Systemprogrammierung

Grundlagen von Betriebssystemen

Teil C – IX.2 Prozessverwaltung: Einplanungsverfahren

30. Oktober 2025

Rüdiger Kapitza

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Agenda

Gliederung

Lehrstoff

- gängige Klassen der **Ein-/Umplanung** von Prozessen kennenlernen und in ihrer Bedeutung einschätzen können
  - jedes Verfahren einer Klasse hat bestimmte **Gütemerkmale** im Fokus
  - bei mehreren Merkmalen müsste ein **Kompromiss** gefunden werden
  - scheidet Konfliktlösung aus, ist eine geeignete **Priorisierung** vorzunehmen

Lehrstoff

- gängige Klassen der **Ein-/Umplanung** von Prozessen kennenlernen und in ihrer Bedeutung einschätzen können
  - jedes Verfahren einer Klasse hat bestimmte **Gütemerkmale** im Fokus
  - bei mehreren Merkmalen müsste ein **Kompromiss** gefunden werden
  - scheidet Konfliktlösung aus, ist eine geeignete **Priorisierung** vorzunehmen
- die Verfahren auf **nicht-funktionale Eigenschaften** untersuchen, so Gemeinsamkeiten und Unterschiede erfassen
  - Gerechtigkeit, minimale Antwort- oder Durchlaufzeit
  - maximaler Durchsatz, maximale Auslastung
  - Termineinhaltung, Dringlichkeiten genügend, Vorhersagbarkeit

Lehrstoff

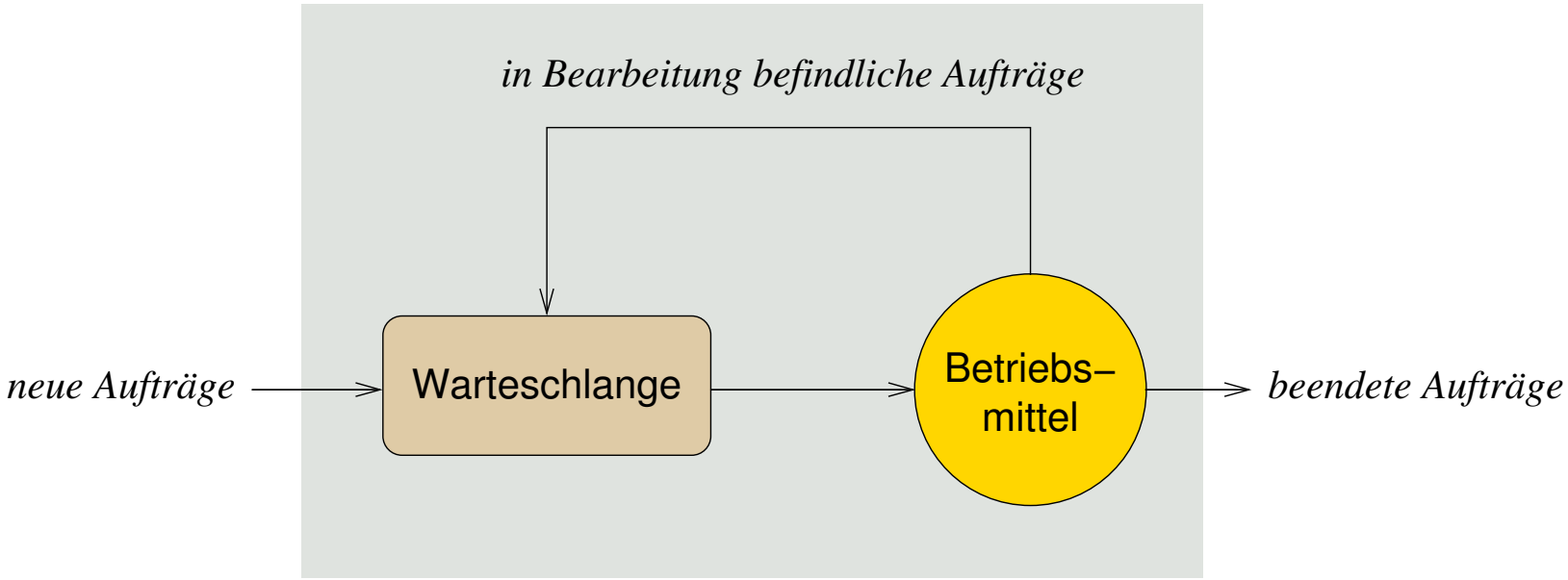
- gängige Klassen der **Ein-/Umplanung** von Prozessen kennenlernen und in ihrer Bedeutung einschätzen können
  - jedes Verfahren einer Klasse hat bestimmte **Gütemerkmale** im Fokus
  - bei mehreren Merkmalen müsste ein **Kompromiss** gefunden werden
  - scheidet Konfliktlösung aus, ist eine geeignete **Priorisierung** vorzunehmen
- die Verfahren auf **nicht-funktionale Eigenschaften** untersuchen, so Gemeinsamkeiten und Unterschiede erfassen
  - Gerechtigkeit, minimale Antwort- oder Durchlaufzeit
  - maximaler Durchsatz, maximale Auslastung
  - Termineinhaltung, Dringlichkeiten genügend, Vorhersagbarkeit
- erkennen, dass es die „eierlegende Wollmilchsau“ auch in einer virtuellen Welt nicht geben kann…

Kein einziges Verfahren zur Ein-/Umplanung von Prozessen hat nur Vorteile, befriedigt alle Bedürfnisse, genügt allen Ansprüchen.

Einplanungsalgorithmen

vgl. [8, S. 40]

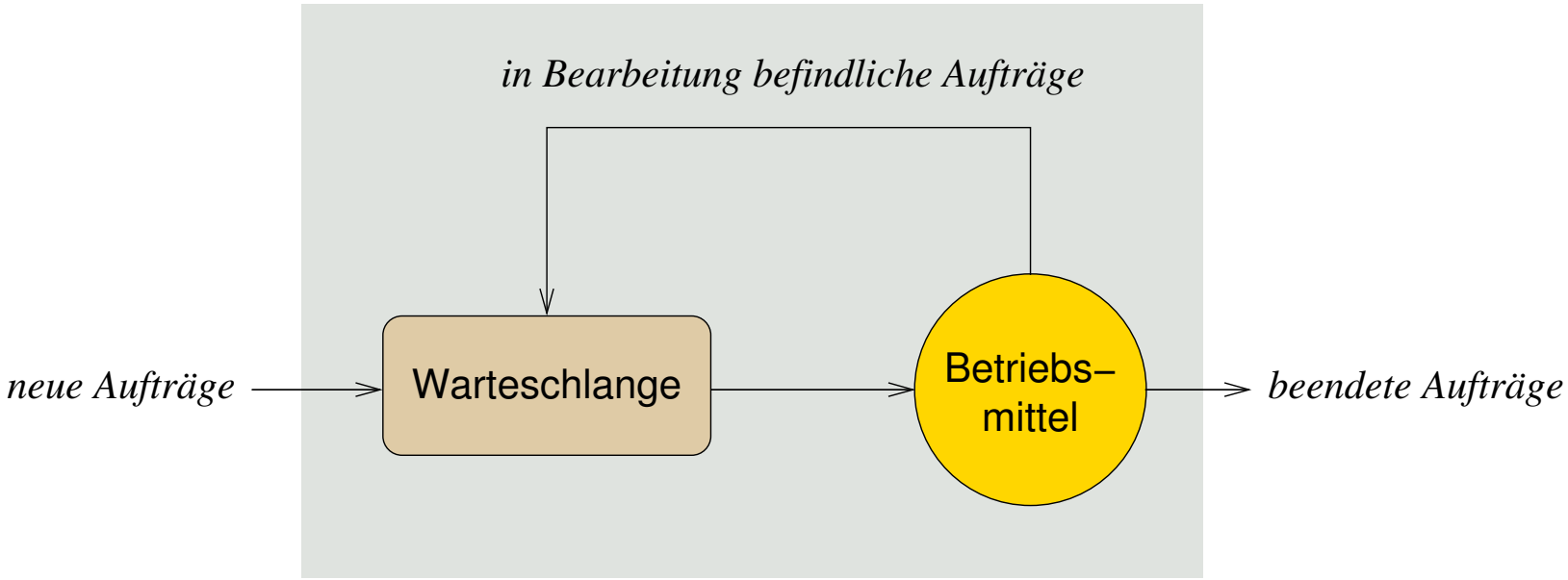
- Verwaltung von (betriebsmittelgebundenen) **Warteschlangen**



Einplanungsalgorithmen

vgl. [8, S. 40]

- Verwaltung von (betriebsmittelgebundenen) **Warteschlangen**



Ein einzelner Einplanungsalgorithmus ist charakterisiert durch die Reihenfolge von Prozessen in der Warteschlange und die Bedingungen, unter denen die Prozesse in die Warteschlange eingereiht werden. [12]

Warteschlangentheorie

vgl. [8, S. 41]

- die Charakterisierung von **Einplanungsalgorithmen** macht glauben, Betriebssysteme fokussiert „mathematisch“ studieren zu müssen
  - R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
  - E. G. Coffman, P. J. Denning. *Operating System Theory*.
  - L. Kleinrock. *Queuing Systems*, *Volume I: Theory*.

Warteschlangentheorie

vgl. [8, S. 41]

- die Charakterisierung von **Einplanungsalgorithmen** macht glauben, Betriebssysteme fokussiert „mathematisch“ studieren zu müssen
  - R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
  - E. G. Coffman, P. J. Denning. *Operating System Theory*.
  - L. Kleinrock. *Queuing Systems*, *Volume I: Theory*.
- praktische Umsetzung offenbart jedoch einen **Querschnittsbelang** (*cross-cutting concern*), der sich kaum modularisieren lässt
  - spez. Betriebsmittelmerkmale stehen ggf. Bedürfnissen der Prozesse, die Aufträge zur Betriebsmittelnutzung abgesetzt haben, gegenüber
  - dabei ist die Prozessreihenfolge in Warteschlangen (bereit, blockiert) ein Aspekt, die Auftragsreihenfolge dagegen ein anderer Aspekt
  - **Interferenz**[1] bei der Durchsetzung der Strategien kann die Folge sein

[1] lat. *inter* zwischen und *ferire* von altfrz. *s’entreferir* sich gegenseitig schlagen.

