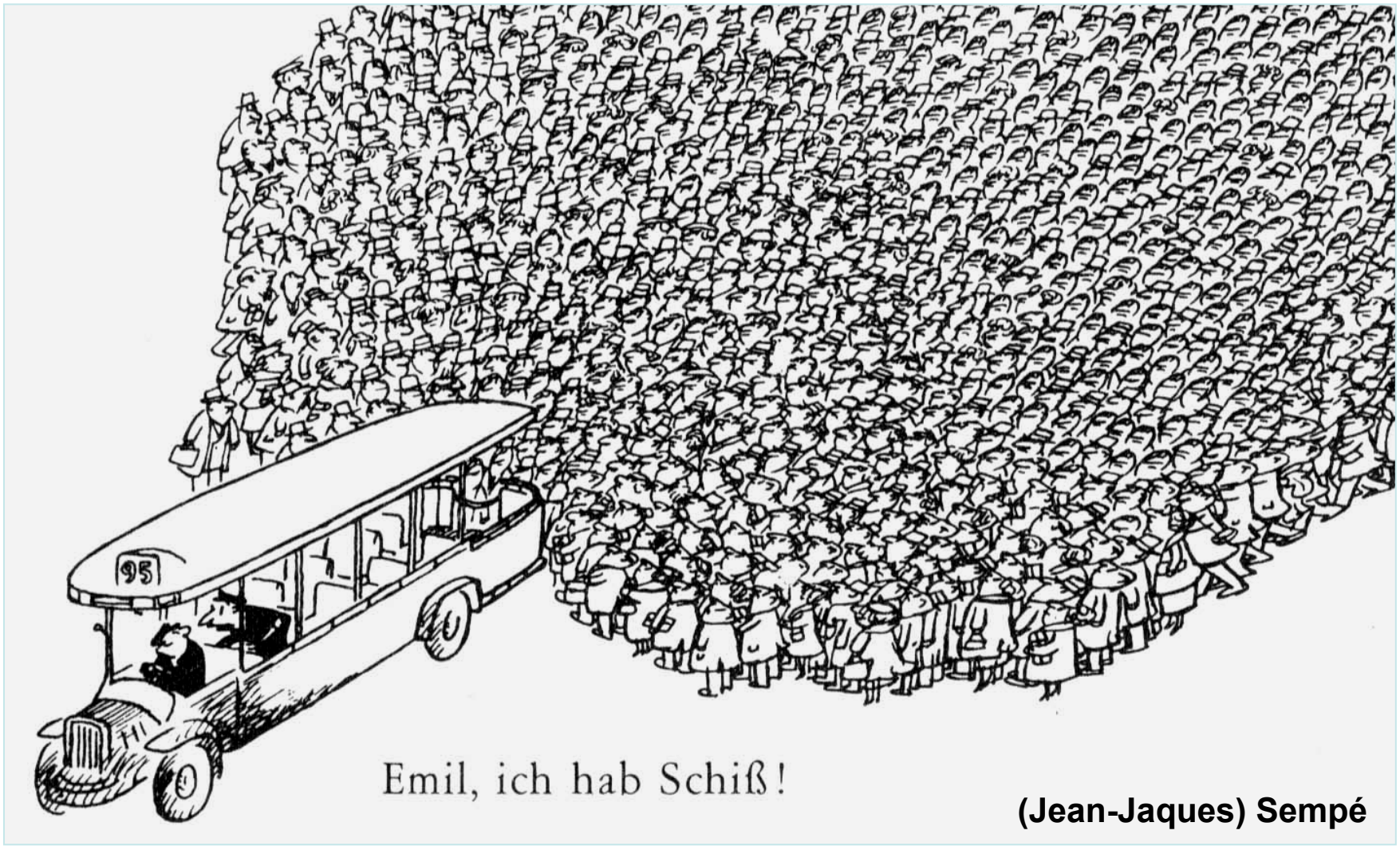
Scheduling

Betriebssysteme
WS 2011/2012

Jörg Kaiser
IVS – EOS

Otto-von-Guericke-Universität Magdeburg

Das Scheduling Problem:



(Jean-Jaques) Sempé

Scheduling - Beispiele

Tagesplan ➡ Ressource: Zeit

Stundenpläne ➡ Ressource: Räume, Dozenten

Car Sharing ➡ Resourcen: Autos

Flughafenüberwachung ➡ Ressource: Luftraum, Landebahn(en)

Badezimmer in WGs ➡ Ressource - obvious

Scheduling - Ablaufplanung

Durch Scheduling wird eine Reihenfolge für die Ausführung von Prozessen (Threads, Tasks) festgelegt. Die Kriterien für die Reihenfolge können an verschiedenen Strategien orientiert sein.

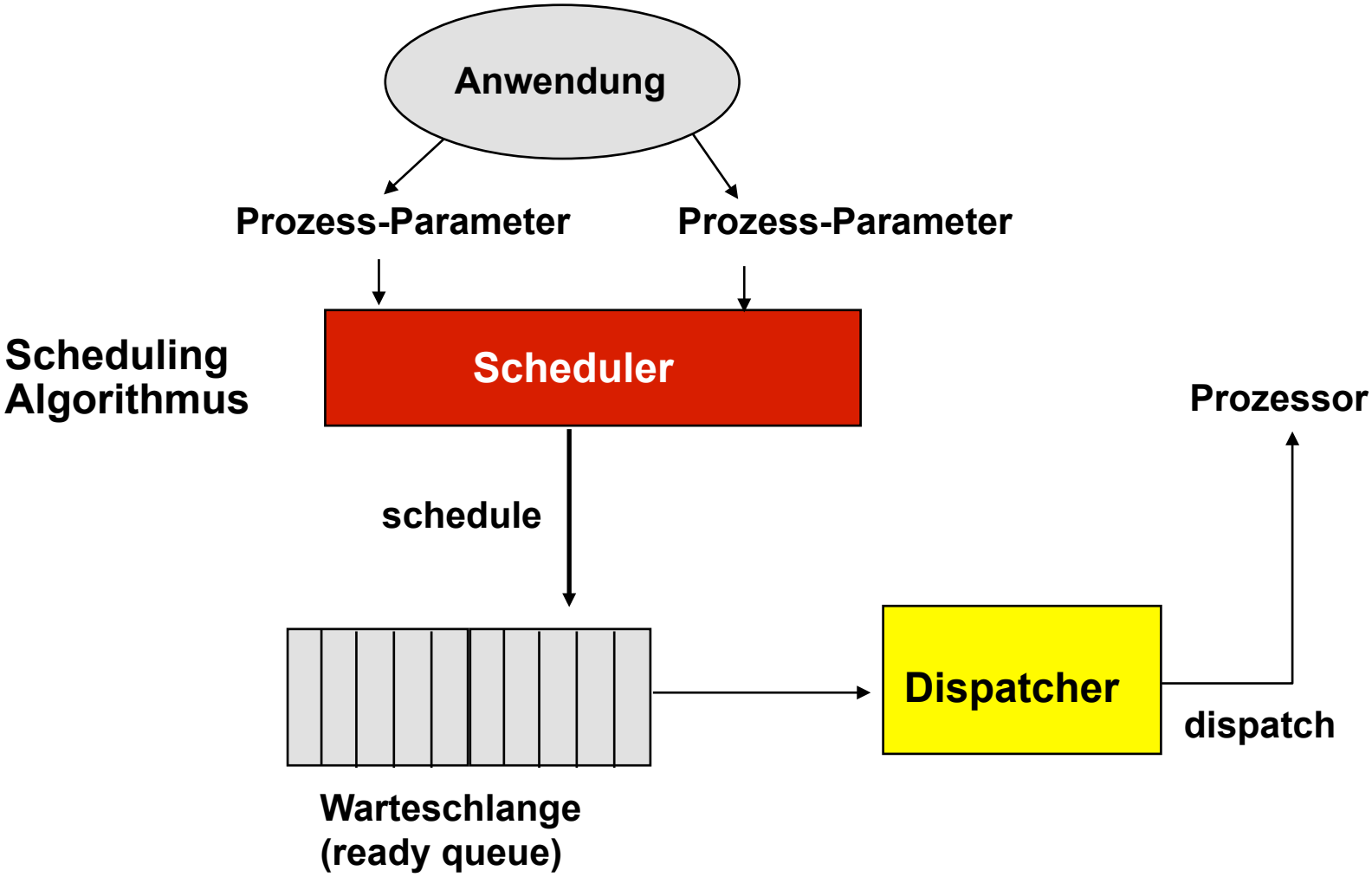
Der Scheduler verwaltet die Ressource "Prozessor".

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



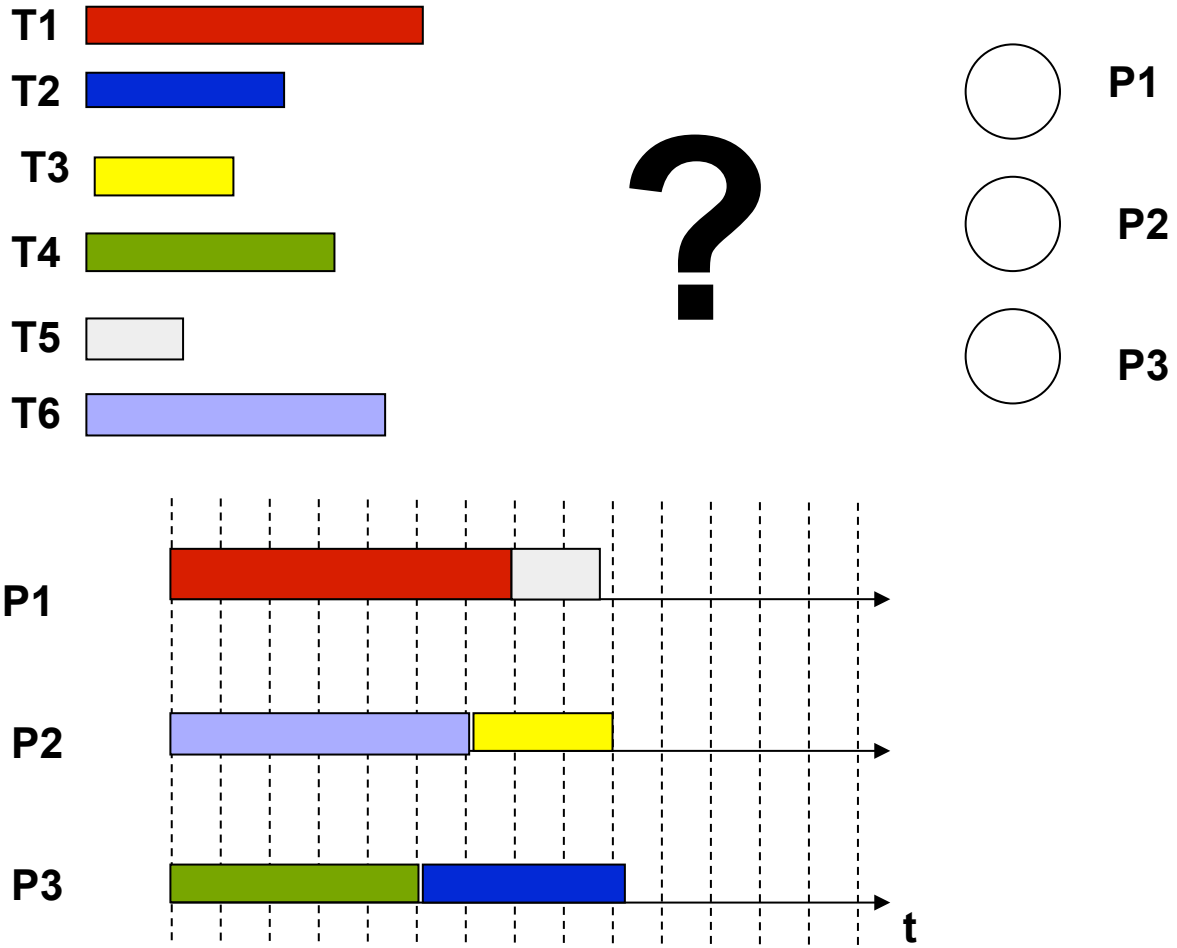
Scheduling - Beispiele

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

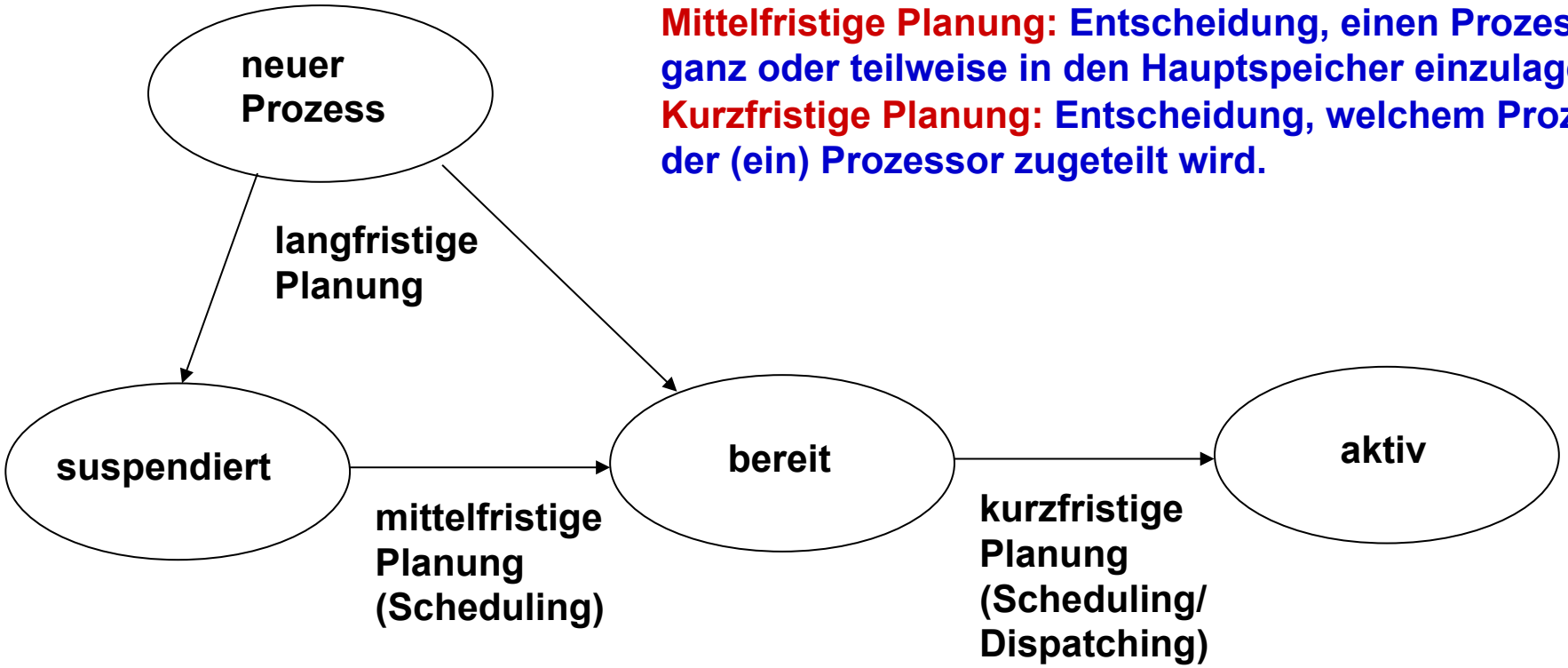
Minimierung der Länge des Ablaufplans
Minimierung der maximale Antwortzeit
Minimierung der mittleren (gewichtete) Antwortzeit
Minimierung der maximale Verspätung
Minimierung der Anzahl verspäteter Prozesse
...