Warteschlangentheorie

vgl. [8, S. 41]

- die Charakterisierung von **Einplanungsalgorithmen** macht glauben, Betriebssysteme fokussiert „mathematisch“ studieren zu müssen
  - R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
  - E. G. Coffman, P. J. Denning. *Operating System Theory*.
  - L. Kleinrock. *Queuing Systems*, *Volume I: Theory*.
- praktische Umsetzung offenbart jedoch einen **Querschnittsbelang** (*cross-cutting concern*), der sich kaum modularisieren lässt
  - spez. Betriebsmittelmerkmale stehen ggf. Bedürfnissen der Prozesse, die Aufträge zur Betriebsmittelnutzung abgesetzt haben, gegenüber
  - dabei ist die Prozessreihenfolge in Warteschlangen (bereit, blockiert) ein Aspekt, die Auftragsreihenfolge dagegen ein anderer Aspekt
  - **Interferenz**[1] bei der Durchsetzung der Strategien kann die Folge sein
- Einplanungsverfahren stehen und fallen mit den Vorgaben, die für die jeweilige **Zieldomäne** zu treffen sind
  - die „Eier-legende Wollmilchsau“ kann es nicht geben
  - Kompromisslösungen sind geläufig — aber nicht in allen Fällen tragfähig

[1] lat. *inter* zwischen und *ferire* von altfrz. *s’entreferir* sich gegenseitig schlagen.

Gliederung

Einordnung

Klassifikation

Kooperativ vs. Präemptiv

- **Souverän** ist die Anwendung oder das Betriebssystem verhält sich Entwicklungen gegenüber zuvorkommend, vorsorglich, vorbeugend

Kooperativ vs. Präemptiv

- **Souverän** ist die Anwendung oder das Betriebssystem verhält sich Entwicklungen gegenüber zuvorkommend, vorsorglich, vorbeugend
  kooperative Planung (*cooperative scheduling*)
  - Ein-/Umplanung voneinander abhängiger Prozesse
  - Prozessen wird die CPU nicht zugunsten anderer Prozesse entzogen
  - der laufende Prozess gibt die CPU nur mittels Systemaufruf ab
    - die Systemaufrufbehandlung aktiviert (direkt/indirekt) den Scheduler
    - systemaufruffreie Endlosschleifen beeinträchtigen andere Prozesse
  - **CPU-Monopolisierung** ist möglich: *run to completion*

Kooperativ vs. Präemptiv

- **Souverän** ist die Anwendung oder das Betriebssystem verhält sich Entwicklungen gegenüber zuvorkommend, vorsorglich, vorbeugend

  kooperative Planung (*cooperative scheduling*)
  - Ein-/Umplanung voneinander abhängiger Prozesse
  - Prozessen wird die CPU nicht zugunsten anderer Prozesse entzogen
  - der laufende Prozess gibt die CPU nur mittels Systemaufruf ab
    - die Systemaufrufbehandlung aktiviert (direkt/indirekt) den Scheduler
    - systemaufruffreie Endlosschleifen beeinträchtigen andere Prozesse
  - **CPU-Monopolisierung** ist möglich: *run to completion*

  präemptive Planung (*preemptive scheduling*)
  - Ein-/Umplanung voneinander unabhängiger Prozesse
  - Prozessen kann die CPU entzogen werden, zugunsten anderer
  - der laufende Prozess wird ereignisbedingt von der CPU verdrängt
    - die Ereignisbehandlung aktiviert (direkt/indirekt) den Scheduler
    - Endlosschleifen beeinträchtigen andere Prozesse nicht (bzw. kaum)
  - Monopolisierung der CPU ist nicht möglich: **CPU-Schutz**

Kooperativ vs. Präemptiv

- **Souverän** ist die Anwendung oder das Betriebssystem verhält sich Entwicklungen gegenüber zuvorkommend, vorsorglich, vorbeugend

  kooperative Planung (*cooperative scheduling*)
  - Ein-/Umplanung voneinander abhängiger Prozesse
  - Prozessen wird die CPU nicht zugunsten anderer Prozesse entzogen
  - der laufende Prozess gibt die CPU nur mittels Systemaufruf ab
    - die Systemaufrufbehandlung aktiviert (direkt/indirekt) den Scheduler
    - systemaufruffreie Endlosschleifen beeinträchtigen andere Prozesse
  - **CPU-Monopolisierung** ist möglich: *run to completion*

  präemptive Planung (*preemptive scheduling*)
  - Ein-/Umplanung voneinander unabhängiger Prozesse
  - Prozessen kann die CPU entzogen werden, zugunsten anderer
  - der laufende Prozess wird ereignisbedingt von der CPU verdrängt
    - die Ereignisbehandlung aktiviert (direkt/indirekt) den Scheduler
    - Endlosschleifen beeinträchtigen andere Prozesse nicht (bzw. kaum)
  - Monopolisierung der CPU ist nicht möglich: **CPU-Schutz**
- **Synergie**: auf Maschinenprogrammebene kooperative *user threads*, auf Betriebssystemebene präemptive *kernel threads* [8, vgl. S. 27]...

Deterministisch vs. Probabilistisch

- alle Abläufe durch ***à priori* Wissen** eindeutig festlegen können oder die **Wahrscheinlichkeit** berücksichtigend

Deterministisch vs. Probabilistisch

- alle Abläufe durch ***à priori* Wissen** eindeutig festlegen können oder die **Wahrscheinlichkeit** berücksichtigend

  deterministische Planung (*deterministic scheduling*)
  - alle Prozesse (Rechenstoßlängen)[2] & ggf. **Termine** sind bekannt
  - die genaue Vorhersage der CPU-Auslastung ist möglich
  - das System stellt die Einhaltung von **Zeitgarantien** sicher
  - die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast

[2] Bei (strikten) Echtzeitsystemen mindestens die Stoßlänge des „schlimmsten Falls“ (*worst-case execution time*, WCET).

Deterministisch vs. Probabilistisch

- alle Abläufe durch ***à priori* Wissen** eindeutig festlegen können oder die **Wahrscheinlichkeit** berücksichtigend

  deterministische Planung (*deterministic scheduling*)
  - alle Prozesse (Rechenstoßlängen)[2] & ggf. **Termine** sind bekannt
  - die genaue Vorhersage der CPU-Auslastung ist möglich
  - das System stellt die Einhaltung von **Zeitgarantien** sicher
  - die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast

  probabilistische Planung (*probabilistic scheduling*)
  - Prozesse (exakte Rechenstoßl.) sind unbekannt, ggf. auch Termine
  - die CPU-Auslastung kann lediglich abgeschätzt werden
  - das System kann Zeitgarantien weder geben noch einhalten
  - Zeitgarantien sind durch die Anwendung sicherzustellen

[2] Bei (strikten) Echtzeitsystemen mindestens die Stoßlänge des „schlimmsten Falls“ (*worst-case execution time*, WCET).

Deterministisch vs. Probabilistisch

- alle Abläufe durch ***à priori* Wissen** eindeutig festlegen können oder die **Wahrscheinlichkeit** berücksichtigend

  deterministische Planung (*deterministic scheduling*)
  - alle Prozesse (Rechenstoßlängen)[2] & ggf. **Termine** sind bekannt
  - die genaue Vorhersage der CPU-Auslastung ist möglich
  - das System stellt die Einhaltung von **Zeitgarantien** sicher
  - die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast

  probabilistische Planung (*probabilistic scheduling*)
  - Prozesse (exakte Rechenstoßl.) sind unbekannt, ggf. auch Termine
  - die CPU-Auslastung kann lediglich abgeschätzt werden
  - das System kann Zeitgarantien weder geben noch einhalten
  - Zeitgarantien sind durch die Anwendung sicherzustellen
- dabei fällt die Abgrenzung nicht immer so scharf aus: wahrscheinliche Abläufe vorherzusagen, ist sehr nützlich…

[2] Bei (strikten) Echtzeitsystemen mindestens die Stoßlänge des „schlimmsten Falls“ (*worst-case execution time*, WCET).

Statisch vs. Dynamisch

- Abläufe entkoppelt von oder gekoppelt mit der Programmausführung bestimmen und entsprechend entwickeln

Statisch vs. Dynamisch

- Abläufe entkoppelt von oder gekoppelt mit der Programmausführung bestimmen und entsprechend entwickeln
  statische Planung (*off-line scheduling*), vorlaufend
  - vor Betrieb des Prozess- oder Rechensystems
  - die Berechnungskomplexität verbietet Planung im Betrieb
    - z.B. die Berechnung, dass alle Zeitvorgaben garantiert eingehalten werden
    - unter Berücksichtigung jeder abfangbaren katastrophalen Situation
  - Ergebnis der Vorberechnung ist ein **vollständiger Ablaufplan**
    - u.a. erstellt per Quelltextanalyse spezieller „Übersetzer“
    - oft zeitgesteuert abgearbeitet als Teil der Prozesseinlastung
  - die Verfahren sind zumeist beschränkt auf **strikte Echtzeitsysteme**

Statisch vs. Dynamisch

- Abläufe entkoppelt von oder gekoppelt mit der Programmausführung bestimmen und entsprechend entwickeln

  statische Planung (*off-line scheduling*), vorlaufend
  - vor Betrieb des Prozess- oder Rechensystems
  - die Berechnungskomplexität verbietet Planung im Betrieb
    - z.B. die Berechnung, dass alle Zeitvorgaben garantiert eingehalten werden
    - unter Berücksichtigung jeder abfangbaren katastrophalen Situation
  - Ergebnis der Vorberechnung ist ein **vollständiger Ablaufplan**
    - u.a. erstellt per Quelltextanalyse spezieller „Übersetzer“
    - oft zeitgesteuert abgearbeitet als Teil der Prozesseinlastung
  - die Verfahren sind zumeist beschränkt auf **strikte Echtzeitsysteme**

  dynamische Planung (*on-line scheduling*), mitlaufend
  - während Betrieb des Prozess- oder Rechensystems
  - Stapelsysteme, interaktive Systeme, verteilte Systeme
  - schwache und feste Echtzeitsysteme