Systemorientierte Ziele

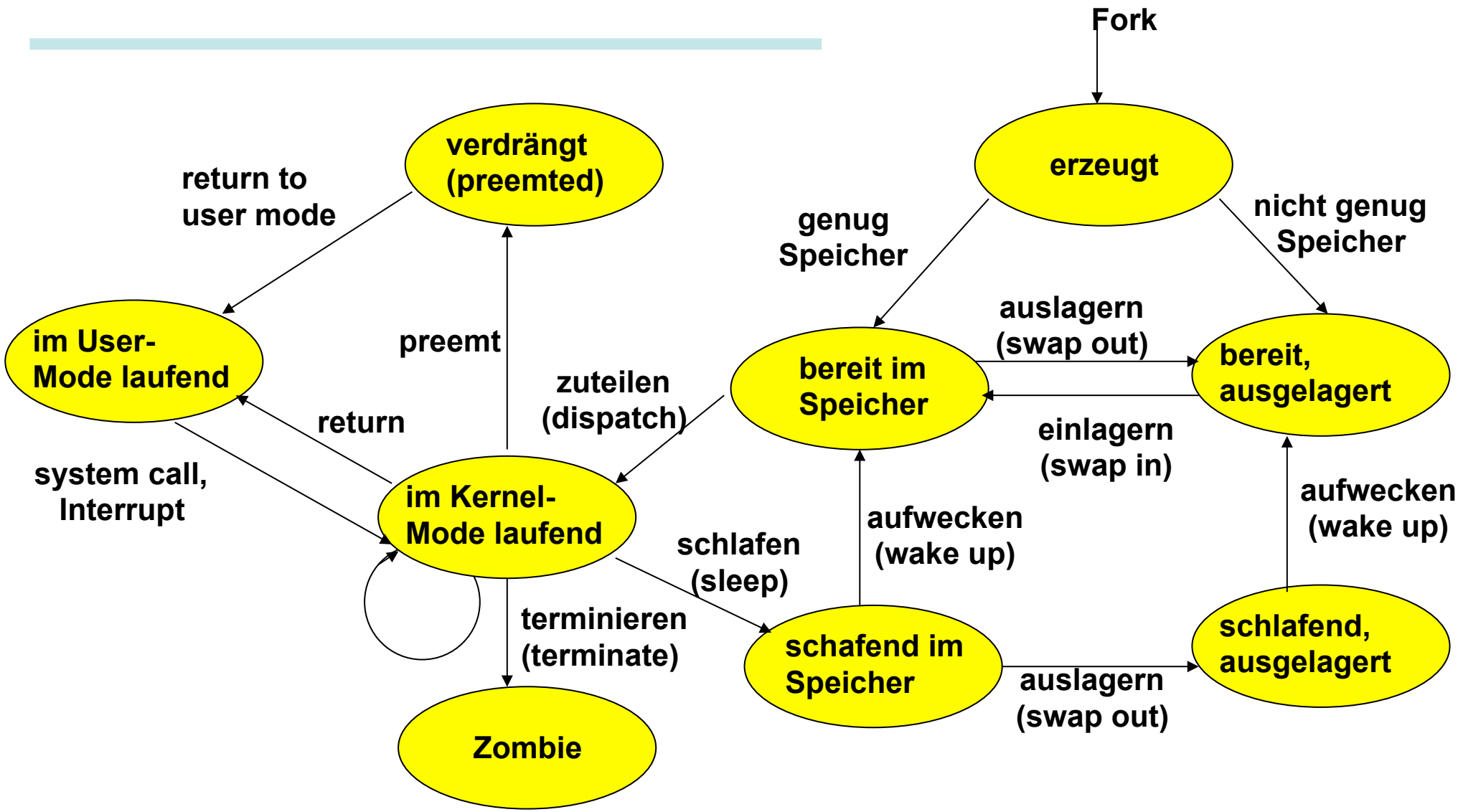
Maximaler Durchsatz
Maximale Prozessorauslastung
....

Arten des Scheduling

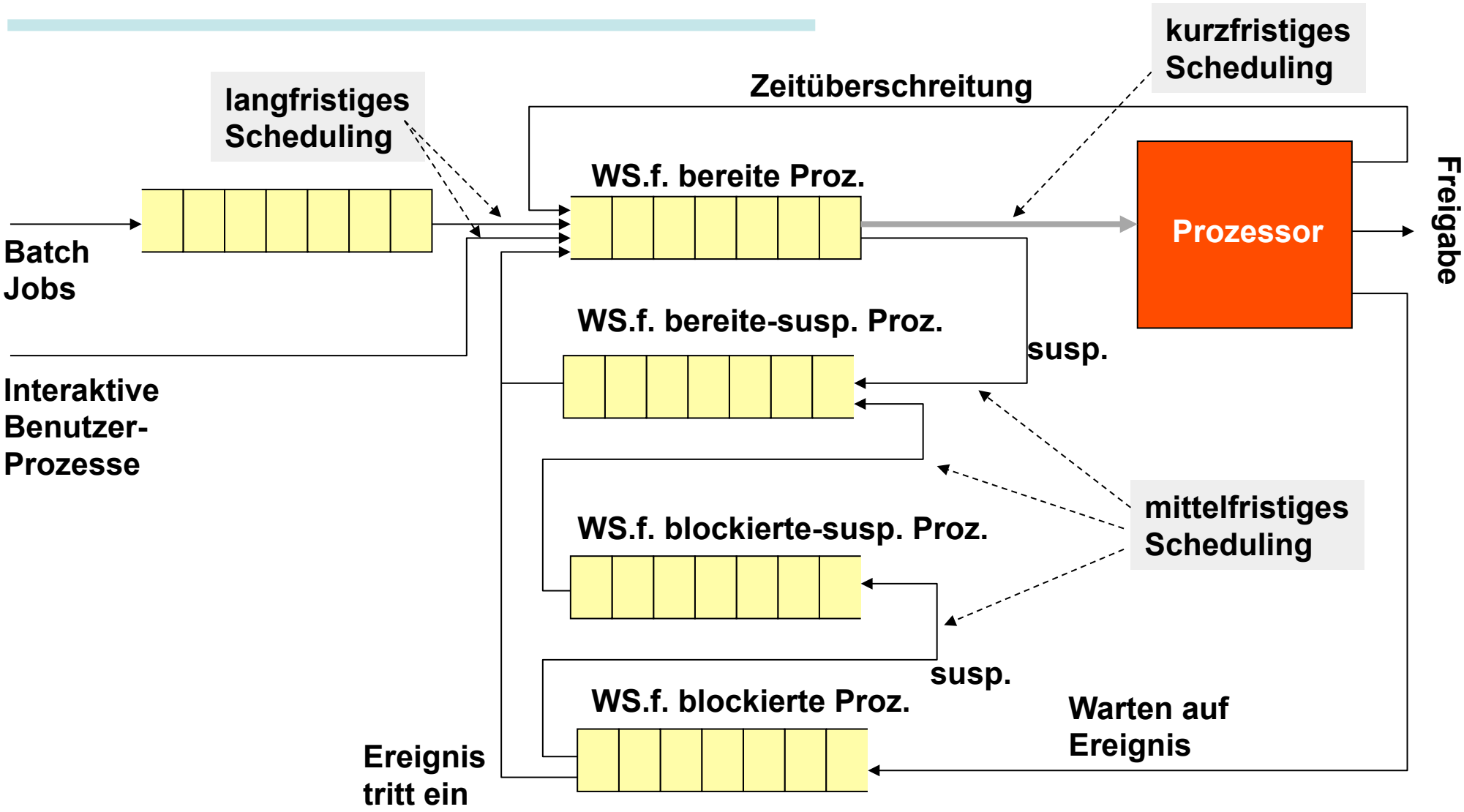
Langfristige Planung: Entscheidung, ob ein Prozess in die Menge der auszuführenden Prozesse aufgenommen wird.
Mittelfristige Planung: Entscheidung, einen Prozess ganz oder teilweise in den Hauptspeicher einzulagern.
Kurzfristige Planung: Entscheidung, welchem Prozess der (ein) Prozessor zugeteilt wird.



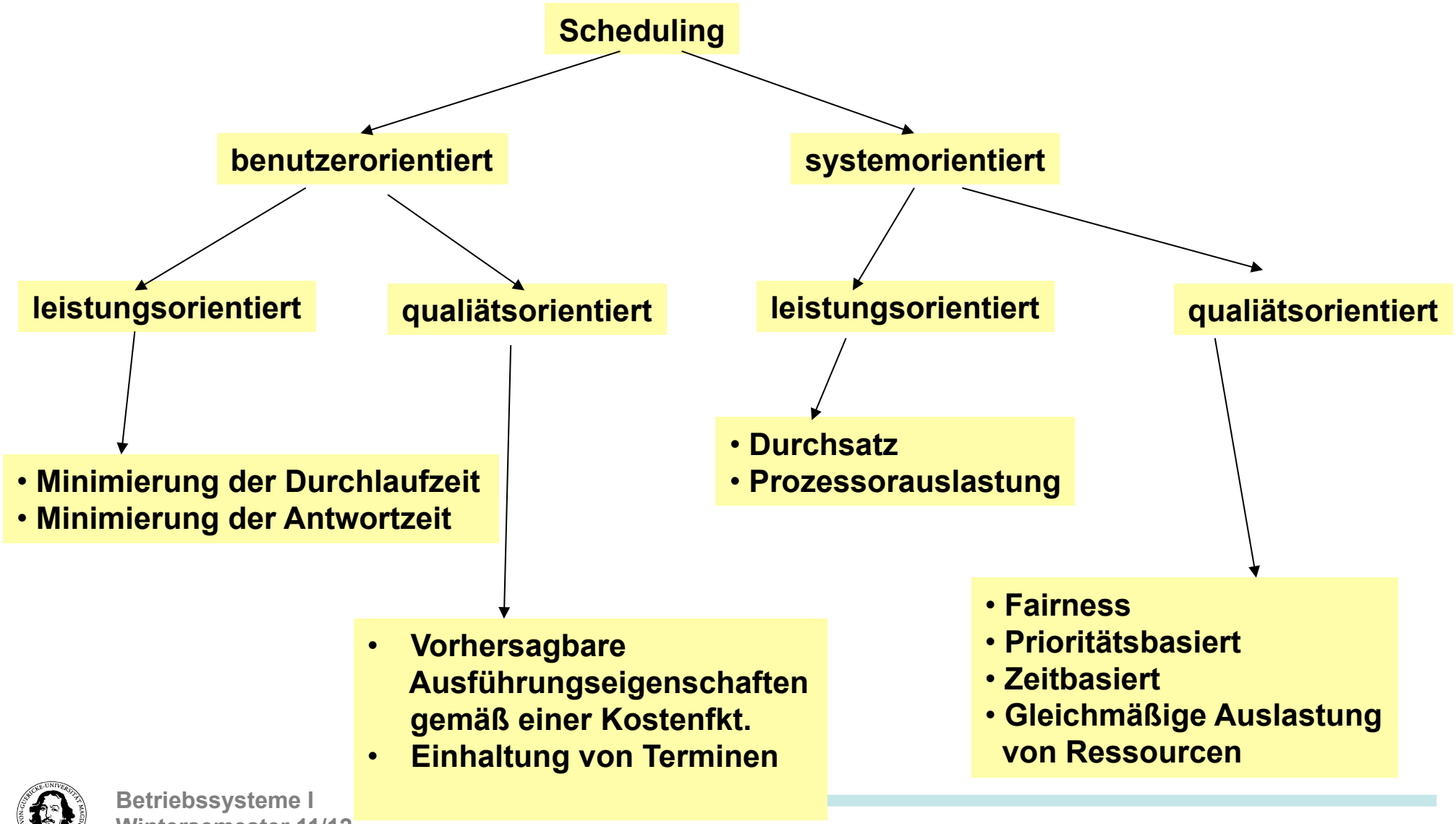
Unix Prozesszustände



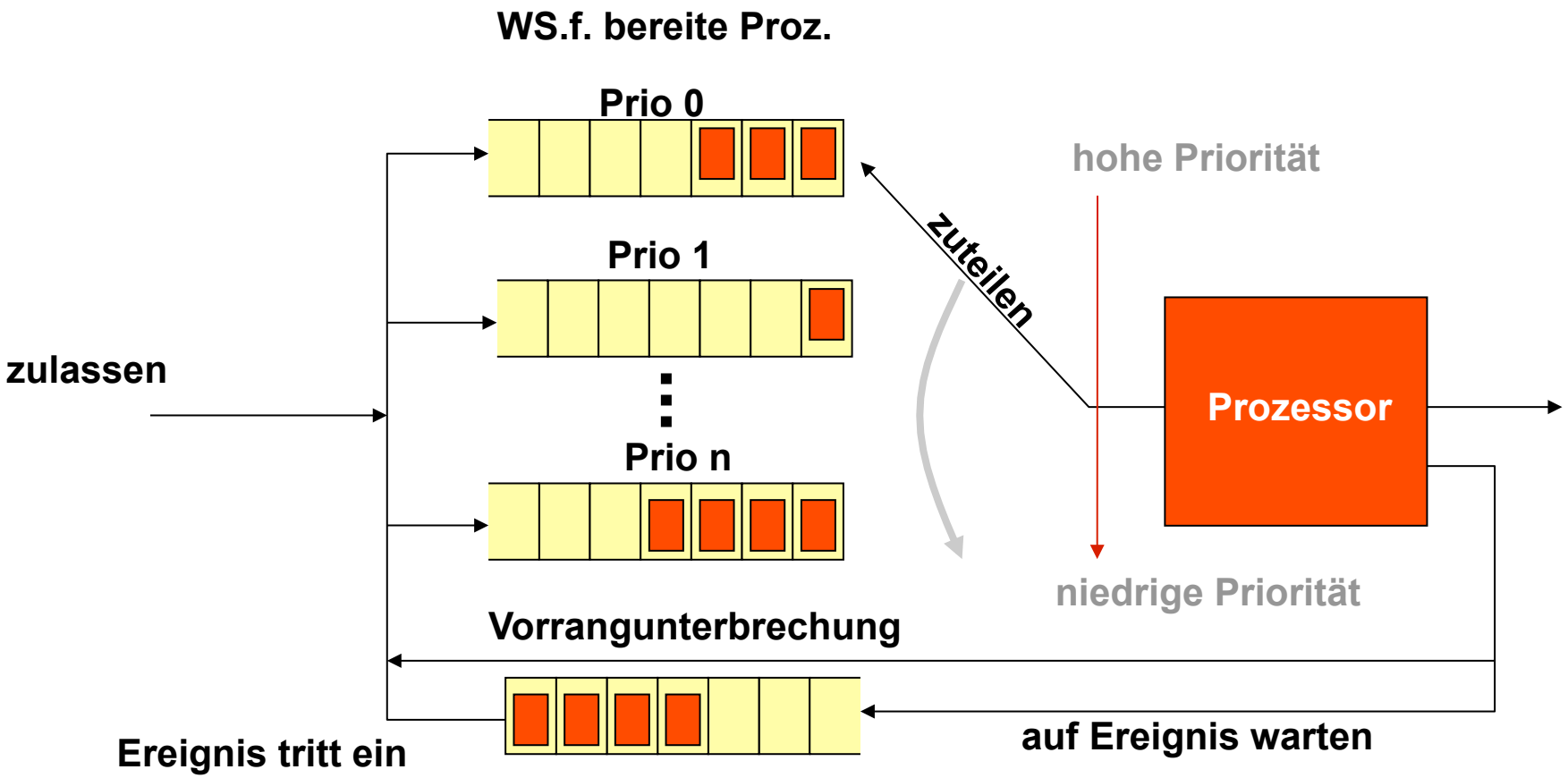
Warteschlangen (Queues)



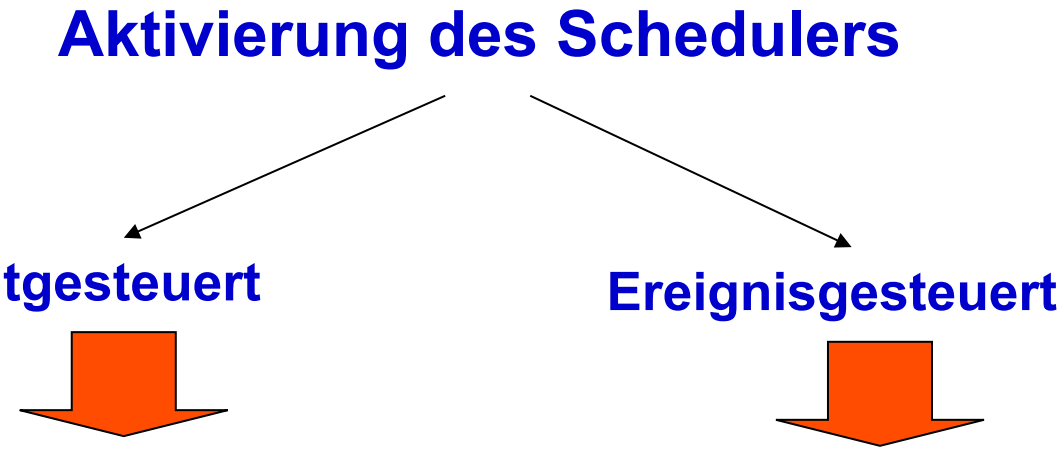
Ziele des Scheduling



Warteschlangen mit Prioritäten



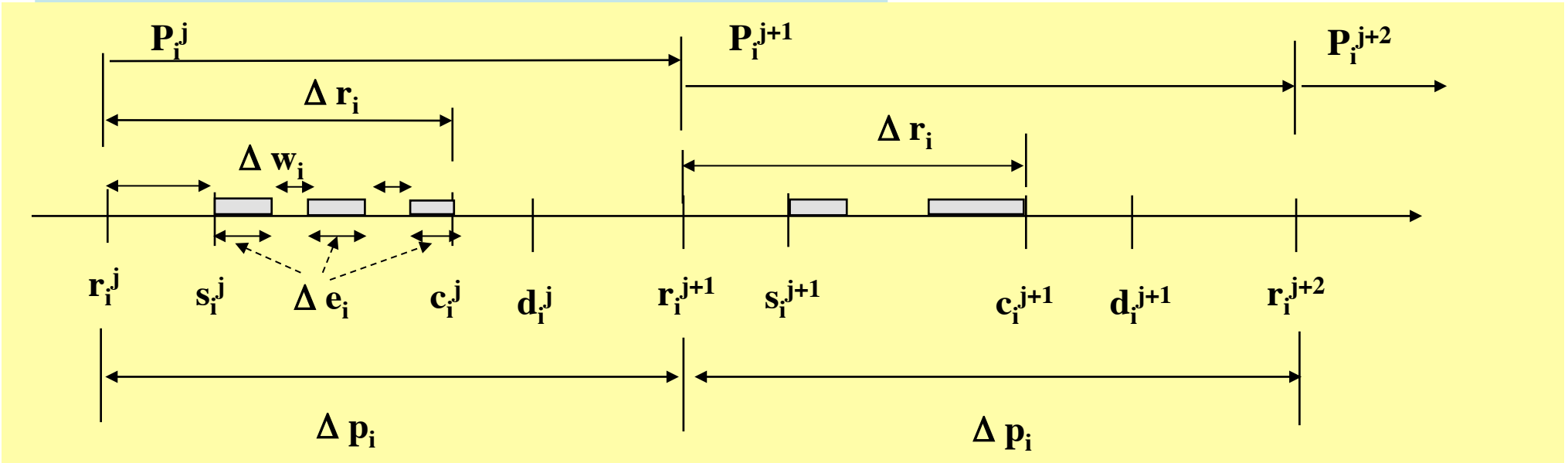
Wann wird der Scheduler aktiviert?



- Ablauf einer Zeitscheibe
- Zeitüberschreitung
- festgelegter Zeitpunkt

- Expliziter Aufruf
- Impliziter Aufruf (Abgabe d. Kontr.)
- Unterbrechung
- Systemaufruf (z.B. E/A)
- Prozess blockiert oder terminiert
- 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)

Δe_i : Ausführungszeit des Prozesses (execution time)

Δp_i: Periode des Prozesses

Δw_i : Wartezeit des Prozesses

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

$\Delta r_i / \Delta e_i$: Normalisierte Durchlaufzeit

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 augrund ihrer Ausführungsgeschichte und anderer Kriterien eingeordnet werden.

Standard Schedulingstrategien

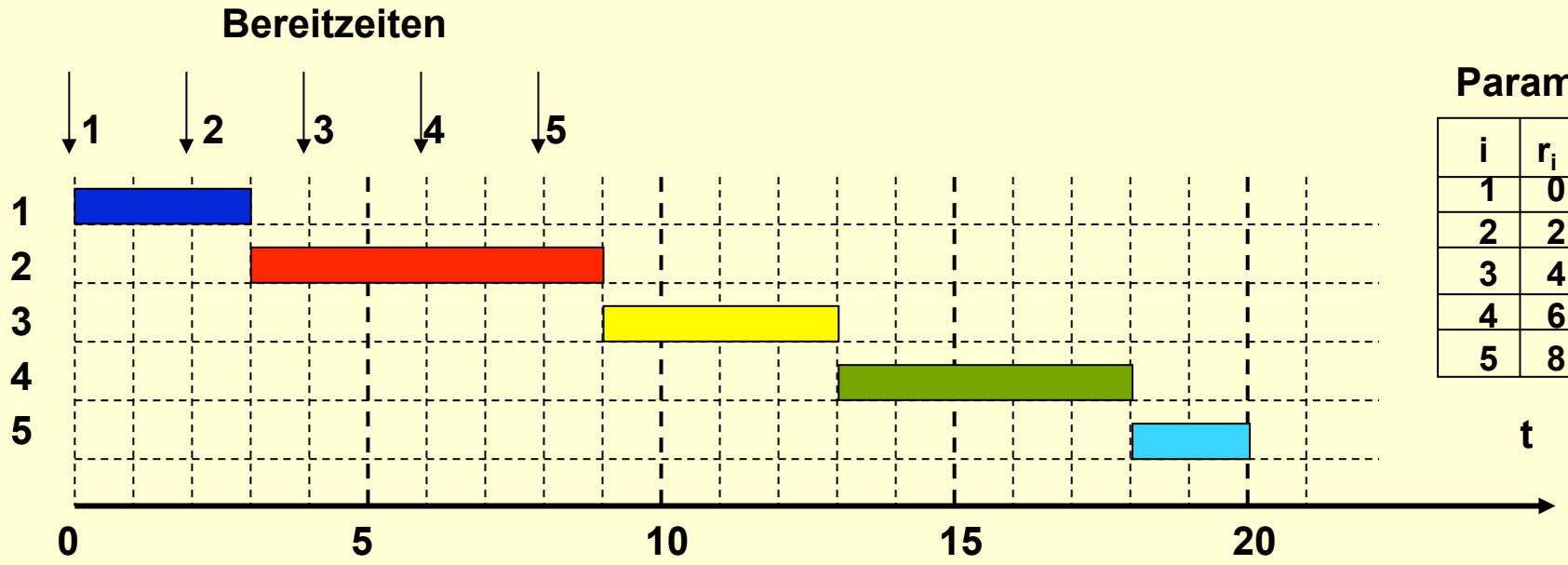
Prozessparameter für die folgenden Beispiele:

P-Nr.	Bereitzeit r_i	Ausführungszeit Δ e_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Prozess #	1	2	3	4	5	Mittelwert:
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

normalisierte Durchlaufzeit

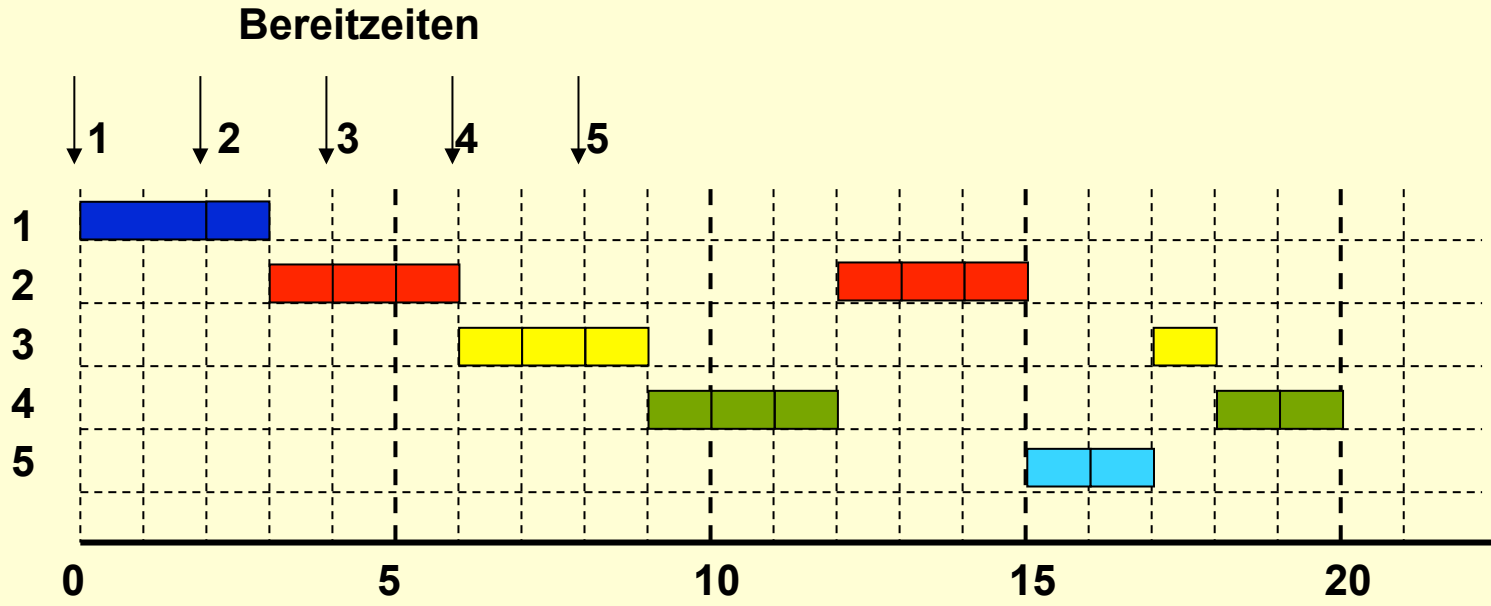
Beispiel für FCFS

Prozess	Bereitzeit		Bearbeitungszeit	Startzeit	Abschluss-	Durchlaufzeit	Norm. D-Zeit
	r_k	Δe_i	s_k	c_k	Δr_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

FCFS benachteiligt kurze Prozesse.

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

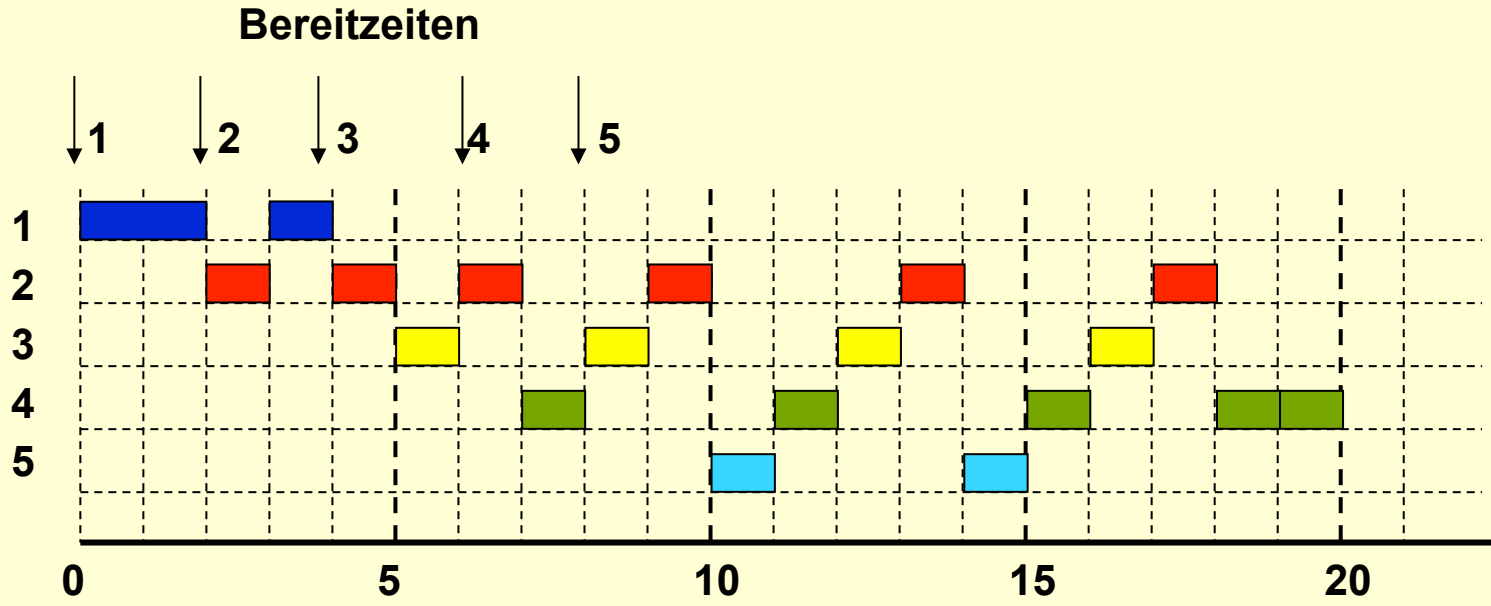
RR: q = 3

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

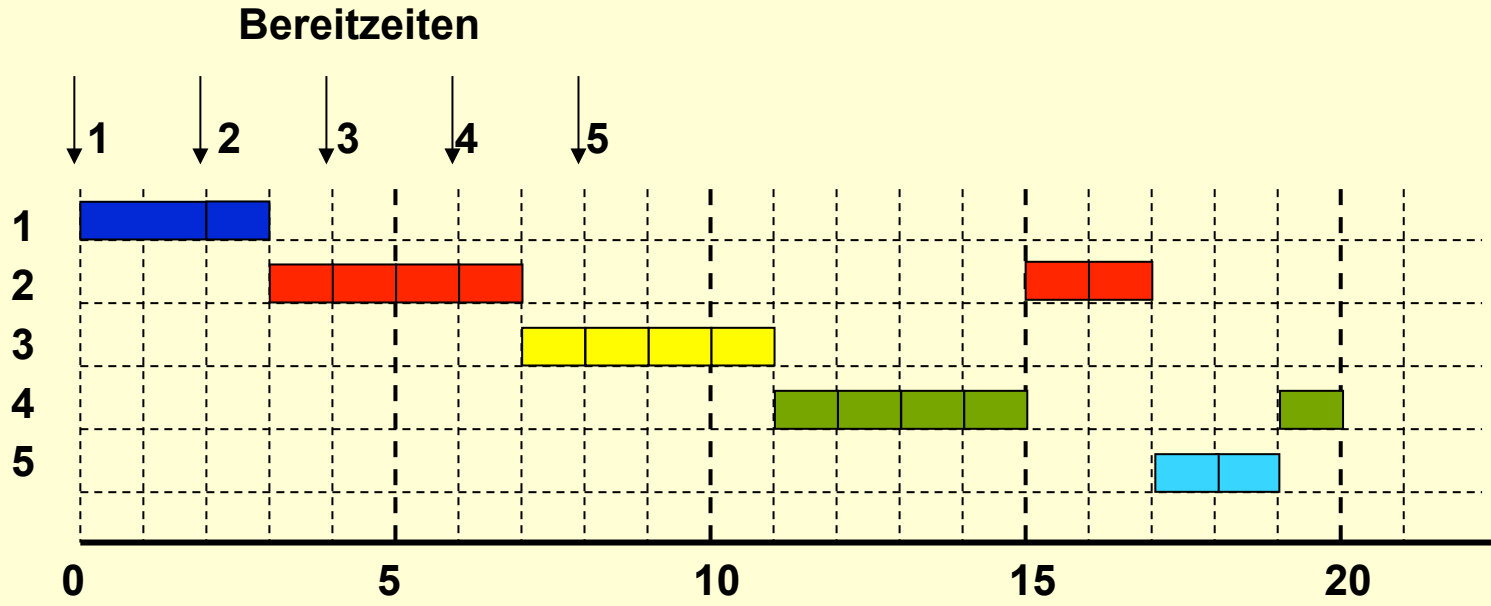
RR: q = 1

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

RR: q = 4

Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	19	11	20	17	
Durchlaufzeit	3	17	7	14	9	9,60
$\Delta r_i / \Delta e_i$	1,00	2,83	1,75	2,80	4,50	2,51