Statisch vs. Dynamisch

- Abläufe entkoppelt von oder gekoppelt mit der Programmausführung bestimmen und entsprechend entwickeln

  statische Planung (*off-line scheduling*), vorlaufend
  - vor Betrieb des Prozess- oder Rechensystems
  - die Berechnungskomplexität verbietet Planung im Betrieb
    - z.B. die Berechnung, dass alle Zeitvorgaben garantiert eingehalten werden
    - unter Berücksichtigung jeder abfangbaren katastrophalen Situation
  - Ergebnis der Vorberechnung ist ein **vollständiger Ablaufplan**
    - u.a. erstellt per Quelltextanalyse spezieller „Übersetzer“
    - oft zeitgesteuert abgearbeitet als Teil der Prozesseinlastung
  - die Verfahren sind zumeist beschränkt auf **strikte Echtzeitsysteme**

  dynamische Planung (*on-line scheduling*), mitlaufend
  - während Betrieb des Prozess- oder Rechensystems
  - Stapelsysteme, interaktive Systeme, verteilte Systeme
  - schwache und feste Echtzeitsysteme
- auch hier ist die Abgrenzung nicht immer so scharf: von vorläufigen Ablaufplänen ausgehen zu können, ist sehr hilfreich…

Asymmetrisch vs. Symmetrisch (1)

- für **mehrere Prozessoren** Abläufe nach verschiedenen Kriterien oder ihren wechselseitigen Entsprechungen festlegen

Asymmetrisch vs. Symmetrisch (1)

- für **mehrere Prozessoren** Abläufe nach verschiedenen Kriterien oder ihren wechselseitigen Entsprechungen festlegen
  asymmetrische Planung (*asymmetric scheduling*)
  - je nach den Prozessoreigenschaften der **Maschinenprogrammebene**
  - obligatorisch in einem asymmetrischen Multiprozessorsystem
    - Rechnerarchitektur mit programmierbare Spezialprozessoren
    - z.B. Grafik- und/oder Kommunikationsprozessoren einerseits
    - ein Feld konventioneller (gleichartiger) Prozessoren andererseits
  - optional in einem symmetrischen Multiprozessorsystem (s.u.)
    - das Betriebssystem hat freie Hand über die Prozessorvergabe
  - Prozesse in funktionaler Hinsicht ungleich verteilen (müssen)

Asymmetrisch vs. Symmetrisch (1)

- für **mehrere Prozessoren** Abläufe nach verschiedenen Kriterien oder ihren wechselseitigen Entsprechungen festlegen

  asymmetrische Planung (*asymmetric scheduling*)
  - je nach den Prozessoreigenschaften der **Maschinenprogrammebene**
  - obligatorisch in einem asymmetrischen Multiprozessorsystem
    - Rechnerarchitektur mit programmierbare Spezialprozessoren
    - z.B. Grafik- und/oder Kommunikationsprozessoren einerseits
    - ein Feld konventioneller (gleichartiger) Prozessoren andererseits
  - optional in einem symmetrischen Multiprozessorsystem (s.u.)
    - das Betriebssystem hat freie Hand über die Prozessorvergabe
  - Prozesse in funktionaler Hinsicht ungleich verteilen (müssen)

  symmetrische Planung (*symmetric scheduling*)
  - je nach den Prozessoreigenschaften der **Befehlssatzebene**
  - identische Prozessoren, alle geeignet zur Programmausführung
  - Prozesse gleichmäßig auf die Prozessoren verteilen: Lastausgleich

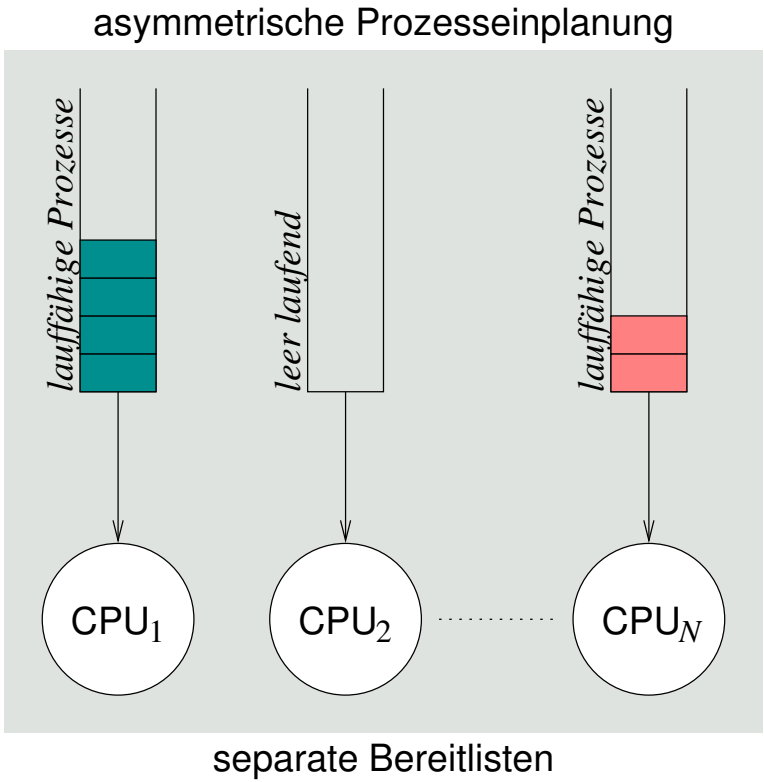
Asymmetrisch vs. Symmetrisch (1)

- für **mehrere Prozessoren** Abläufe nach verschiedenen Kriterien oder ihren wechselseitigen Entsprechungen festlegen

  asymmetrische Planung (*asymmetric scheduling*)
  - je nach den Prozessoreigenschaften der **Maschinenprogrammebene**
  - obligatorisch in einem asymmetrischen Multiprozessorsystem
    - Rechnerarchitektur mit programmierbare Spezialprozessoren
    - z.B. Grafik- und/oder Kommunikationsprozessoren einerseits
    - ein Feld konventioneller (gleichartiger) Prozessoren andererseits
  - optional in einem symmetrischen Multiprozessorsystem (s.u.)
    - das Betriebssystem hat freie Hand über die Prozessorvergabe
  - Prozesse in funktionaler Hinsicht ungleich verteilen (müssen)

  symmetrische Planung (*symmetric scheduling*)
  - je nach den Prozessoreigenschaften der **Befehlssatzebene**
  - identische Prozessoren, alle geeignet zur Programmausführung
  - Prozesse gleichmäßig auf die Prozessoren verteilen: Lastausgleich
- dabei kann jedem Prozessor eine eigene Bereitliste zugeordnet sein oder (Gruppen von) Prozessoren teilen sich eine Bereitliste

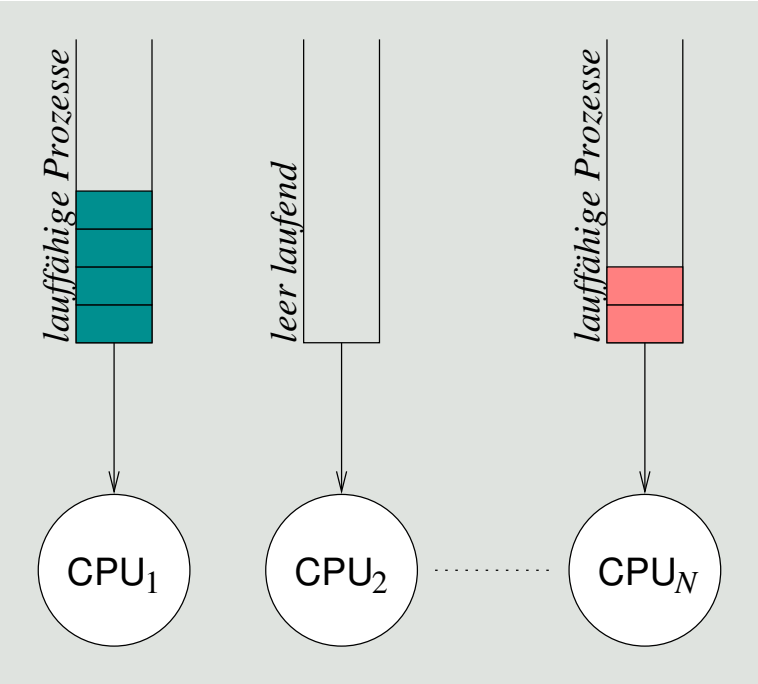
Asymmetrisch vs. Symmetrisch (2)



- lokale Bereitliste
- ggf. ungleichmäßige Auslastung

Asymmetrisch vs. Symmetrisch (2)

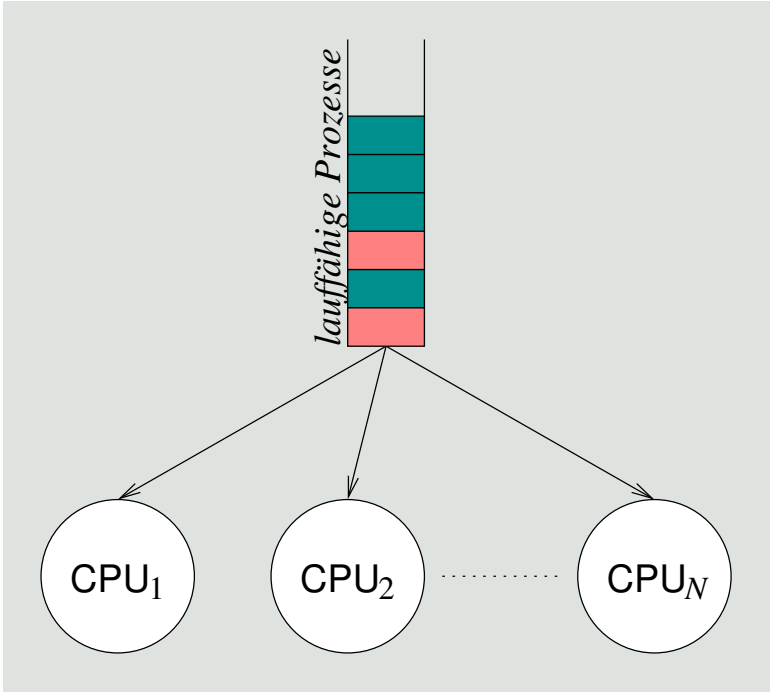
asymmetrische Prozesseinplanung



separate Bereitlisten

- lokale Bereitliste
- ggf. ungleichmäßige Auslastung

symmetrische Prozesseinplanung

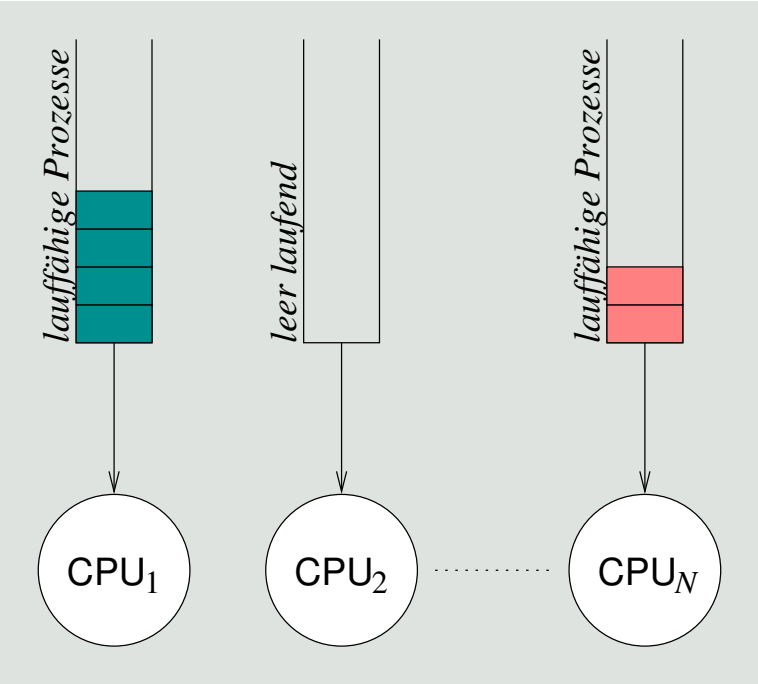


gemeinsame Bereitliste

- globale Bereitliste
- ggf. gleichmäßige Auslastung

Asymmetrisch vs. Symmetrisch (2)

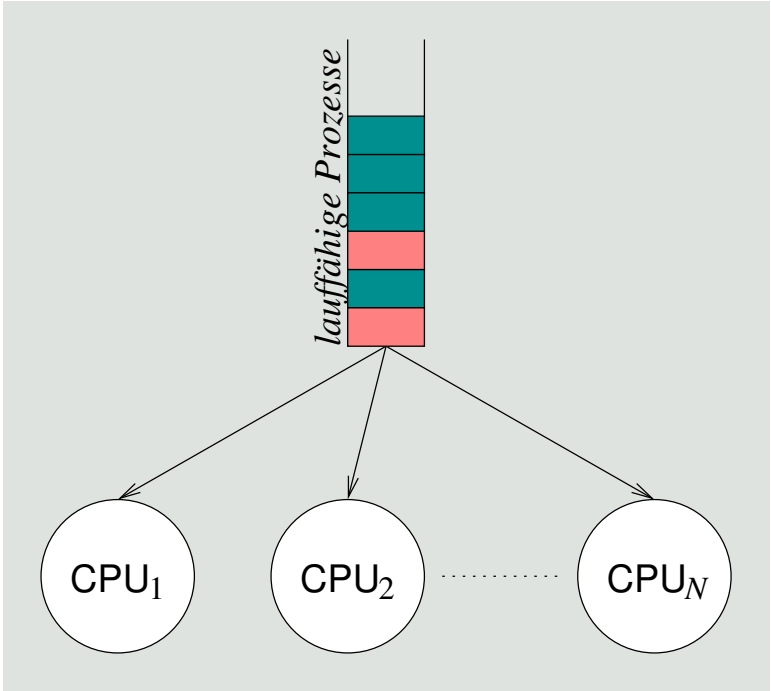
asymmetrische Prozesseinplanung



separate Bereitlisten

- lokale Bereitliste
- ggf. ungleichmäßige Auslastung
- ohne gegenseitige Beeinflussung
  - keine Multiprozessorsynchronisation

symmetrische Prozesseinplanung



gemeinsame Bereitliste

- globale Bereitliste
- ggf. gleichmäßige Auslastung
- gegenseitige Beeinflussung
  - Multiprozessorsynchronisation

Gliederung

Verfahrensweisen

Überblick

Klassische Planungs- bzw. Auswahlverfahren

- betrachtet werden grundlegende Ansätze für **Uniprozessorsysteme**, je nach Klassifikationsmerkmal bzw. nichtfunktionaler Eigenschaft:

kooperativ	FCFS	gerecht
	▪ wer zuerst kommt, mahlt zuerst…	
verdrängend	RR, VRR	reihum
	▪ jeder gegen jeden…	
probabilistisch	SPN (SJF), SRTF, HRRN	priorisierend
	▪ die Kleinen nach vorne…	
mehrstufig	MLQ, MLFQ (FB)	
	▪ Multikulti…	

Klassische Planungs- bzw. Auswahlverfahren

- betrachtet werden grundlegende Ansätze für **Uniprozessorsysteme**, je nach Klassifikationsmerkmal bzw. nichtfunktionaler Eigenschaft:

kooperativ	FCFS	gerecht
	▪ wer zuerst kommt, mahlt zuerst…	
verdrängend	RR, VRR	reihum
	▪ jeder gegen jeden…	
probabilistisch	SPN (SJF), SRTF, HRRN	priorisierend
	▪ die Kleinen nach vorne…	
mehrstufig	MLQ, MLFQ (FB)	
	▪ Multikulti…	

- dabei steht die Fähigkeit zur **Interaktion** mit „externen Prozessen” (insb. dem Menschen) als Gütemerkmal im Vordergrund
  - d.h., Auswirkungen auf die **Antwortzeit** von Prozessen

Verfahrensweisen

Kooperativ

FCFS

first-come, first-served

Fair, einfach zu implementieren (FIFO), …, dennoch problematisch.

FCFS

first-come, first-served

Fair, einfach zu implementieren (FIFO), …, dennoch problematisch.

- Prozesse werden nach ihrer **Ankunftszeit** (*arrival time*) eingeplant und in der sich daraus ergebenden Reihenfolge auch verarbeitet
  - nicht-verdrängendes Verfahren, setzt kooperative Prozesse voraus

FCFS

first-come, first-served

Fair, einfach zu implementieren (FIFO), …, dennoch problematisch.

- Prozesse werden nach ihrer **Ankunftszeit** (*arrival time*) eingeplant und in der sich daraus ergebenden Reihenfolge auch verarbeitet
  - nicht-verdrängendes Verfahren, setzt kooperative Prozesse voraus
- gerechtes Verfahren auf Kosten einer im Mittel höheren Antwortzeit und niedrigerem E/A-Durchsatz
  - suboptimal bei einem Mix von kurzen und langen Rechenstößen

Prozesse mit { langen / kurzen } Rechenstößen werden { begünstigt / benachteiligt }

Fair, einfach zu implementieren (FIFO), …, dennoch problematisch.

- Prozesse werden nach ihrer **Ankunftszeit** (*arrival time*) eingeplant und in der sich daraus ergebenden Reihenfolge auch verarbeitet
  - nicht-verdrängendes Verfahren, setzt kooperative Prozesse voraus
- gerechtes Verfahren auf Kosten einer im Mittel höheren Antwortzeit und niedrigerem E/A-Durchsatz
  - suboptimal bei einem Mix von kurzen und langen Rechenstößen

Prozesse mit { langen / kurzen } Rechenstößen werden { begünstigt / benachteiligt }

- Problem: **Konvoieffekt**
  - kurze Prozesse bzw. Rechenstöße folgen einem langen…

FCFS: Konvoieffekt

- Durchlaufzeit kurzer Prozesse im Mix mit langen Prozessen:

Prozess	Zeiten					T_q/T_s
	Ankunft	T_s	Start	Ende	T_q	
A	0	1	0	1	1	1.00
B	1	100	1	101	100	1.00
C	2	1	101	102	100	100.00
D	3	100	102	202	199	1.99
$\emptyset$					100	26.00

T_s = Bedienzeit, T_q = Duchlaufzeit

normalisierte Duchlaufzeit (T_q/T_s)

FCFS: Konvoieffekt

- Durchlaufzeit kurzer Prozesse im Mix mit langen Prozessen:

Prozess	Zeiten					T_q/T_s
	Ankunft	T_s	Start	Ende	T_q	
A	0	1	0	1	1	1.00
B	1	100	1	101	100	1.00
C	2	1	101	102	100	100.00
D	3	100	102	202	199	1.99
$\emptyset$					100	26.00

T_s = Bedienzeit, T_q = Duchlaufzeit

normalisierte Duchlaufzeit (T_q/T_s)

- ideal für A und B, unproblematisch für D

FCFS: Konvoieffekt

- Durchlaufzeit kurzer Prozesse im Mix mit langen Prozessen:

Prozess	Zeiten					T_q/T_s
	Ankunft	T_s	Start	Ende	T_q	
A	0	1	0	1	1	1.00
B	1	100	1	101	100	1.00
C	2	1	101	102	100	100.00
D	3	100	102	202	199	1.99
$\emptyset$					100	26.00

T_s = Bedienzeit, T_q = Duchlaufzeit

normalisierte Duchlaufzeit (T_q/T_s)

- ideal für A und B, unproblematisch für D
- schlecht für C
  - sie steht in einem extrem schlechten Verhältnis zur Bedienzeit T_s
  - typischer Effekt im Falle von kurzen Prozessen, die langen folgen

Verfahrensweisen

Verdrängend

Verdrängendes FCFS, Zeitscheiben, CPU-Schutz.