normalisierte Durchlaufzeit

Probleme mit RR

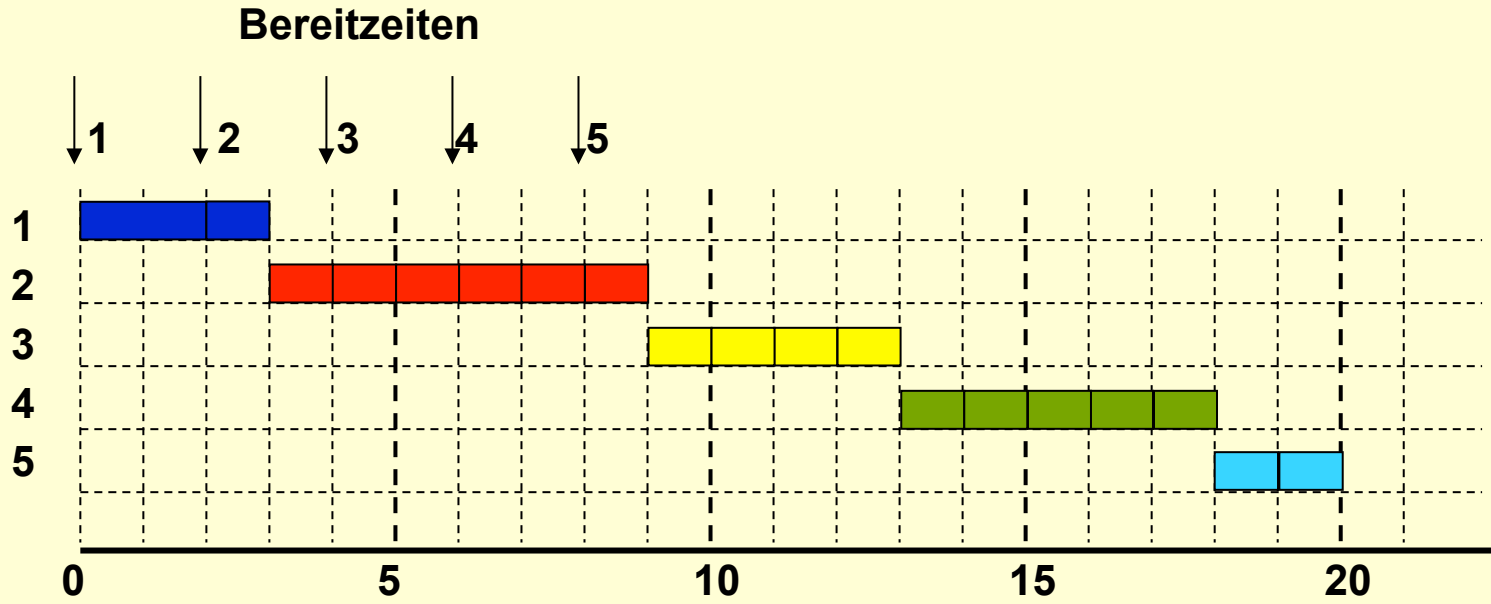
Verfahren ist abhängig von Zeitscheibenlänge

Zeitquantum > durchschnittl. Durchlaufzeit: **Kurze Prozesse werden benachteiligt.**

Zeitquantum < durchschnittl. Durchlaufzeit: **Viele Scheduler-Aufrufe**

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

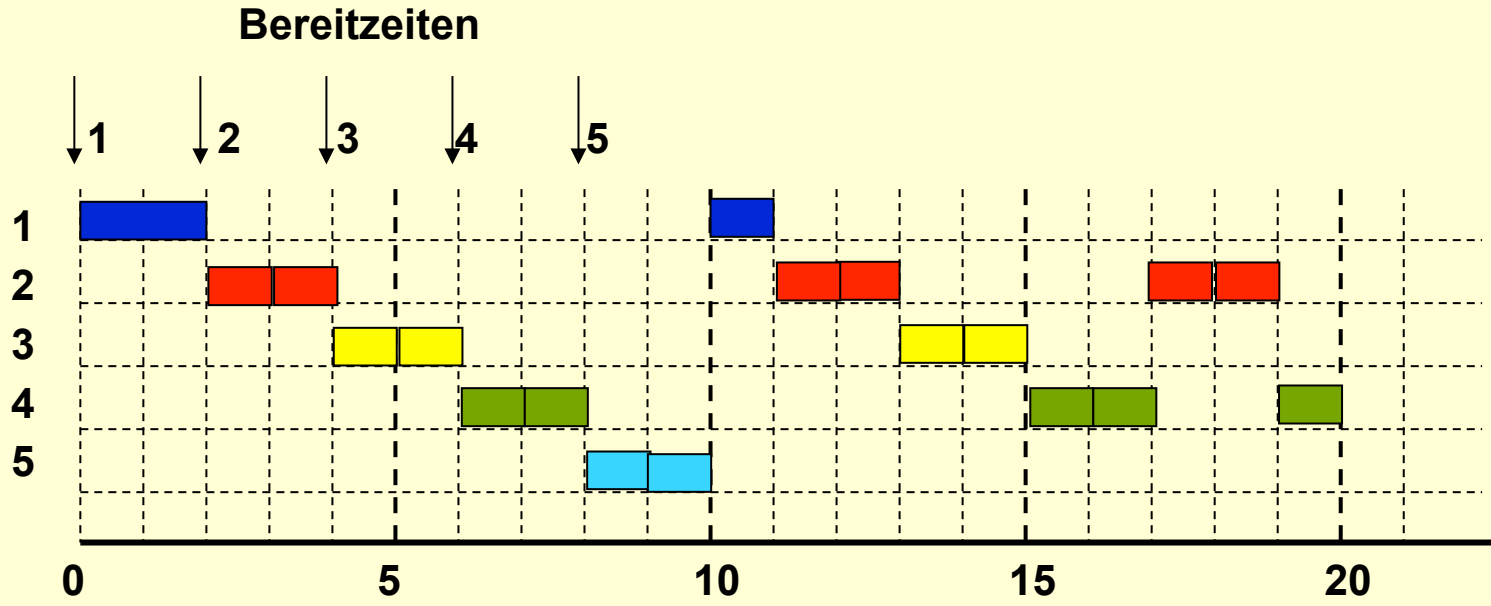
RR: q = 6

Prozess #	1	2	3	4	5	Mittelwert:
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

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

RR: q = 2

Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	11	19	15	20	10	
Durchlaufzeit	11	17	11	14	2	11
$\Delta r_i / \Delta e_i$	3,70	2,83	2,75	2,80	1,00	2,62

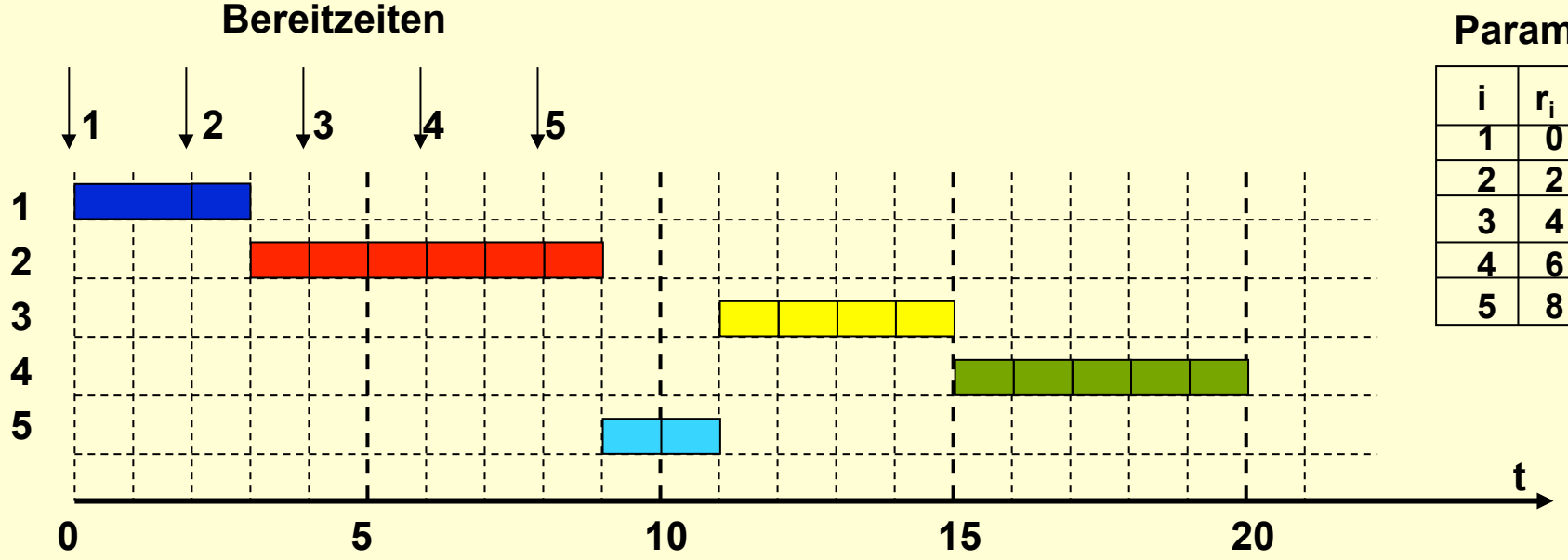
normalisierte Durchlaufzeit

Bisher werden bein Scheduling lediglich die Bereitzeiten berücksichtigt

Ausführungsdauern werden nicht betrachtet!!!

SPN - Shortest Process Next

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



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

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)

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

Δr_i: tatsächliche Ausführungszeit der i-ten Ausführung

S_{n+1}: Abschätzung der (n+1)ten Ausführung

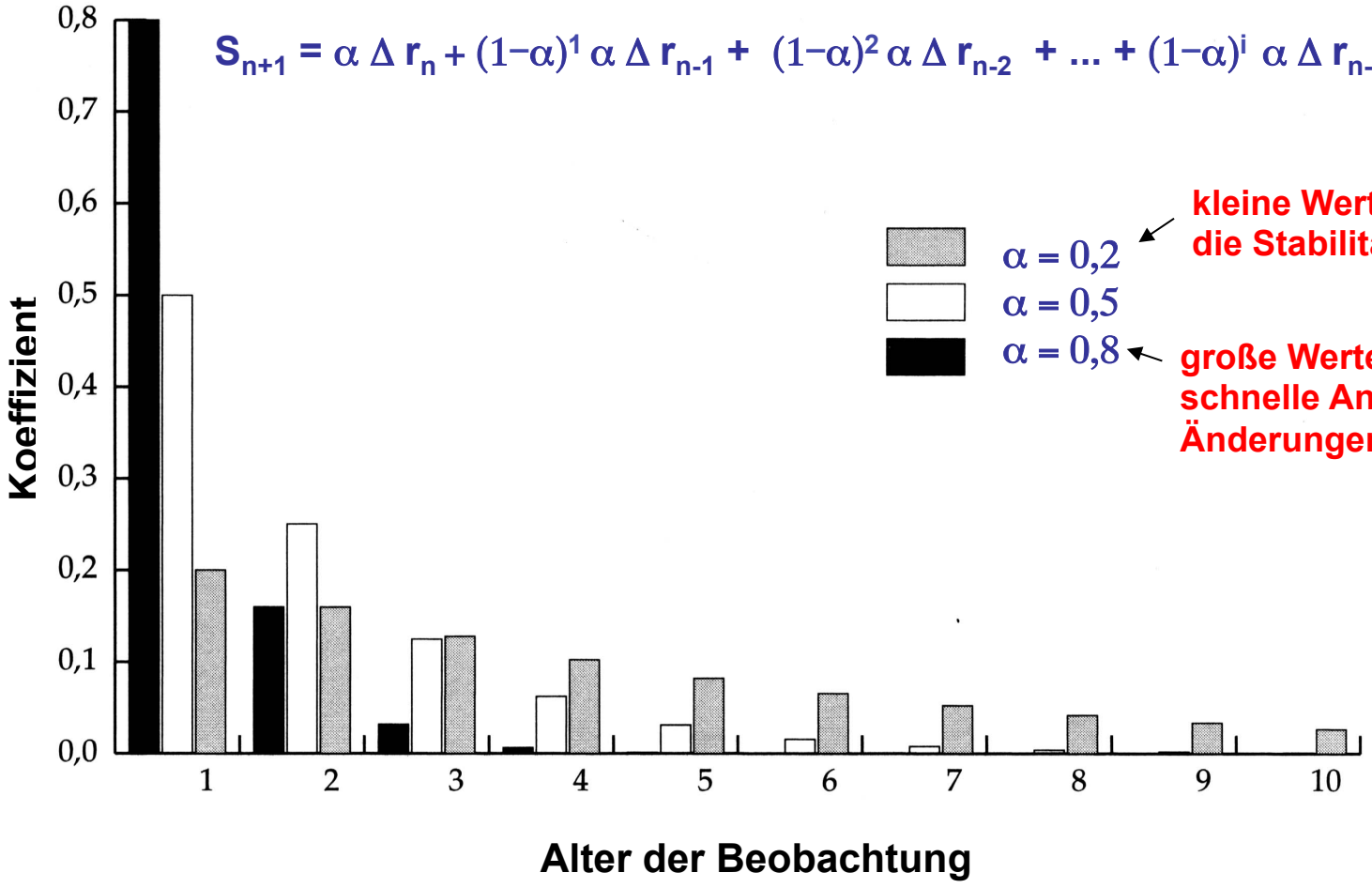
Lässt sich umstellen zu: $S_{n+1} = ((n-1)/n)S_n + (1/n)\,\Delta r_n$