Verdrängendes FCFS, Zeitscheiben, CPU-Schutz.

- Prozesse werden nach ihrer Ankunftszeit ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - verdrängendes Verfahren, nutzt **periodische Unterbrechungen**
    - Zeitgeber (*timer*) liefert asynchrone Programmunterbrechungen
  - jeder Prozess erhält eine **Zeitscheibe** (*time slice*) zugeteilt
    - obere Schranke für die Rechenstoßlänge eines laufenden Prozesses

Verdrängendes FCFS, Zeitscheiben, CPU-Schutz.

- Prozesse werden nach ihrer Ankunftszeit ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - verdrängendes Verfahren, nutzt **periodische Unterbrechungen**
    - Zeitgeber (*timer*) liefert asynchrone Programmunterbrechungen
  - jeder Prozess erhält eine **Zeitscheibe** (*time slice*) zugeteilt
    - obere Schranke für die Rechenstoßlänge eines laufenden Prozesses
- Verringerung der bei FCFS auftretenden Benachteiligung von Prozessen mit kurzen Rechenstößen
  - die **Zeitscheibenlänge** bestimmt die Effektivität des Verfahrens
    - zu lang, Degenierung zu FCFS; zu kurz, sehr hoher Mehraufwand
  - Faustregel: etwas länger als die Dauer eines „typischen Rechenstoßes“

Verdrängendes FCFS, Zeitscheiben, CPU-Schutz.

- Prozesse werden nach ihrer Ankunftszeit ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - verdrängendes Verfahren, nutzt **periodische Unterbrechungen**
    - Zeitgeber (*timer*) liefert asynchrone Programmunterbrechungen
  - jeder Prozess erhält eine **Zeitscheibe** (*time slice*) zugeteilt
    - obere Schranke für die Rechenstoßlänge eines laufenden Prozesses
- Verringerung der bei FCFS auftretenden Benachteiligung von Prozessen mit kurzen Rechenstößen
  - die **Zeitscheibenlänge** bestimmt die Effektivität des Verfahrens
    - zu lang, Degenierung zu FCFS; zu kurz, sehr hoher Mehraufwand
  - Faustregel: etwas länger als die Dauer eines „typischen Rechenstoßes“
- Problem: **Konvoieffekt**
  - Prozesse kürzer als die Zeitscheibe folgen einem, der verdrängt wird…

RR: Konvoieffekt

- da die Zuteilung der Zeitscheiben an Prozesse nach FCFS geschieht, werden kurze Prozesse nach wie vor benachteiligt:

  E/A-intensive Prozesse schöpfen ihre Zeitscheibe selten voll aus
  - sie beenden ihren Rechenstoß freiwillig
    - vor Ablauf der Zeitscheibe

  CPU-intensive Prozesse schöpfen ihre Zeitscheibe meist voll aus
  - sie beenden ihren Rechenstoß unfreiwillig
    - durch Verdrängung

RR: Konvoieffekt

- da die Zuteilung der Zeitscheiben an Prozesse nach FCFS geschieht, werden kurze Prozesse nach wie vor benachteiligt:

  E/A-intensive Prozesse schöpfen ihre Zeitscheibe selten voll aus
  - sie beenden ihren Rechenstoß freiwillig
    - vor Ablauf der Zeitscheibe

  CPU-intensive Prozesse schöpfen ihre Zeitscheibe meist voll aus
  - sie beenden ihren Rechenstoß unfreiwillig
    - durch Verdrängung
- unabhängig davon werden jedoch alle Prozesse immer reihum bedient

RR: Konvoieffekt

- da die Zuteilung der Zeitscheiben an Prozesse nach FCFS geschieht, werden kurze Prozesse nach wie vor benachteiligt

  E/A-intensive Prozesse schöpfen ihre Zeitscheibe selten voll aus
  - sie beenden ihren Rechenstoß freiwillig
    - vor Ablauf der Zeitscheibe

  CPU-intensive Prozesse schöpfen ihre Zeitscheibe meist voll aus
  - sie beenden ihren Rechenstoß unfreiwillig
    - durch Verdrängung
  - unabhängig davon werden jedoch alle Prozesse immer reihum bedient

- wird eine Zeitscheibe durch einen Prozess nicht ausgeschöpft, verteilt sich die CPU-Zeit zu Ungunsten E/A-intensiver Prozesse
  - E/A-intensive Prozesse werden schlechter bedient
  - E/A-Geräte sind schlecht ausgelastet
  - Varianz der Antwortzeit E/A-intensiver Prozesse kann beträchtlich sein
    - in Abhängigkeit vom jeweiligen Mix von Prozessen

VRR

virtual round robin

RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

- auf E/A wartende Prozesse werden mit Beendigung ihres jeweiligen Ein-/Ausgabestoßes bevorzugt eingeplant (d.h., bereit gestellt)

RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

- auf E/A wartende Prozesse werden mit Beendigung ihres jeweiligen Ein-/Ausgabestoßes bevorzugt eingeplant (d.h., bereit gestellt)
  - **Einplanung** mittels einer der Bereitliste vorgeschalteten **Vorzugsliste**
    - FIFO $\leadsto$ evtl. Benachteiligung hoch-interaktiver Prozesse; daher…
    - aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses

RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

- auf E/A wartende Prozesse werden mit Beendigung ihres jeweiligen Ein-/Ausgabestoßes bevorzugt eingeplant (d.h., bereit gestellt)
  - **Einplanung** mittels einer der Bereitliste vorgeschalteten **Vorzugsliste**
    - FIFO $\rightsquigarrow$ evtl. Benachteiligung hoch-interaktiver Prozesse; daher…
    - aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses
  - **Umplanung** bei Ablauf der aktuellen Zeitscheibe
    - die Prozesse auf der Vorzugsliste werden zuerst eingelastet
    - sie bekommen die CPU für die Restdauer ihrer Zeitscheibe zugeteilt
    - bei Ablauf dieser Zeitscheibe werden sie in die Bereitsliste eingereiht

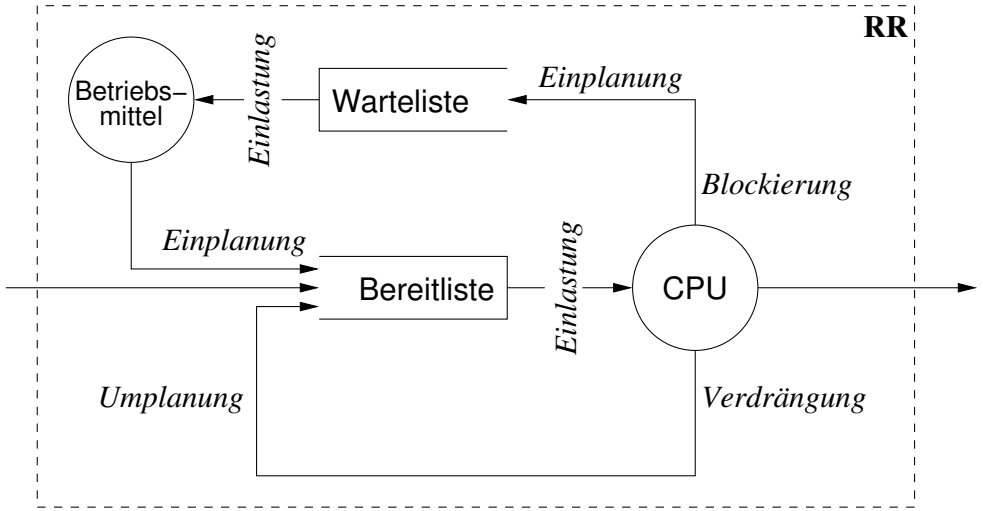
RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

- auf E/A wartende Prozesse werden mit Beendigung ihres jeweiligen Ein-/Ausgabestoßes bevorzugt eingeplant (d.h., bereit gestellt)
  - **Einplanung** mittels einer der Bereitliste vorgeschalteten **Vorzugsliste**
    - FIFO $\rightsquigarrow$ evtl. Benachteiligung hoch-interaktiver Prozesse; daher…
    - aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses
  - **Umplanung** bei Ablauf der aktuellen Zeitscheibe
    - die Prozesse auf der Vorzugsliste werden zuerst eingelastet
    - sie bekommen die CPU für die Restdauer ihrer Zeitscheibe zugeteilt
    - bei Ablauf dieser Zeitscheibe werden sie in die Bereitsliste eingereiht
  - erreicht durch strukturelle Maßnahmen — nicht durch analytische

RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

- auf E/A wartende Prozesse werden mit Beendigung ihres jeweiligen Ein-/Ausgabestoßes bevorzugt eingeplant (d.h., bereit gestellt)
  - **Einplanung** mittels einer der Bereitliste vorgeschalteten **Vorzugsliste**
    - FIFO $\rightsquigarrow$ evtl. Benachteiligung hoch-interaktiver Prozesse; daher…
    - aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses
  - **Umplanung** bei Ablauf der aktuellen Zeitscheibe
    - die Prozesse auf der Vorzugsliste werden zuerst eingelastet
    - sie bekommen die CPU für die Restdauer ihrer Zeitscheibe zugeteilt
    - bei Ablauf dieser Zeitscheibe werden sie in die Bereitsliste eingereiht
  - erreicht durch strukturelle Maßnahmen — nicht durch analytische
- kein voll-verdrängendes Verfahren
  - die Einlastung auch des kürzesten bereitgestellten Prozesses erfolgt nicht zum Zeitpunkt seiner Bereitstellung
  - sondern frühestens nach Ablauf der aktuellen Zeitscheibe

RR vs. VRR



Bereitliste
- lauffähiger Fäden[a]
- dreiseitig bestückt
  - 2 × Einplanung
  - 1 × Umplanung
- unbedingt bedient

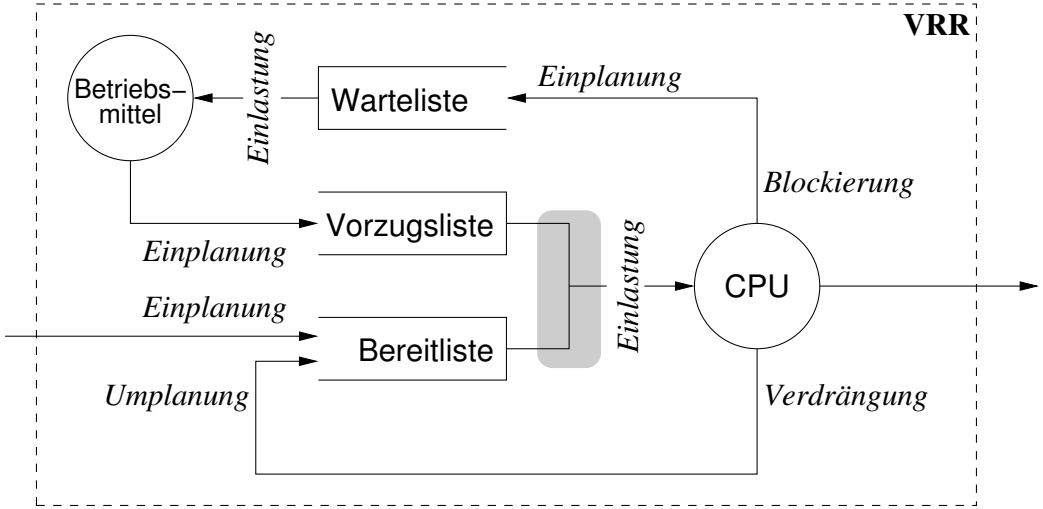
Warteliste
- blockierter Fäden

[a] CPU „Warteliste“

Bereitliste
- wie bei RR
- 2-seitig bestückt
  - 1 × Einplanung
  - 1 × Umplanung
- bedingt bedient

Warteliste
- wie bei RR

Vorzugsliste
- unbedingt bedient



Verfahrensweisen

Probabilistisch

SPN

shortest process next

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

SPN

shortest process next

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant
  - Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:

    Stapelbetrieb Programmierer setzen **Frist** (*time limit*)

    Produktionsbetrieb Erstellung einer **Statistik** durch Probeläufe

    Dialogbetrieb **Abschätzung** von Rechenstoßlängen zur Laufzeit

SPN

shortest process next

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant
  - Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:

    Stapelbetrieb Programmierer setzen **Frist** (*time limit*)

    Produktionsbetrieb Erstellung einer **Statistik** durch Probeläufe

    Dialogbetrieb **Abschätzung** von Rechenstoßlängen zur Laufzeit
  - Abarbeitung einer aufsteigend nach Laufzeiten sortierten Bereitsliste
    - Abschätzung erfolgt vor (statisch) oder zur (dynamisch) Laufzeit

SPN

shortest process next

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant
  - Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:

    Stapelbetrieb Programmierer setzen **Frist** (*time limit*)

    Produktionsbetrieb Erstellung einer **Statistik** durch Probeläufe

    Dialogbetrieb **Abschätzung** von Rechenstoßlängen zur Laufzeit
  - Abarbeitung einer aufsteigend nach Laufzeiten sortierten Bereitsliste
    - Abschätzung erfolgt vor (statisch) oder zur (dynamisch) Laufzeit
- Verkürzung von Antwortzeiten und Steigerung der Gesamtleistung des Systems bei Benachteiligung längerer Prozesse
  - ein **Verhungern** (*starvation*) dieser Prozesse ist möglich

SPN

shortest process next

Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant
  - Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:

    Stapelbetrieb Programmierer setzen **Frist** (*time limit*)

    Produktionsbetrieb Erstellung einer **Statistik** durch Probeläufe

    Dialogbetrieb **Abschätzung** von Rechenstoßlängen zur Laufzeit
  - Abarbeitung einer aufsteigend nach Laufzeiten sortierten Bereitsliste
    - Abschätzung erfolgt vor (statisch) oder zur (dynamisch) Laufzeit
- Verkürzung von Antwortzeiten und Steigerung der Gesamtleistung des Systems bei Benachteiligung längerer Prozesse
  - ein **Verhungern** (*starvation*) dieser Prozesse ist möglich
- ohne Konvoi-Effekt — jedoch ist als praktikable Implementierung nur die **näherungsweise Lösung** möglich
  - da die Rechenstoßlängen nicht exakt im Voraus bestimmbar
  - die obere Grenze einer Stoßlänge nicht selten auch unvorhersagbar ist

- ein **heuristisches Verfahren**, das für jeden Prozess den Mittelwert über seine jeweiligen Rechenstoßlängen bildet
  - damit ist die erwarte Länge des nächsten Rechenstoßes eines Prozesses:

$$S_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^{n} T_i = \frac{1}{n} \cdot T_n + \frac{n-1}{n} \cdot S_n$$

    - arithmetisches Mittel aller gemessenen Rechenstoßlängen des Prozesses

- ein **heuristisches Verfahren**, das für jeden Prozess den Mittelwert über seine jeweiligen Rechenstoßlängen bildet
  - damit ist die erwarte Länge des nächsten Rechenstoßes eines Prozesses:

$$S_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^{n} T_i = \frac{1}{n} \cdot T_n + \frac{n-1}{n} \cdot S_n$$

    - arithmetisches Mittel aller gemessenen Rechenstoßlängen des Prozesses
  - Problem dieser Berechnung ist die **gleiche Gewichtung** aller Rechenstöße
  - jüngere Rechenstöße machen jedoch die **Lokalität** eines Prozesse aus
    - diesen Stößen sollte eine größere Gewichtung gegeben werden (vgl. S. 70)

- ein **heuristisches Verfahren**, das für jeden Prozess den Mittelwert über seine jeweiligen Rechenstoßlängen bildet
  - damit ist die erwarte Länge des nächsten Rechenstoßes eines Prozesses:

$$S_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^{n} T_i = \frac{1}{n} \cdot T_n + \frac{n-1}{n} \cdot S_n$$

    - arithmetisches Mittel aller gemessenen Rechenstoßlängen des Prozesses
  - Problem dieser Berechnung ist die **gleiche Gewichtung** aller Rechenstöße
  - jüngere Rechenstöße machen jedoch die **Lokalität** eines Prozesse aus
    - diesen Stößen sollte eine größere Gewichtung gegeben werden (vgl. S. 70)
- die **Messung** der Dauer eines Rechenstoßes geschieht im Moment der Prozesseinlastung (d.h., der Prozessumschaltung)
  - Stoppzeit T_2 von P_j entspricht (in etwa) der Startzeit T_1 von P_{j+1}
    - gemessen in Uhrzeit (*clock time*) oder Uhrtick (*clock tick*)
  - dann ergibt $T_2 - T_1$ die gemessene Rechenstoßlänge für jeden Prozess P_i
  - der Differenzwert wird im jeweiligen Prozesskontrollblock akkumuliert

SPN: Gewichtung

- mittels **Dämpfungsfiler** (*decay filter*), d.h., der Dämpfung (*decay*) der am weitesten zurückliegenden Rechenstöße:

$$S_{n+1} \quad = \quad \alpha \cdot T_n + (1 - \alpha) \cdot S_n$$

  - mit zuletzt gemessener (T_n) und geschätzter (S_n) Rechenstoßlänge

  - (Dieses statistische Verfahren nennt man auch exponentielle Glättung.)

SPN: Gewichtung

- mittels **Dämpfungsfiler** (*decay filter*), d.h., der Dämpfung (*decay*) der am weitesten zurückliegenden Rechenstöße:

$$S_{n+1} \quad = \quad \alpha \cdot T_n + (1 - \alpha) \cdot S_n$$

  - mit zuletzt gemessener (T_n) und geschätzter (S_n) Rechenstoßlänge
  - für den konstanten **Gewichtungsfaktor** α gilt dabei: $0 < \alpha < 1$
    - drückt die relative Gewichtung einzelner Rechenstöße der Zeitreihe aus
  - (Dieses statistische Verfahren nennt man auch exponentielle Glättung.)

SPN: Gewichtung

- mittels **Dämpfungsfiler** (*decay filter*), d.h., der Dämpfung (*decay*) der am weitesten zurückliegenden Rechenstöße:

$$S_{n+1} \quad = \quad \alpha \cdot T_n + (1 - \alpha) \cdot S_n$$

  - mit zuletzt gemessener (T_n) und geschätzter (S_n) Rechenstoßlänge
  - für den konstanten **Gewichtungsfaktor** α gilt dabei: $0 < \alpha < 1$
    - drückt die relative Gewichtung einzelner Rechenstöße der Zeitreihe aus
  - (Dieses statistische Verfahren nennt man auch exponentielle Glättung.)

- um die Wirkung des Gewichtungsfaktors zu verdeutlichen, die teilweise Expansion der Gleichung wie folgt:
  - $S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \ldots + (1 - \alpha)^i \alpha T_{n-1} + \ldots + (1 - \alpha)^n S_1$

SPN: Gewichtung

- mittels **Dämpfungsfiler** (*decay filter*), d.h., der Dämpfung (*decay*) der am weitesten zurückliegenden Rechenstöße:

$$S_{n+1} \quad = \quad \alpha \cdot T_n + (1 - \alpha) \cdot S_n$$

  - mit zuletzt gemessener (T_n) und geschätzter (S_n) Rechenstoßlänge
  - für den konstanten **Gewichtungsfaktor** α gilt dabei: $0 < \alpha < 1$
    - drückt die relative Gewichtung einzelner Rechenstöße der Zeitreihe aus
  - (Dieses statistische Verfahren nennt man auch exponentielle Glättung.)