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

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

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

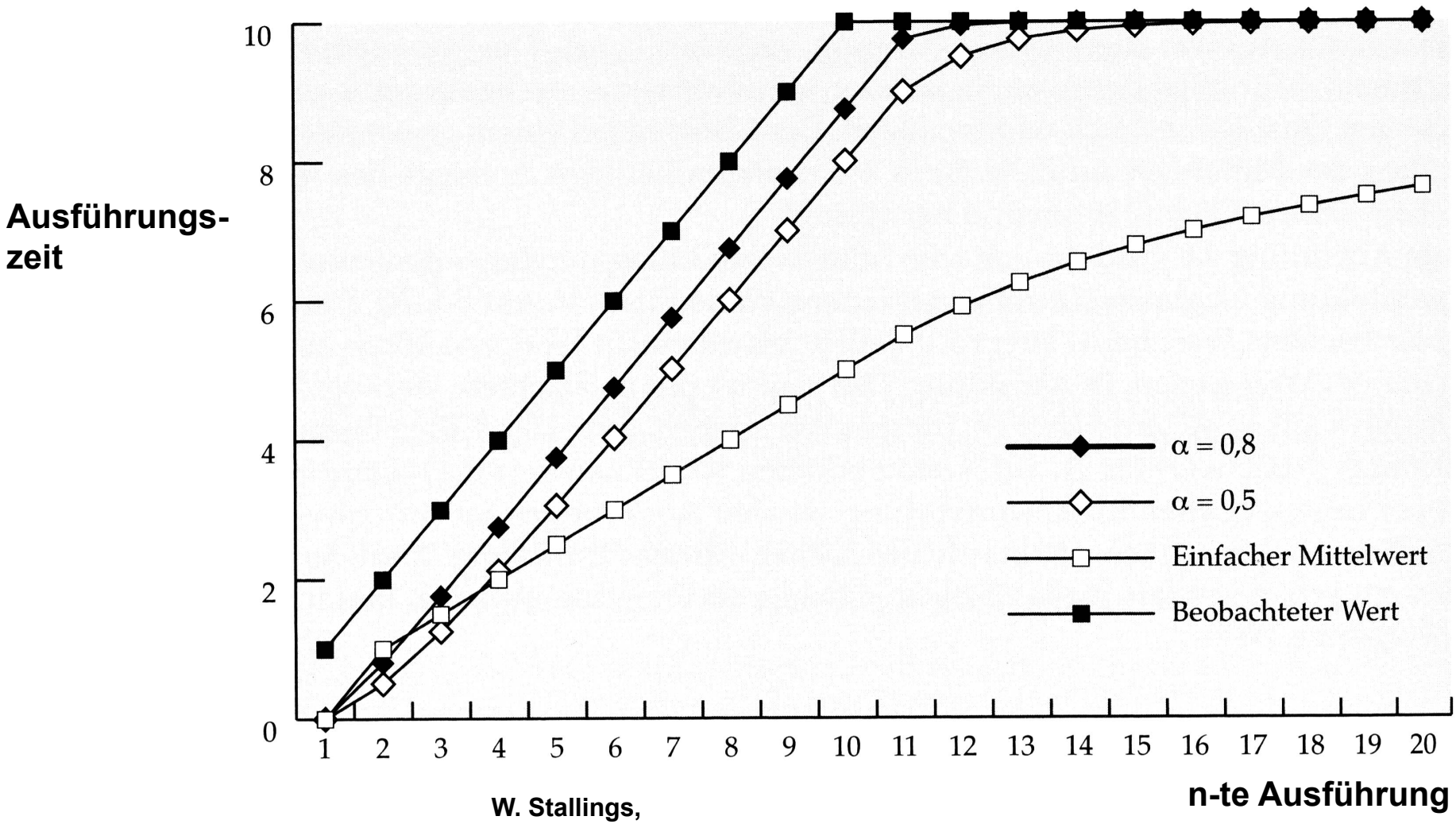


kleine Werte für α erhöhen die Stabilität der Schätzung

große Werte für α erlauben schnelle Anpassung an Änderungen

W. Stallings, Moderne BS, Kap. 9

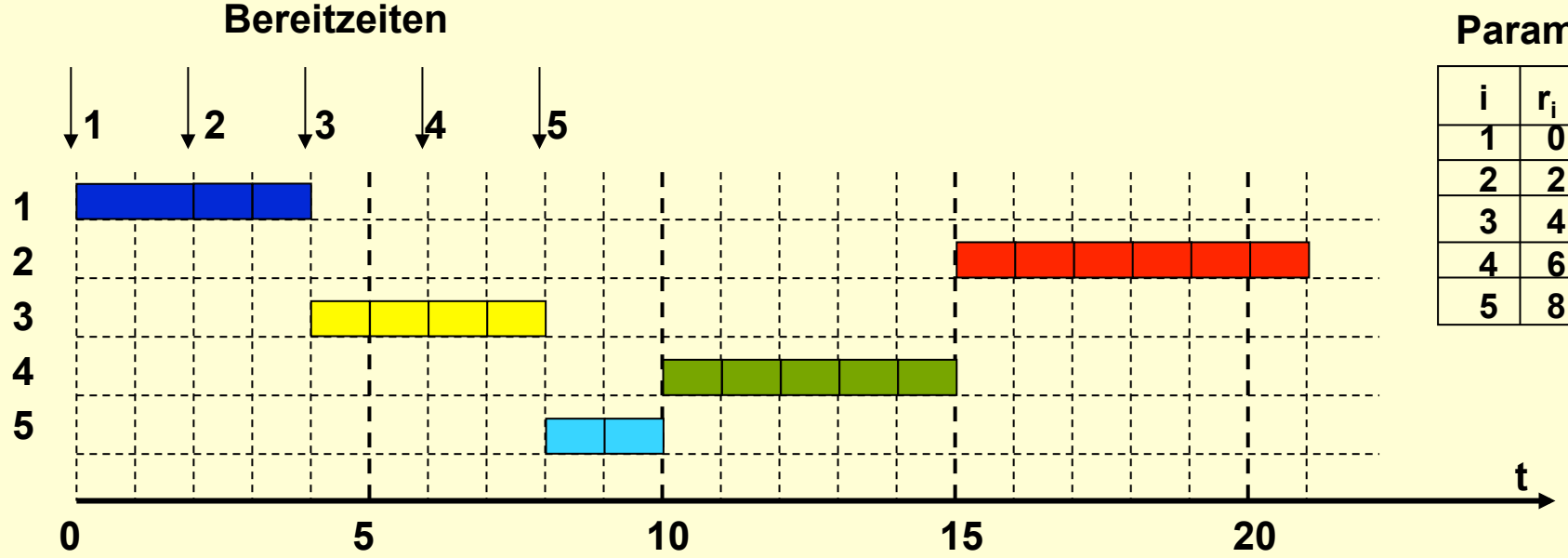
Vergleich der Verfahren zur Mittelwertbildung



W. Stallings, Moderne BS, Kap. 9

Probleme mit SPN

Lange Prozesse werden stark benachteiligt !



Parameter:

i	r_i	Δe_i
1	0	4
2	2	6
3	4	4
4	6	5
5	8	2

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

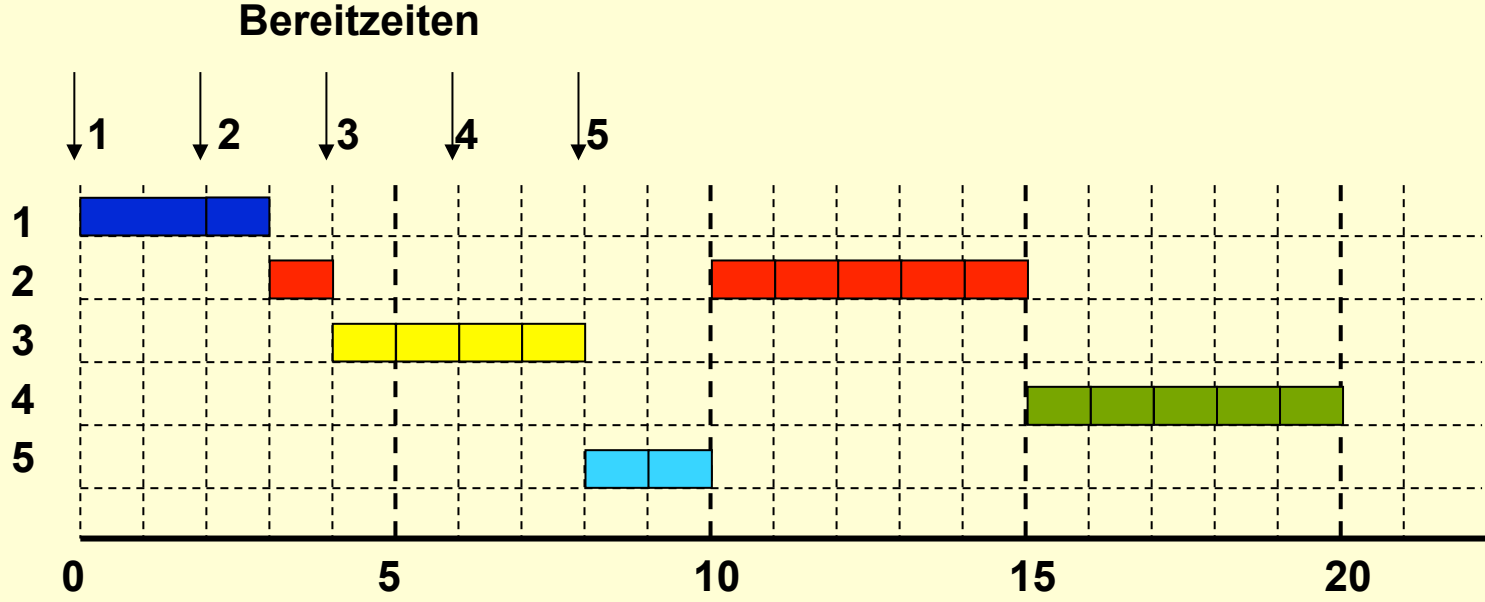
Probleme mit SPN

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.



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

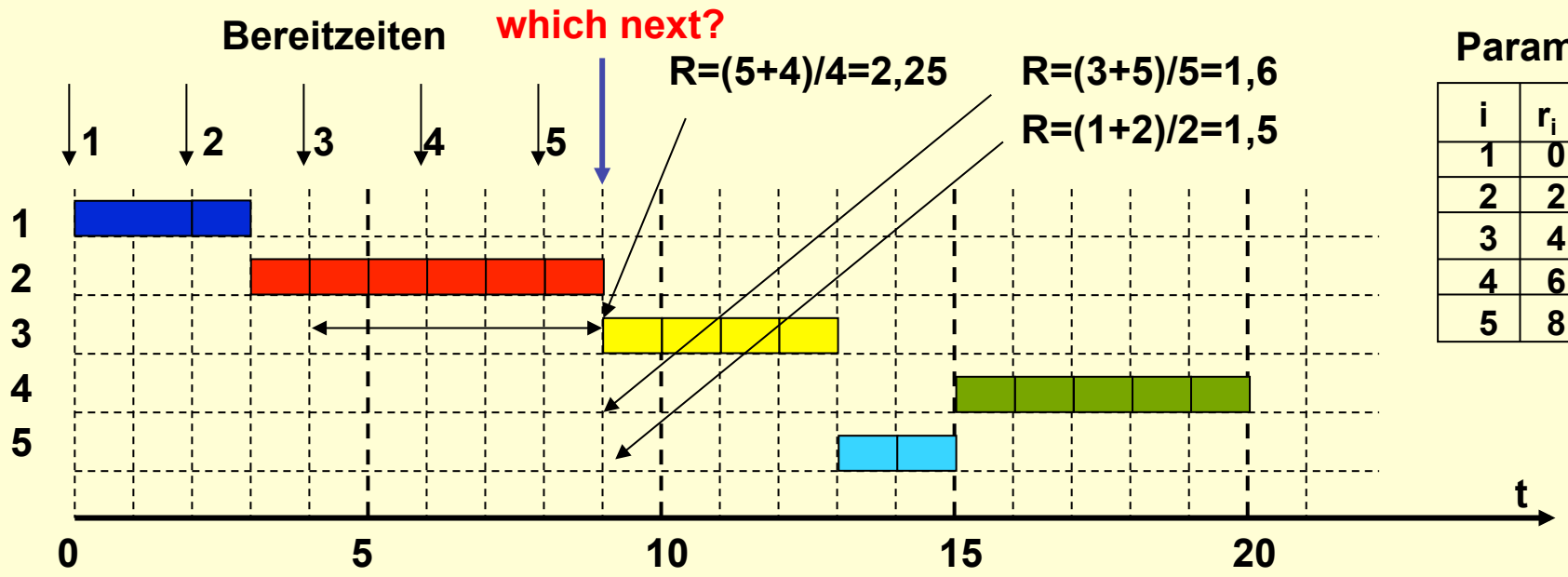
Prozess #	1	2	3	4	5	Mittelwert:
Abschlusszeit	3	15	8	20	10	
Durchlaufzeit	3	13	4	14	2	7,20
$\Delta r_i / \Delta e_i$	1,00	2,17	1,00	2,80	1,00	1,59

normalisierte Durchlaufzeit

HRRN - Highest Response Ratio Next

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

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



Parameter:

i	r_i	Δe_i
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

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

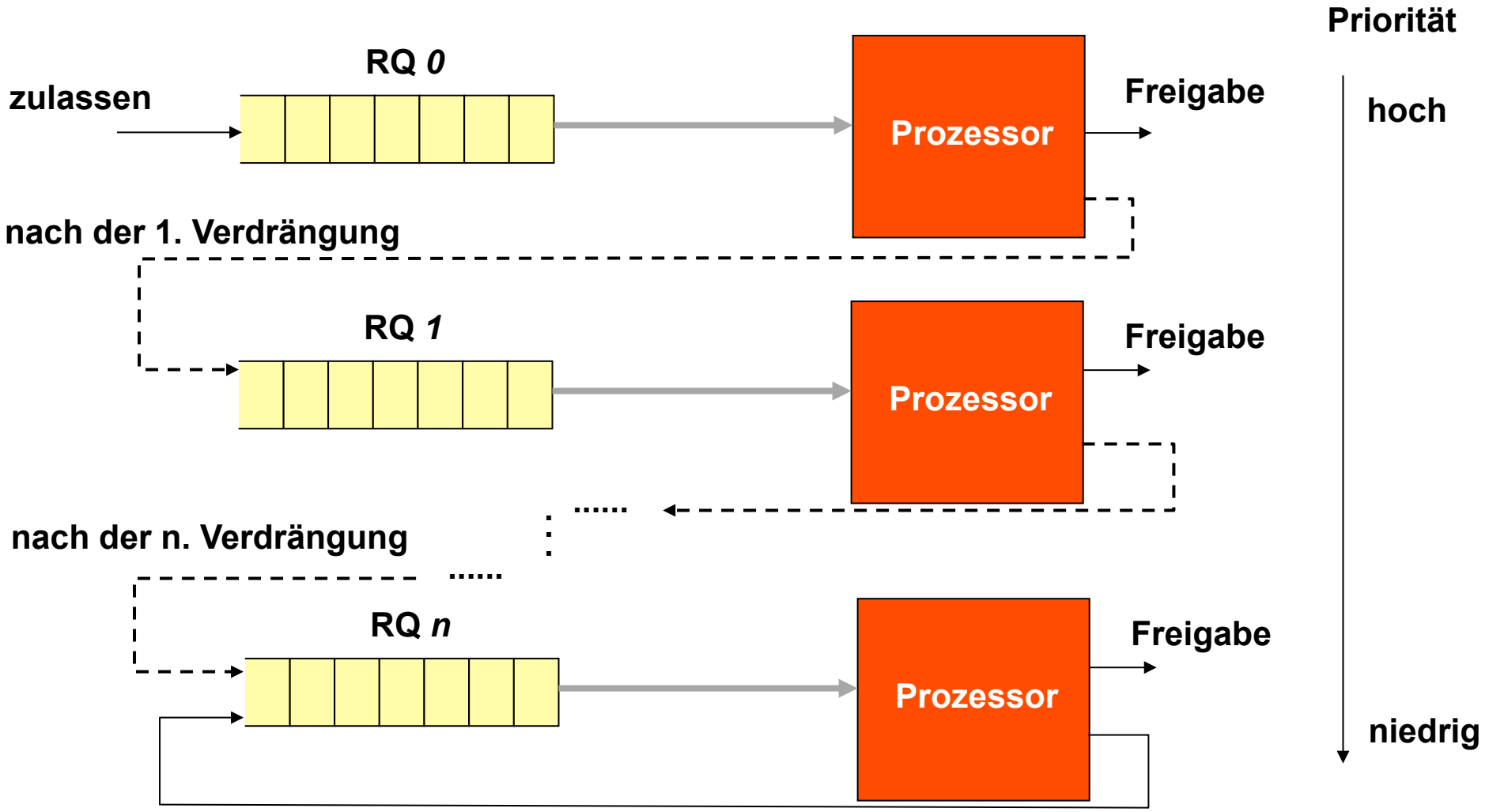
normalisierte Durchlaufzeit

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,..,n-1 wird das FCFS-Verfahren angewendet. Jeder Prozess bekommt ein festes Zeitquantum**
- **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*