- um die Wirkung des Gewichtungsfaktors zu verdeutlichen, die teilweise Expansion der Gleichung wie folgt:
  - $S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \ldots + (1 - \alpha)^i \alpha T_{n-1} + \ldots + (1 - \alpha)^n S_1$

- Beispiel der Entwicklung für $\alpha = 0.8$:
  - $S_{n+1} = 0.8\,T_n + 0.16\,T_{n-1} + 0.032\,T_{n-2} + 0.0064\,T_{n-3} + \ldots$
  - zurückliegende Rechenstöße des Prozesses verlieren schnell an Gewicht

HRRN

highest response ratio next

Hungerfreies SPN.

Hungerfreies SPN.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und periodisch unter Berücksichtigung ihrer **Wartezeit** umgeplant

Hungerfreies SPN.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und periodisch unter Berücksichtigung ihrer **Wartezeit** umgeplant
  - in regelmäßigen Zeitabständen wird ein Verhältniswert R berechnet:

$$R = \frac{w + s}{s}$$

    w aktuell abgelaufene Wartezeit eines Prozesses
    s erwartete (d.h., abgeschätzte) Bedienzeit eines Prozesses
  - ausgewählt wird der Prozess mit dem größten Verhältniswert R

Hungerfreies SPN.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und periodisch unter Berücksichtigung ihrer **Wartezeit** umgeplant
  - in regelmäßigen Zeitabständen wird ein Verhältniswert R berechnet:

$$R = \frac{w + s}{s}$$

    w aktuell abgelaufene Wartezeit eines Prozesses
    s erwartete (d.h., abgeschätzte) Bedienzeit eines Prozesses
  - ausgewählt wird der Prozess mit dem größten Verhältniswert R
- die periodische Aktualisierung betrifft alle Einträge in der Bereitliste und findet im Hintergrund des aktuellen Prozesses statt
  - ausgelöst durch einen Uhrtick (*clock tick*)

Hungerfreies SPN.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und periodisch unter Berücksichtigung ihrer **Wartezeit** umgeplant
  - in regelmäßigen Zeitabständen wird ein Verhältniswert R berechnet:

$$R = \frac{w + s}{s}$$

    w aktuell abgelaufene Wartezeit eines Prozesses
    s erwartete (d.h., abgeschätzte) Bedienzeit eines Prozesses
  - ausgewählt wird der Prozess mit dem größten Verhältniswert R
- die periodische Aktualisierung betrifft alle Einträge in der Bereitliste und findet im Hintergrund des aktuellen Prozesses statt
  - ausgelöst durch einen Uhrtick (*clock tick*)
- Anmerkung: ein Anstieg der Wartezeit eines Prozesses bedeutet seine **Alterung** (*aging*)
  - der Alterung entgegenwirken beugt Verhungern (*starvation*) vor

SRTF

shortest remaining time first

Verdrängendes SPN, Hungergefahr, Effektivität von VRR.

Verdrängendes SPN, Hungergefahr, Effektivität von VRR.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **spontan** umgeplant

Verdrängendes SPN, Hungergefahr, Effektivität von VRR.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **spontan** umgeplant
  - sei T_{et} die erwartete Rechenstoßlänge eines eintreffenden Prozesses
  - sei T_{rt} die verbleibende Rechenstoßlänge des laufenden Prozesses
  - der laufende Prozess wird verdrängt, wenn gilt: $T_{et} < T_{rt}$

Verdrängendes SPN, Hungergefahr, Effektivität von VRR.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **spontan** umgeplant
  - sei T_{et} die erwartete Rechenstoßlänge eines eintreffenden Prozesses
  - sei T_{rt} die verbleibende Rechenstoßlänge des laufenden Prozesses
  - der laufende Prozess wird verdrängt, wenn gilt: $T_{et} < T_{rt}$
- die **Umplanung** erfolgt ereignisbedingt und (ggf. voll) verdrängend im Moment der Ankunftszeit eines Prozesses
  - z.B. bei Beendigung des Ein-/Ausgabestoßes eines wartenden Prozesses
  - allgemein: bei Aufhebung der Wartebedingung für einen Prozess

SRTF

shortest remaining time first

Verdrängendes SPN, Hungergefahr, Effektivität von VRR.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **spontan** umgeplant
  - sei T_{et} die erwartete Rechenstoßlänge eines eintreffenden Prozesses
  - sei T_{rt} die verbleibende Rechenstoßlänge des laufenden Prozesses
  - der laufende Prozess wird verdrängt, wenn gilt: $T_{et} < T_{rt}$
- die **Umplanung** erfolgt ereignisbedingt und (ggf. voll) verdrängend im Moment der Ankunftszeit eines Prozesses
  - z.B. bei Beendigung des Ein-/Ausgabestoßes eines wartenden Prozesses
  - allgemein: bei Aufhebung der Wartebedingung für einen Prozess
- bei **Verdrängung** kommt der betreffende Prozess entsprechend der Restdauer seiner erwarteten Rechenstoßlänge auf die Bereitliste
  - führt allgemein zu besseren Antwort- und Durchlaufzeiten
  - gegenüber VRR steht der Aufwand zur Rechenstoßlängenabschätzung

Verfahrensweisen

Mehrstufig

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant
  - Aufteilung der Bereitliste in separate („getypte“) Listen
    - z.B. für System-, Dialog- und Stapelprozesse

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant
  - Aufteilung der Bereitliste in separate („getypte“) Listen
    - z.B. für System-, Dialog- und Stapelprozesse
  - mit jeder Liste eine **lokale Einplanungsstrategie** verbinden
    - z.B. SPN, RR und FCFS

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant
  - Aufteilung der Bereitliste in separate („getypte“) Listen
    - z.B. für System-, Dialog- und Stapelprozesse
  - mit jeder Liste eine **lokale Einplanungsstrategie** verbinden
    - z.B. SPN, RR und FCFS
  - zwischen den Listen eine **globale Einplanungsstrategie** definieren

    statisch
    - Liste einer bestimmten Prioritätsebene fest zuordnen
    - Hungergefahr für Prozesse tiefer liegender Listen

    dynamisch
    - die Listen im Zeitmultiplexverfahren wechseln
    - z.B. 40 % System-, 40 % Dialog-, 20 % Stapelprozesse

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant
  - Aufteilung der Bereitliste in separate („getypte“) Listen
    - z.B. für System-, Dialog- und Stapelprozesse
  - mit jeder Liste eine **lokale Einplanungsstrategie** verbinden
    - z.B. SPN, RR und FCFS
  - zwischen den Listen eine **globale Einplanungsstrategie** definieren

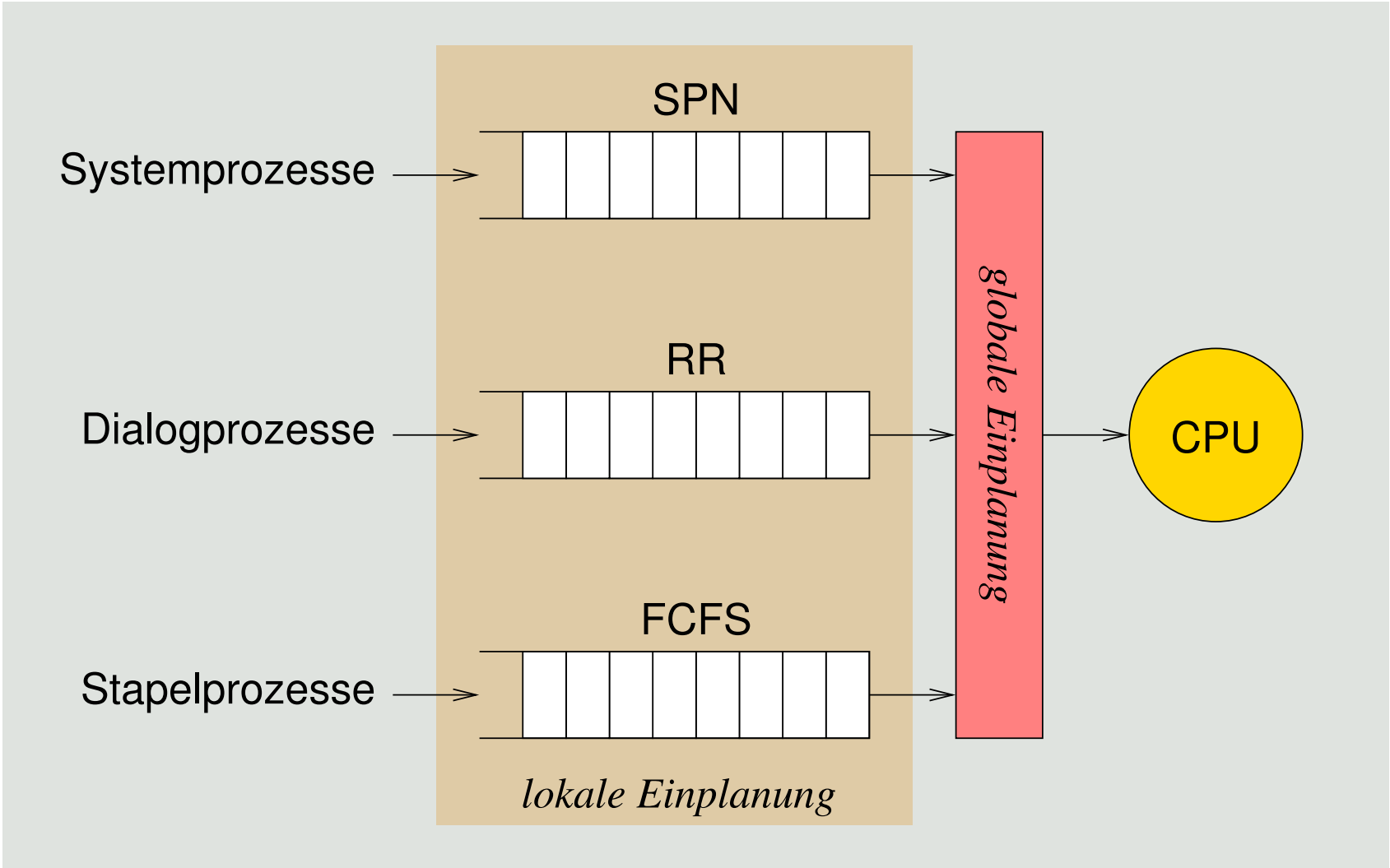
    statisch
    - Liste einer bestimmten Prioritätsebene fest zuordnen
    - Hungergefahr für Prozesse tiefer liegender Listen

    dynamisch
    - die Listen im Zeitmultiplexverfahren wechseln
    - z.B. 40 % System-, 40 % Dialog-, 20 % Stapelprozesse