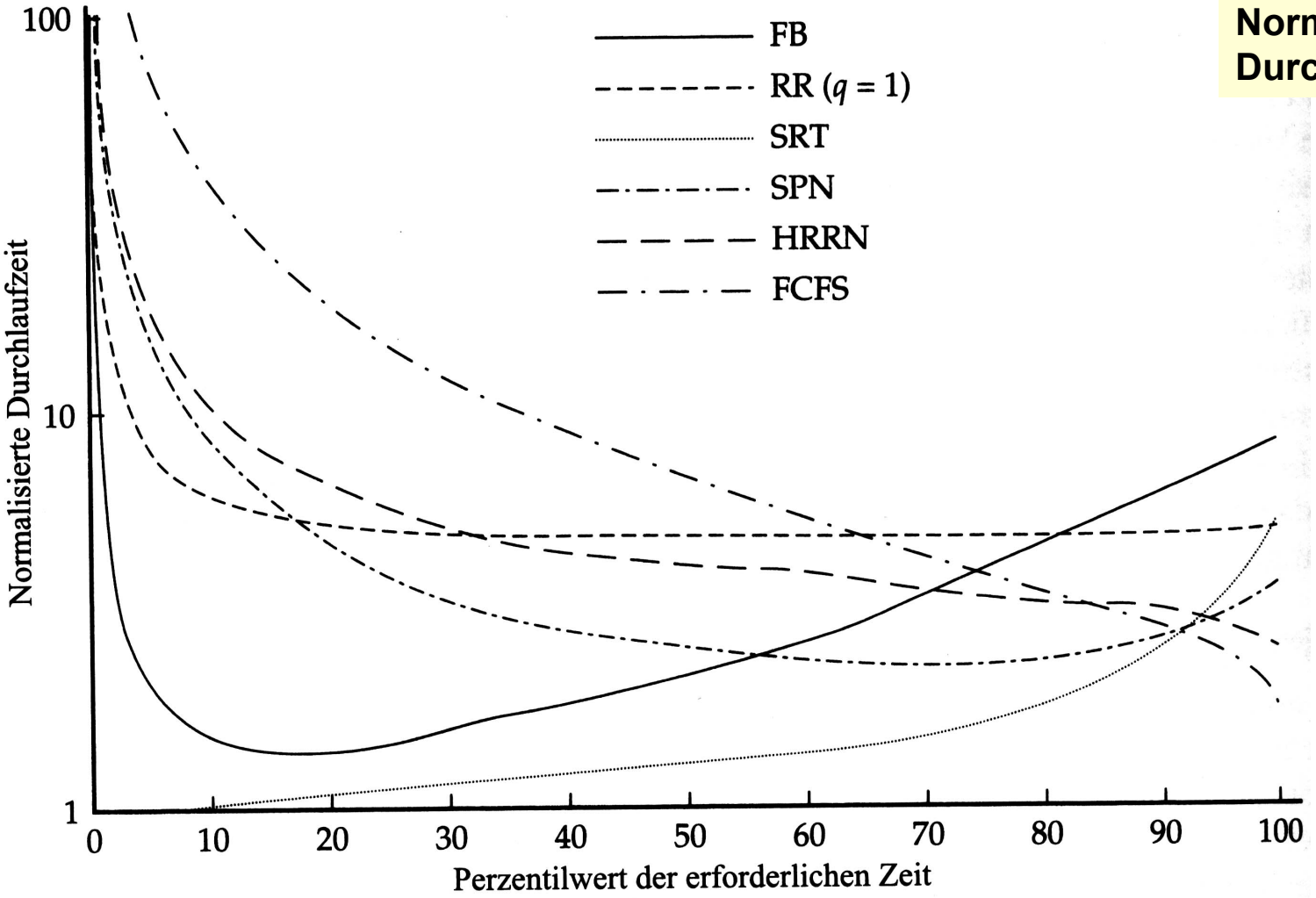
Vergleich der Scheduling Strategien

	Auswahl-funktion	Entscheidungsmodus	Durchsatz	Antwortzeit	Overhead	Auswirkung auf Prozesse	Verhun-gern
FCFS	max[*w*]	Nicht unterbrechend	Spielt keine Rolle	Kann hoch sein, ins-besondere, bei großer Varianz der Prozess-ausführungszeiten	Minimal	Benachteiligt kurze Prozesse; benach-teiligt E/A-lastige Prozesse	Nein
Round Robin	Konstante	Unterbrechend (nach bestimmter Zeitdauer)	Ist gering, wenn Zeitdau-er gering ist.	Bietet bei kurzen Prozessen gute Antwortzeit	Minimal	Faire Behandlung	Nein
SPN	min [Δe]	Nicht unterbrechend	Hoch	Bietet bei kurzen Prozessen gute Antwortzeit	Kann groß sein	Benachteiligt lange Prozesse	Möglich
SRT	min [ΔR]	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

[ΔR]= Gesamtausführungszeit - bereits verwendete Zeit

W. Stallings, Moderne BS, Kap. 9

Vergleich der Schedulingstrategien

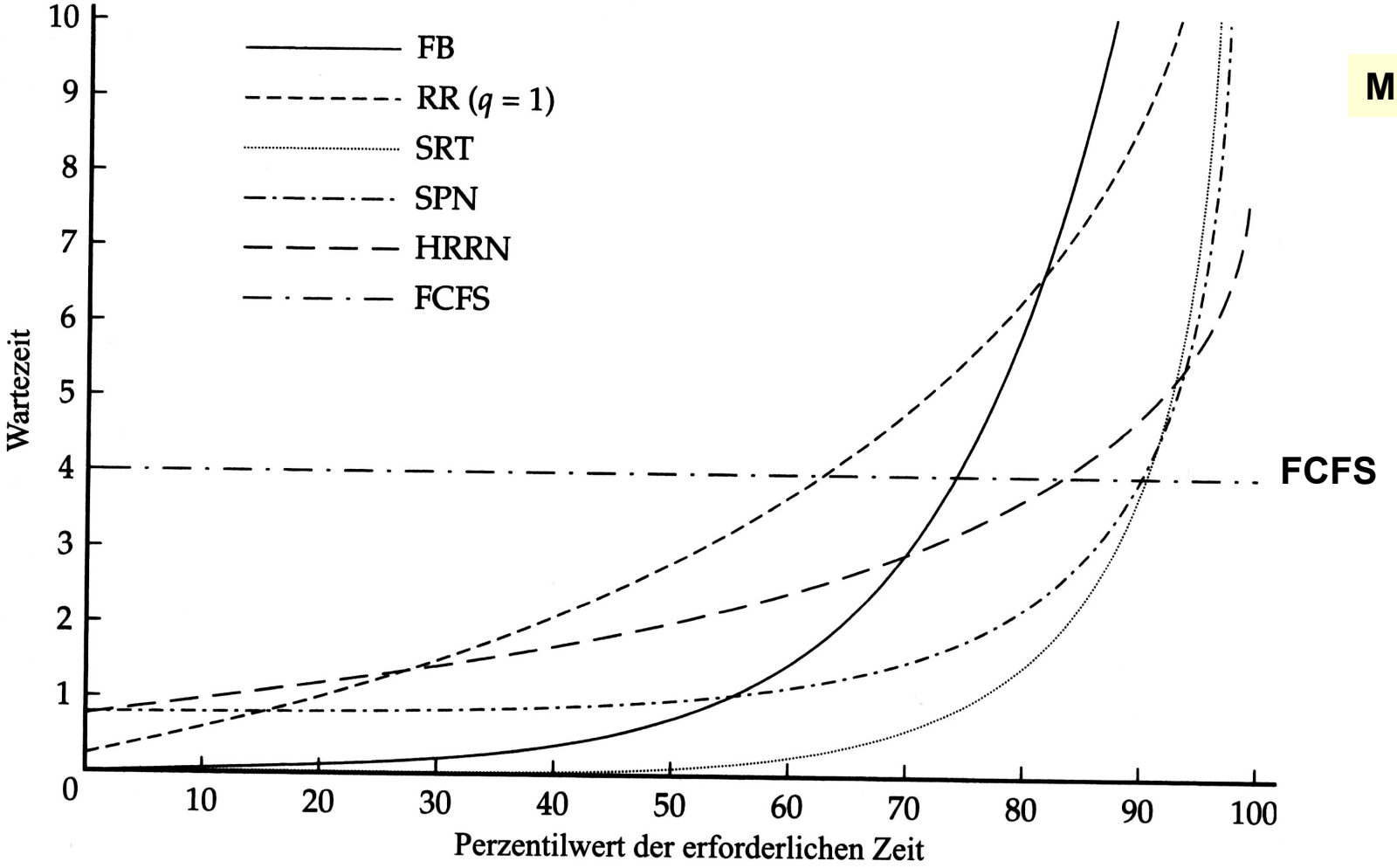


Normalisierte Durchlaufzeit

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

W. Stallings, Moderne BS, Kap. 9

Vergleich der Schedulingstrategien



Mittlere Wartezeit

FCFS

W. Stallings, Moderne BS, Kap. 9

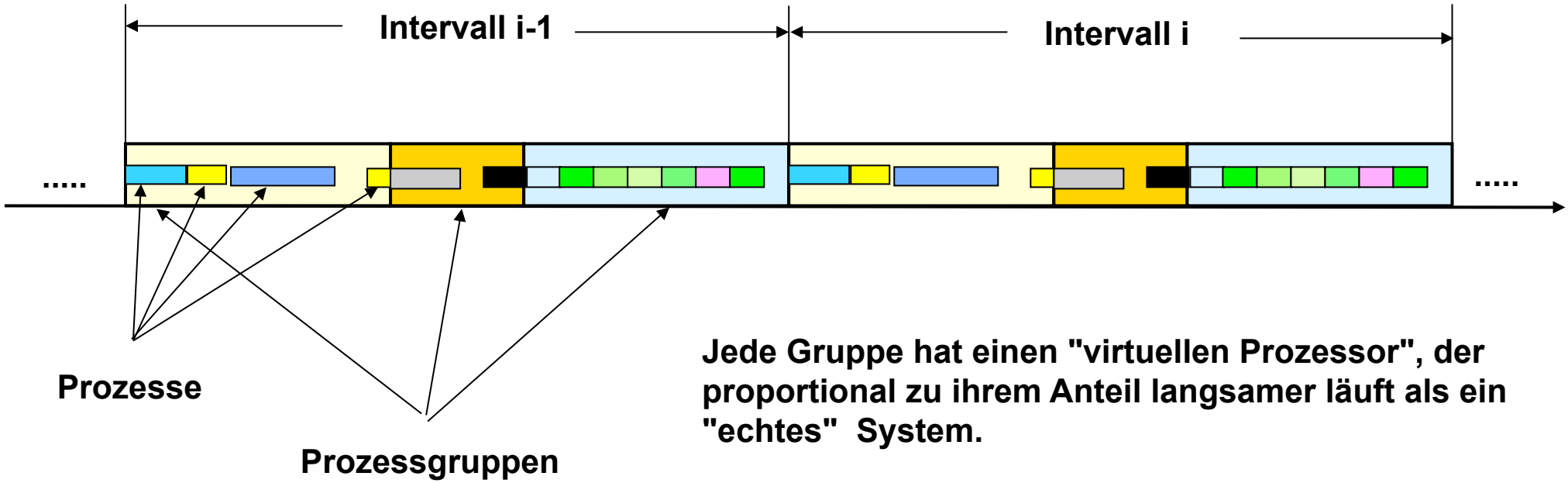
Standard Schedulingstrategien

- **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.

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



Jede Gruppe hat einen "virtuellen Prozessor", der proportional zu ihrem Anteil langsamer läuft als ein "echtes" System.

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 $\Sigma W_k = 1$

Fair-Share-Scheduling

"Fair" beinhaltet:

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

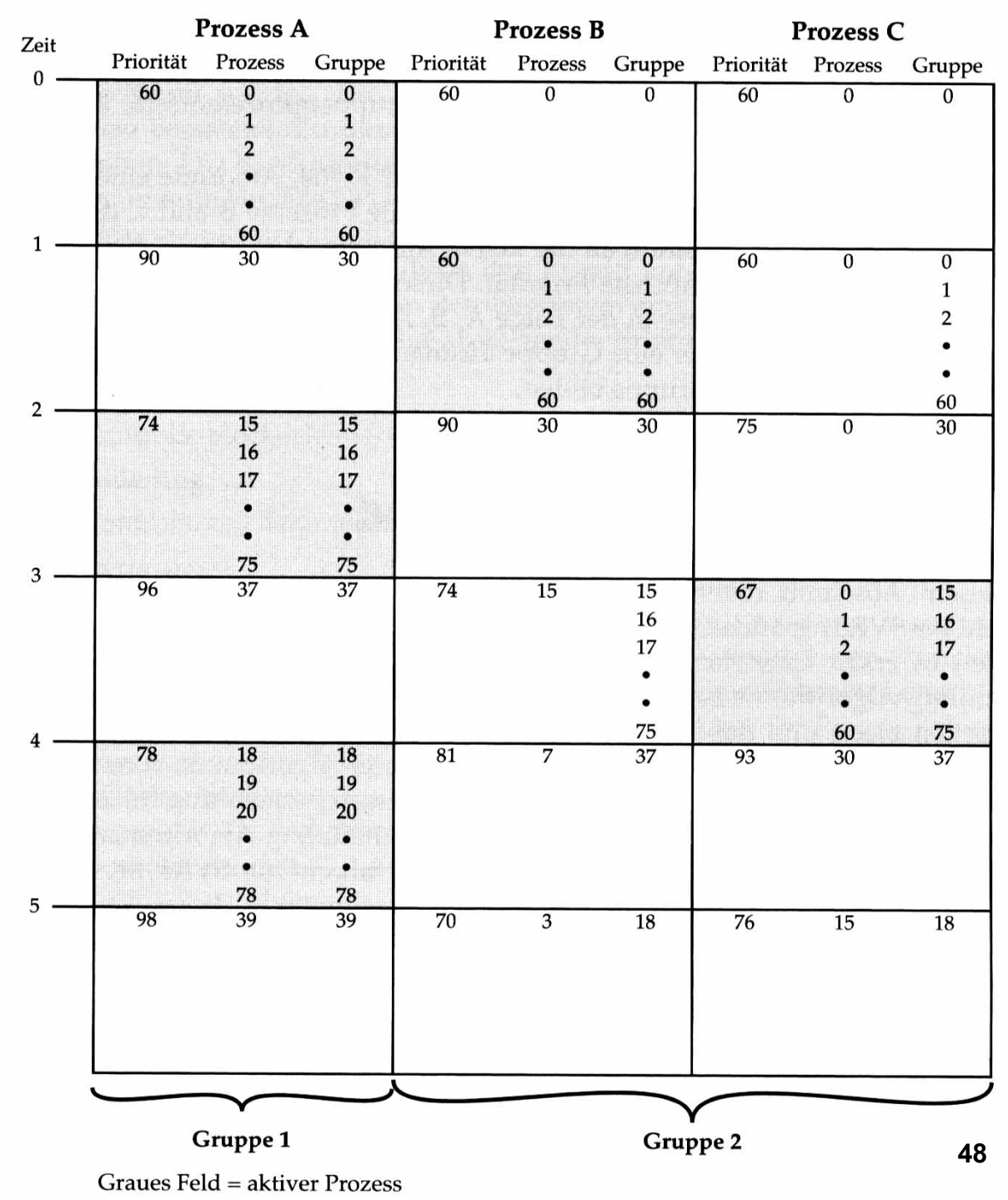
Für Prozess *j* der Gruppe *k* gilt dabei die Festlegung:

- $CPU_j\ (i) = CPU_j\ (i\text{-}1) / 2$ — CPU Auslastung von Prozess *j* im Intervall *i*.
- $GCPU_k\ (i) = GCPU_k\ (i\text{-}1) / 2$ — CPU Auslastung von Gruppe *k* im Intervall *i*.
- $P_j\ (i) = Basis + CPU_j\ (i\text{-}1) / 4 + GCPU_k\ (i\text{-}1) / 8 \times W_k$

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

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

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

Gruppe 1 Gruppe 2

Graues Feld = aktiver Prozess

Beispiele:

- Klassisches Unix Scheduling
- Linux
- Unix SVR4
- W2K

Klassisches Unix Scheduling

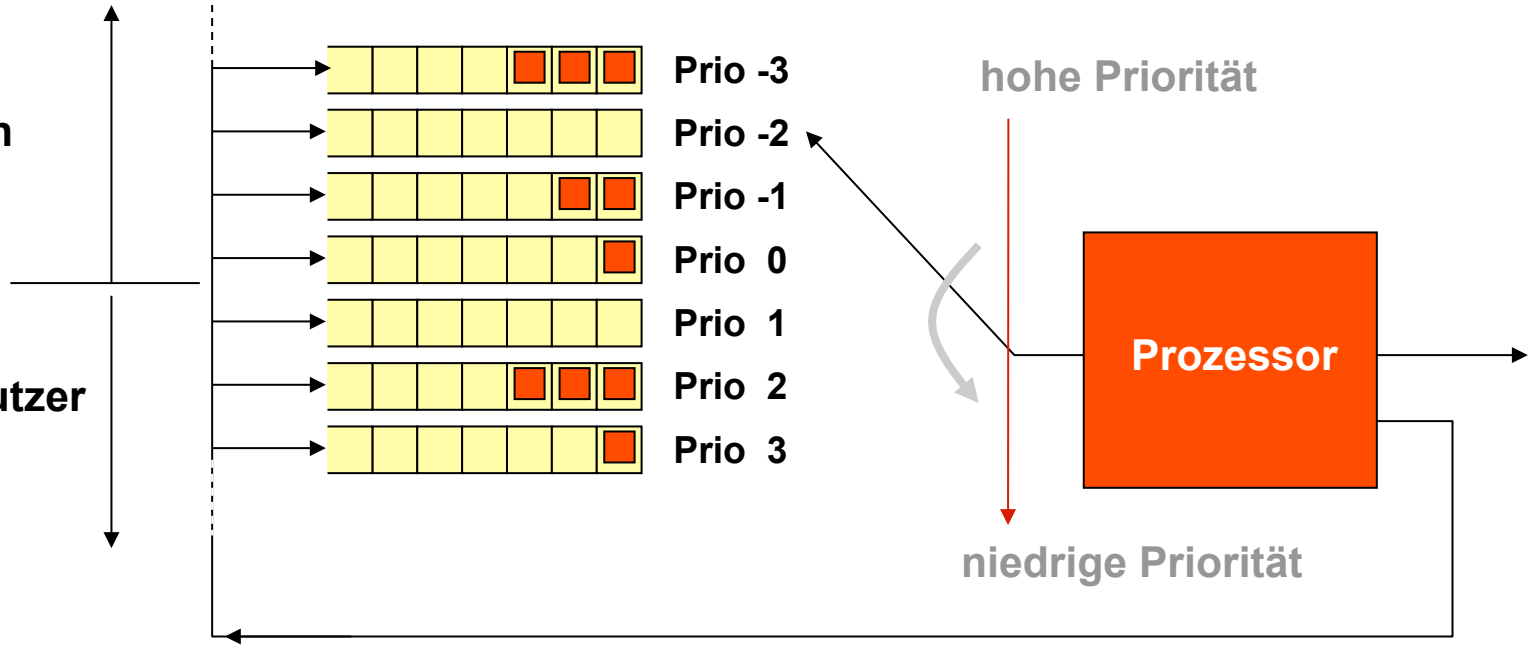
Eingesetzt in SVR3 und 4.3 BSD. Eine modifizierte Version wird in Linux verwendet.

- **Optimiert für interaktive Systeme, d.h. gute Antwortzeiten und faire Zuordnung.**
- **Variante des Fair-Share Schedulings, d.h. 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.

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

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:

1. **Swapper**
2. **E/A Kontrolle (blockorientierte Geräte)**
3. **Dateiänderungen**
4. **E/A Kontrolle (zeichenorientierte Geräte)**
5. **Benutzerprozesse**

Priorität= Basis + CPUj (i-1) /4 + nice

Neue ”Auslastung"= alte Auslastung/2

Auslastung steigt bei einem Prozess im aktiven Intervall an!

W. Stallings, Moderne BS, Kap. 9

Zeit	Prozess A		Prozess B		Prozess C	
	Priorität	CPU-Zähler	Priorität	CPU-Zähler	Priorität	CPU-Zähler
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

Graues Feld = aktiver Prozess

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"**

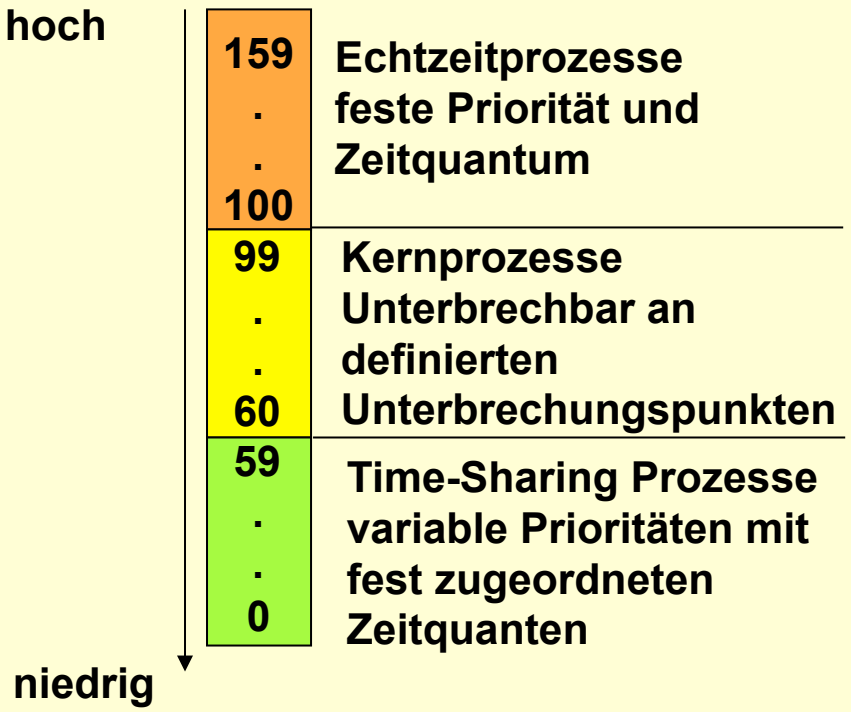
Echtzeit bedeutet hier: FIFO-Threads haben Prioritäten und können nur unterbrochen werden wenn:

- **ein anderer FIFO-Echtzeit-Thread mit höherere Priorität bereit wird,**
- **der aktuelle FIFO-Echtzeit-Thread blockiert,**
- **der aktive FIFO-Echtzeit-Thread explizit den Prozessor freigibt (*sched_yield*)**

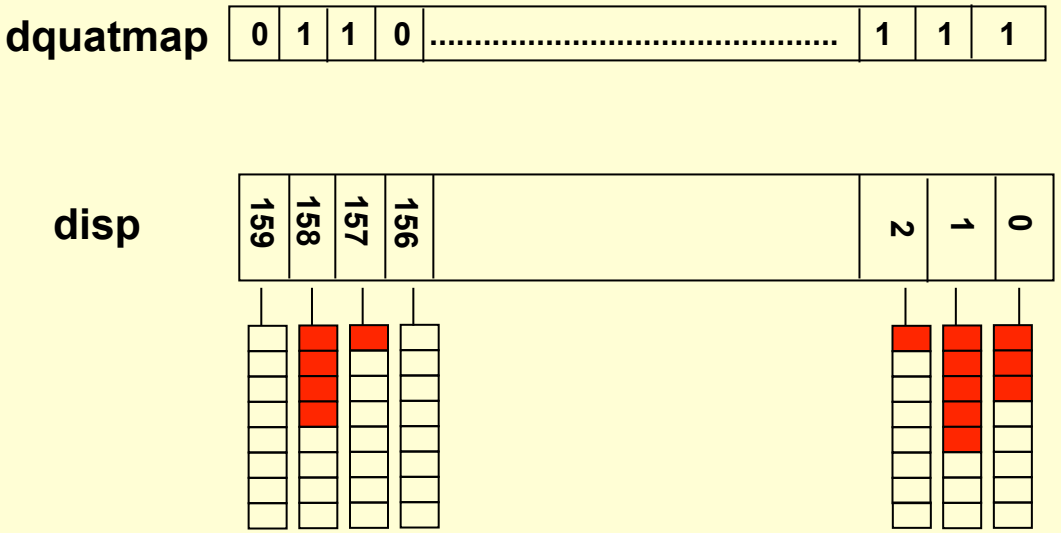
- **für RR-Echtzeit-Threads gilt zusätzlich, dass sie bei abgelaufenem Zeitquantum unterbrochen werden, wenn es einen anderen bereiten Prozess derselben Prioritätsstufe gibt.**

Scheduling in Unix SVR4

Prioritätsklassen und Prioritätsstufen



Organisation der Warteschlange



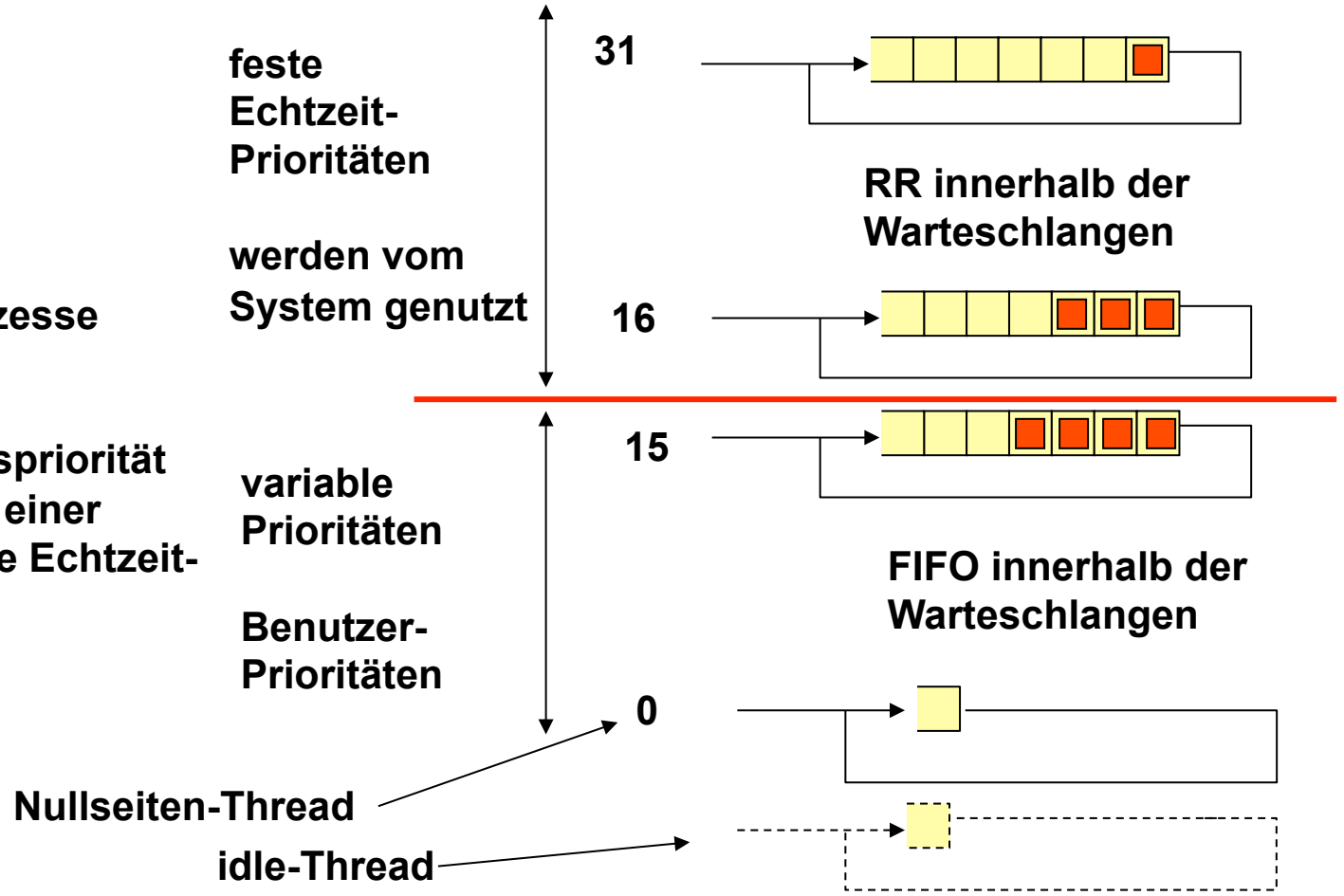
Innerhalb der Time-Sharing Klasse variable Prioritäten und Zeitquanten. Das Zeitquantum hängt von der Prioritätsstufe ab.

Scheduling in W2K

Einheiten des Schedulings: Threads

Priorität =
Basispriorität des Prozesse
+ Priorität des Threads

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



Scheduling in W2K

Festlegung der Basis-Prioritätsklassen

SetPriorityClass

Prozessklassen- (Basis-) Prioritäten

höchste (32) ← niedrigste (0)

Thread-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

SetThreadPriority

Priorität= Basis + CPUj (i-1) /4 + nice

Neue ”Auslastung"= alte Auslastung/2

Auslastung steigt bei einem Prozess im aktiven Intervall an!

W. Stallings, Moderne BS, Kap. 9

Zeit	Prozess A		Prozess B		Prozess C	
	Priorität	CPU-Zähler	Priorität	CPU-Zähler	Priorität	CPU-Zähler
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

Graues Feld = aktiver Prozess

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 höherere 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.**

Ein kurzer Ausflug in das Scheduling unter Echtzeitbedingungen

Bisherige Ziele und Annahmen:

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

Eigenschaften von Echtzeitprozessen:

- Dringlichkeit,
- Zeitliche Vorhersagbarkeit,
- Kenntnis der Bereitzeiten, Ausführungszeiten und Fristen.

General Purpose Scheduling und Echtzeitscheduling besitzen gegensätzliche Ziele!

Beispiele populärer Kostenfunktionen

- **Mittlere Antwortzeit:** $t_r = 1/n \sum_{(i=1, ..., n)} (c_i - r_i)$
  (avarage response time)

- **Zeit bis zum vollständigen Abschluß:** $t_c = \max_i (c_i) - \min_i (r_i)$
  (total completion time)

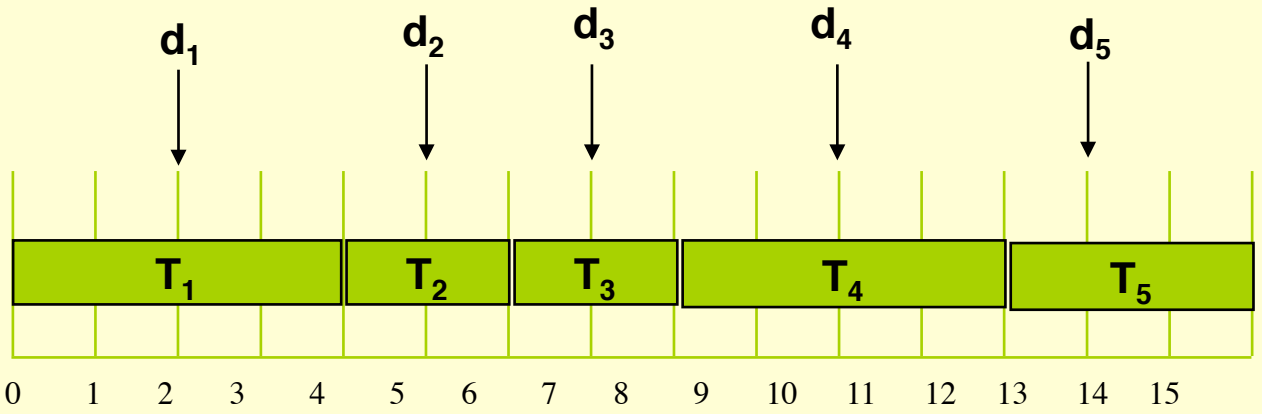
- **Maximale Verspätung:** $L_{max} = \max_i (c_i - d_i)$
  (maximum lateness)

- **Maximale Zahl verspäteter Tasks:** $N_{late} = \sum_{(i=1, ..., n)} miss(c_i)$

  mit: $$miss(c_i) = \begin{cases} 0 & \text{if } c_i \le d_i \\ 1 & \text{sonst} \end{cases}$$

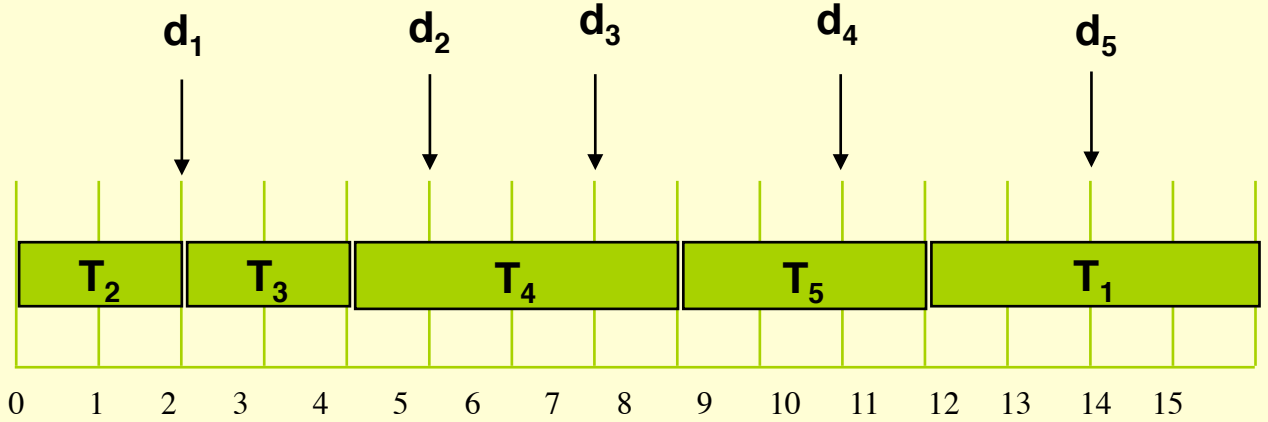
Beispiel für Kostenfkt.

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:** (maximum lateness) $L_{max} = \max_{i} (c_i - d_i)$

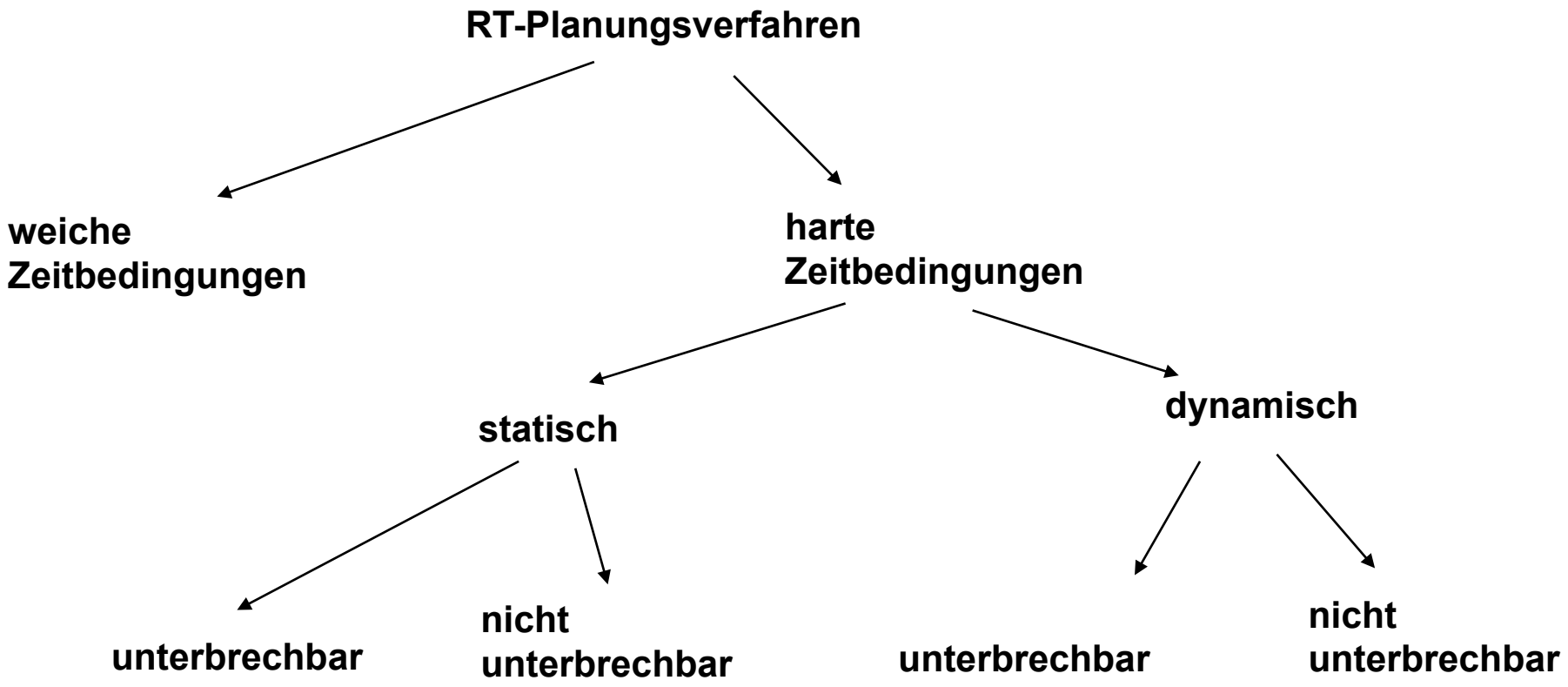
Für Prozesse mit harten Zeitbedingungen gilt dabei:

$$L_{max} \leq 0$$

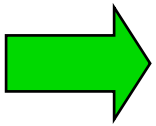
Die vorgegebene Deadline muss immer eingehalten werden.

Taxonomie der RT-Planungsverfahren

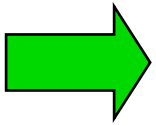
(Cheng, Stankovic, Ramamritham)



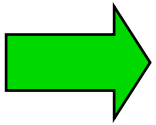
Klassifizierung der Schedulingalgorithmen



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



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



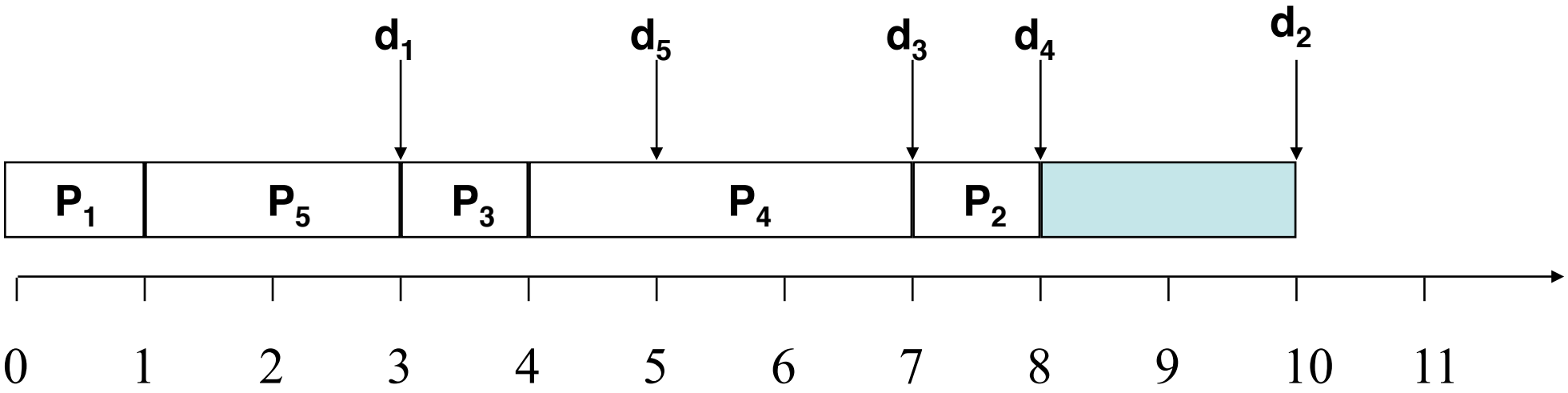
Planungs(Optimalitäts)kriterium bezogen auf eine Kostenfunktion

EDF : Earliest Deadline First

- Die Ausführungszeiten und (absoluten) 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
Δe_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 eines allwissenden Schedulers)

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

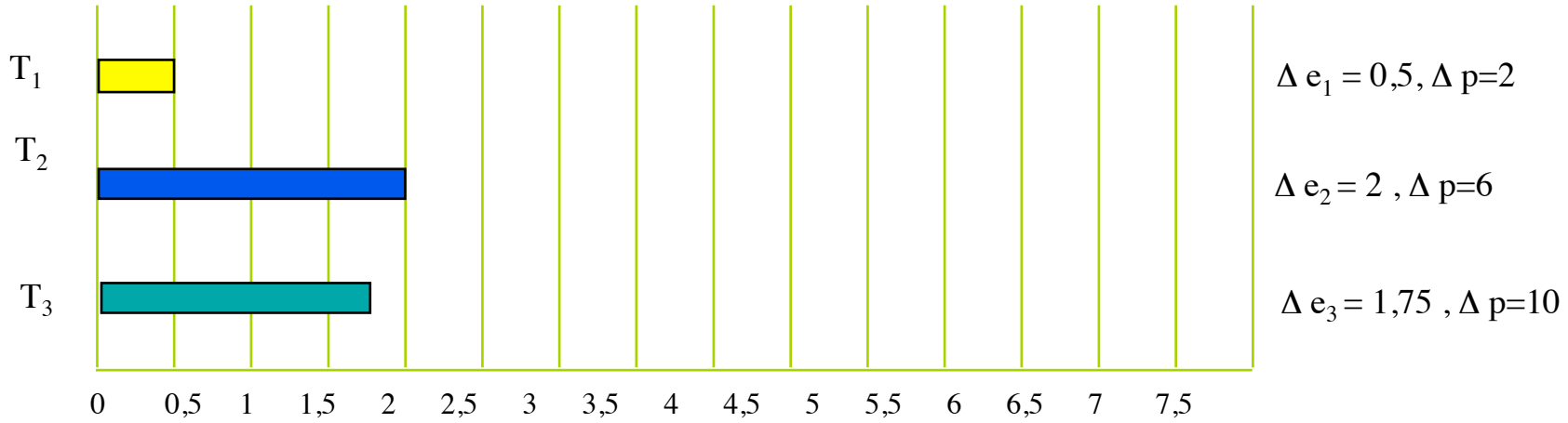
RMS: Echtzeitschedulingverfahren für periodische Prozesse

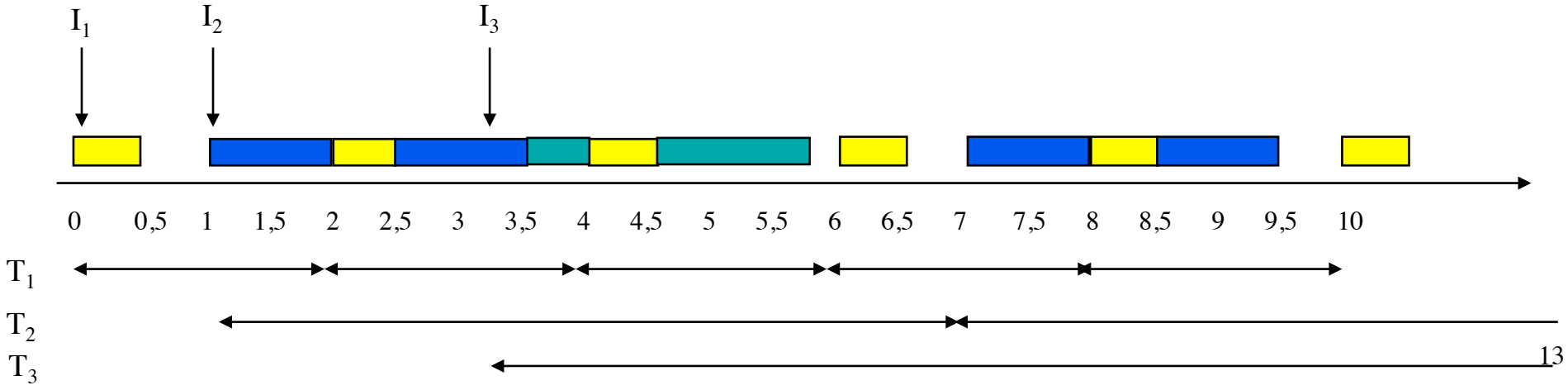
Rate Monotonic Scheduling (RMS): **Für einen periodischen Prozess wird die Priorität als invers zur Länge seiner Periode definiert, d.h. je kürzer die Periode, desto höher die Priorität.**

- **Statisches Planungsverfahren, d.h. die Priorität eines Prozesses ist fest.**
- **Optimales statisches Planungsverfahren, d.h. wenn RMS keinen Plan findet, wird auch kein anderes statischen 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: $U_{lub} = n \left(\sqrt[n]{2} - 1 \right)$.

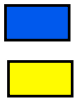
Für n = 1 : U_{lub} = 1
Für n = 2 : U_{lub} = 0,828
Für n → ∞ : $\lim U_{lub}(n) = \ln(2)$ = 0,693

Eigenschaften von RMS :

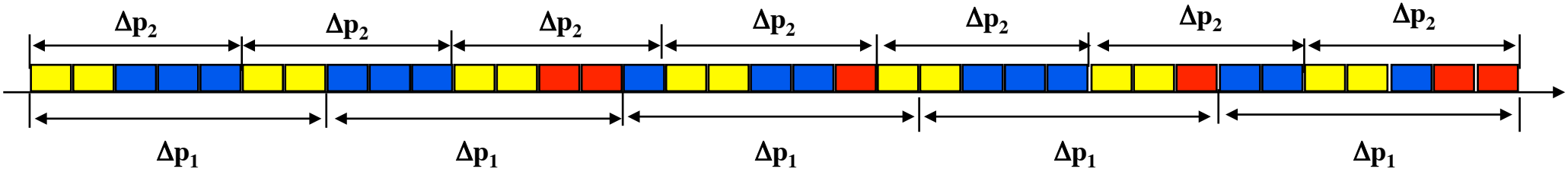
Beispiel für Scheduling nach RMS, in dem die Auslastung über der RMS-Grenze liegt, d.h. RMS keinen Plan findet, obwohl einer existiert.

1. Ausgangssituation: für diese Werte findet RMS einen Plan. Die Auslastung liegt mit 0,828 an der theoretischen Grenze

T_1 : $\Delta e_1 = 3$, $\Delta p_1 = 7$
T_2 : $\Delta e_2 = 2$, $\Delta p_2 = 5$



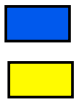
kgV = 35
T_1 : 15 Zeiteinheiten
T_2 : 14 Zeiteinheiten
$29/35 = 0{,}828 \approx n\,(\sqrt[n]{2} - 1)$



Eigenschaften von RMS :

Situation 2: Erhöhung des Rechenbedarfs von T_1 um 1 Einheit ($\Delta e_1 = 4$). RMS kann nicht mehr angewandt werden. Die theoretische Grenze der Auslastung wurde überschritten.

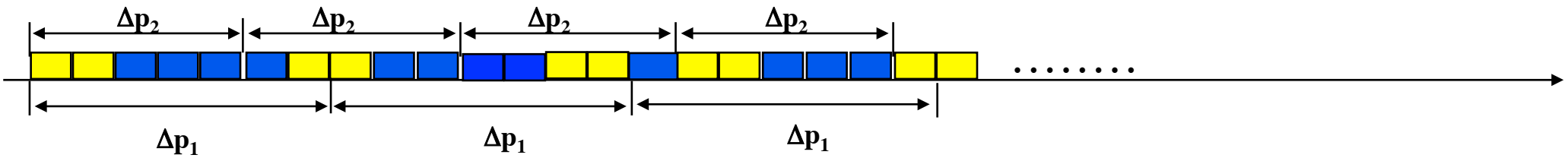
T_1 : $\Delta e_1 = 4$, $\Delta p_1 = 7$
T_2 : $\Delta e_2 = 2$, $\Delta p_2 = 5$



kgV = 35
T_1 : 20 Zeiteinheiten
T_2 : 14 Zeiteinheiten
$34/35 = 0{,}971 > n (\sqrt[n]{2} - 1) = 0{,}828$

T_1 **hat seine Deadline verletzt, da es bis zum Ende seine Intervalls noch keine 4 Zeiteinheiten zur Verfügung hatte.**

Für EDF ist das kein Problem, da das Verfahren den Prozessor bis 1 auslasten kann.



Optimalität von RMS

Satz:

Sei T eine Taskmenge, für die eine gefundene (statische) Prioritätszuordnung bereits einen brauchbaren Plan liefert. Dann wird auch die Prioritätszuordnung nach monotonen Raten einen brauchbaren Plan liefern.

C.L. Liu, J.W. Layland
Scheduling algorithms for multiprogramming in a hard- real-time environment.
Journal of the ACM 20(1), January 73, pp.46-61

Zusammenfassung Scheduling:

Durch Scheduling wird die Ressource "Prozessorzeit" zwischen Prozessen aufgeteilt.

Unterteilung in langfristiges, mittelfristiges und kurzfristiges Scheduling.

Ziele des Schedulings sind an der Anwendung orientiert z.B.

- Auslastung
- Fairness
- Bevorzugung kurzer oder langer Prozesse
- Kein Verhungern
- Abschluss vor einer Deadline

Keine "one fits all" Strategie!

Reale Scheduler sind meist Kombinationen verschiedener Grundstrategien.