- dem Prozess einen Typen zuordnen ist eine statische Entscheidung
  - sie wird zum Zeitpunkt der Prozesserzeugung getroffen

MLQ: Mischbetrieb

System-, Dialog- und Stapelprozesse im Mix



Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant

Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - Hierarchie von Bereitlisten, je nach Anzahl der **Prioritätsebenen**
    - erstmalig eintreffende Prozesse steigen oben ein
    - Zeitscheibenablauf drückt den laufenden Prozess weiter nach unten
  - je nach Ebene verschiedene Einreihungsstrategien und -parameter
    - unterste Ebene arbeitet nach RR, alle anderen (höheren) nach FCFS
    - die Zeitscheibengrößen nehmen von oben nach unten zu

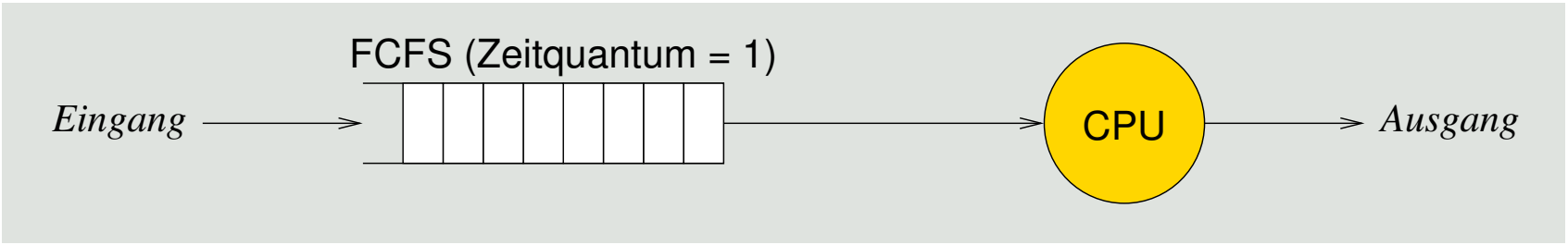
Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - Hierarchie von Bereitlisten, je nach Anzahl der **Prioritätsebenen**
    - erstmalig eintreffende Prozesse steigen oben ein
    - Zeitscheibenablauf drückt den laufenden Prozess weiter nach unten
  - je nach Ebene verschiedene Einreihungsstrategien und -parameter
    - unterste Ebene arbeitet nach RR, alle anderen (höheren) nach FCFS
    - die Zeitscheibengrößen nehmen von oben nach unten zu
- **Bestrafung** (*penalisation*)
  - Prozesse mit langen Rechenstößen fallen nach unten durch
  - Prozesse mit kurzen Rechenstößen laufen relativ schnell durch

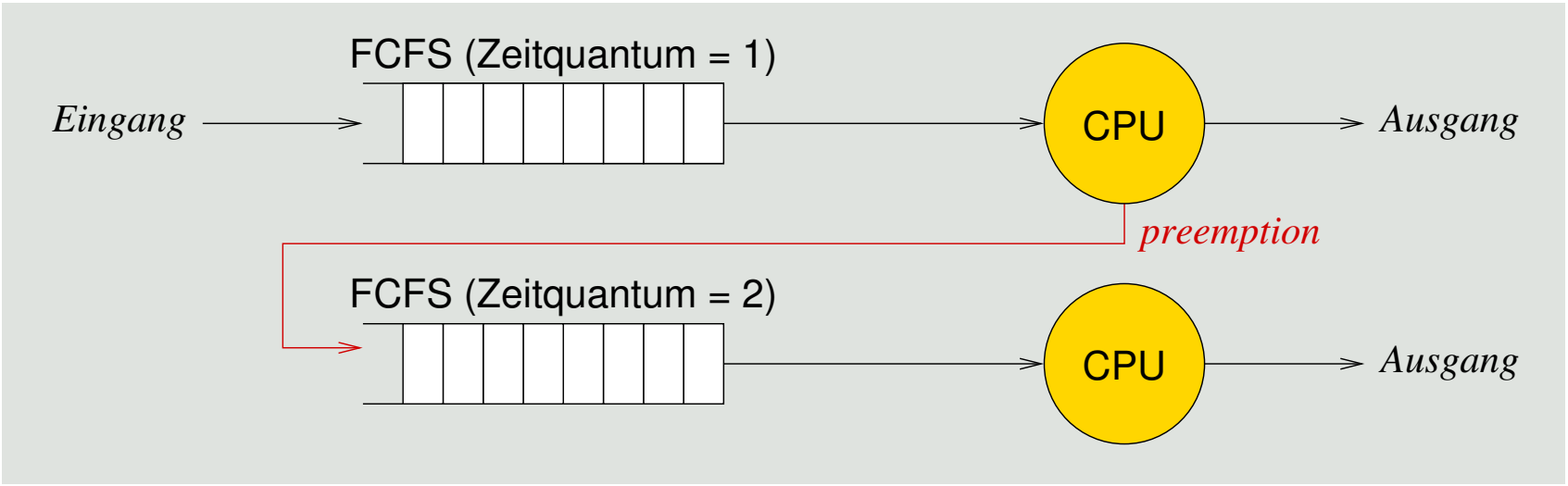
Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - Hierarchie von Bereitlisten, je nach Anzahl der **Prioritätsebenen**
    - erstmalig eintreffende Prozesse steigen oben ein
    - Zeitscheibenablauf drückt den laufenden Prozess weiter nach unten
  - je nach Ebene verschiedene Einreihungsstrategien und -parameter
    - unterste Ebene arbeitet nach RR, alle anderen (höheren) nach FCFS
    - die Zeitscheibengrößen nehmen von oben nach unten zu
- **Bestrafung** (*penalisation*)
  - Prozesse mit langen Rechenstößen fallen nach unten durch
  - Prozesse mit kurzen Rechenstößen laufen relativ schnell durch
- **Alterung** (*ageing*)
  - nach unten durchfallende Prozesse finden seltener statt
  - durchgefallene Prozesse nach einer **Bewährungsfrist** wieder anheben

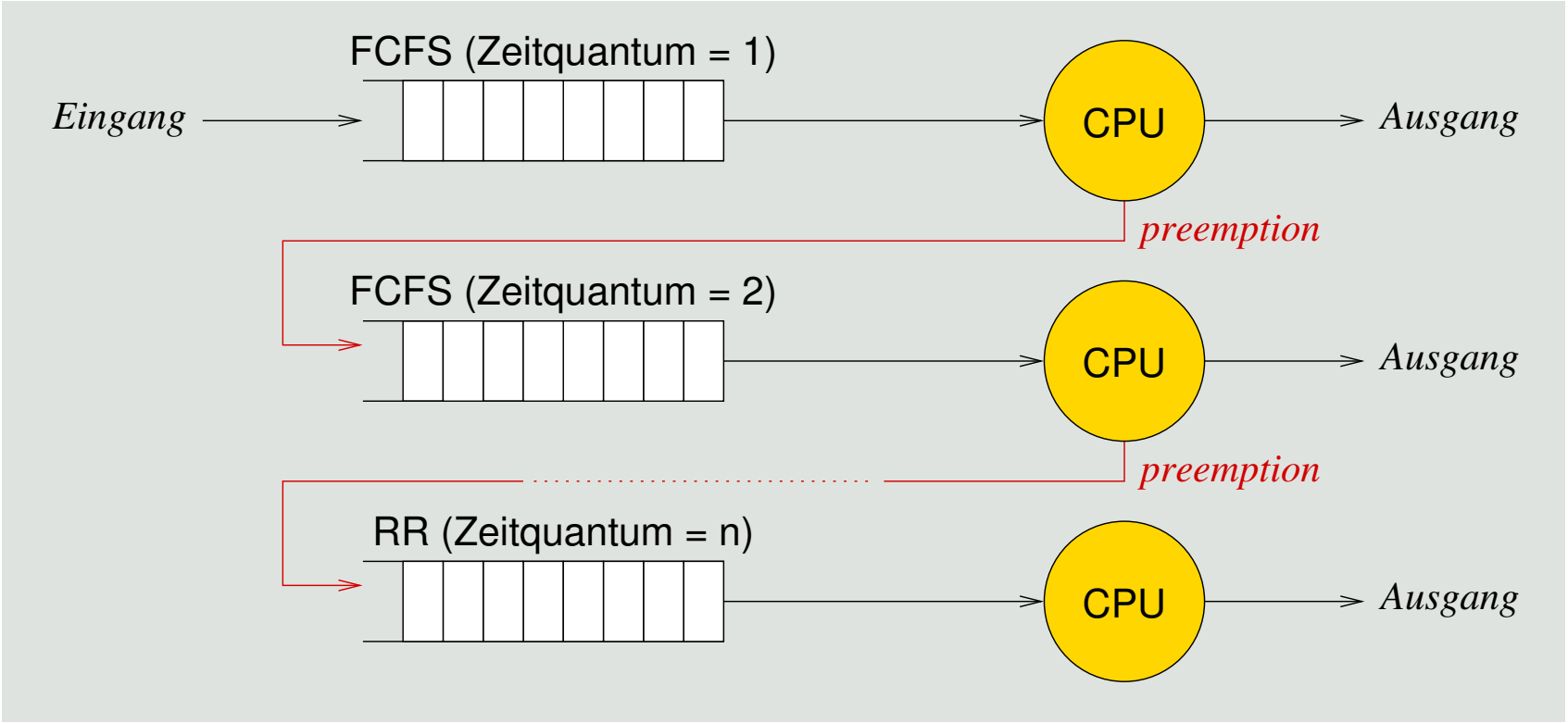
MLFQ: Bestrafung



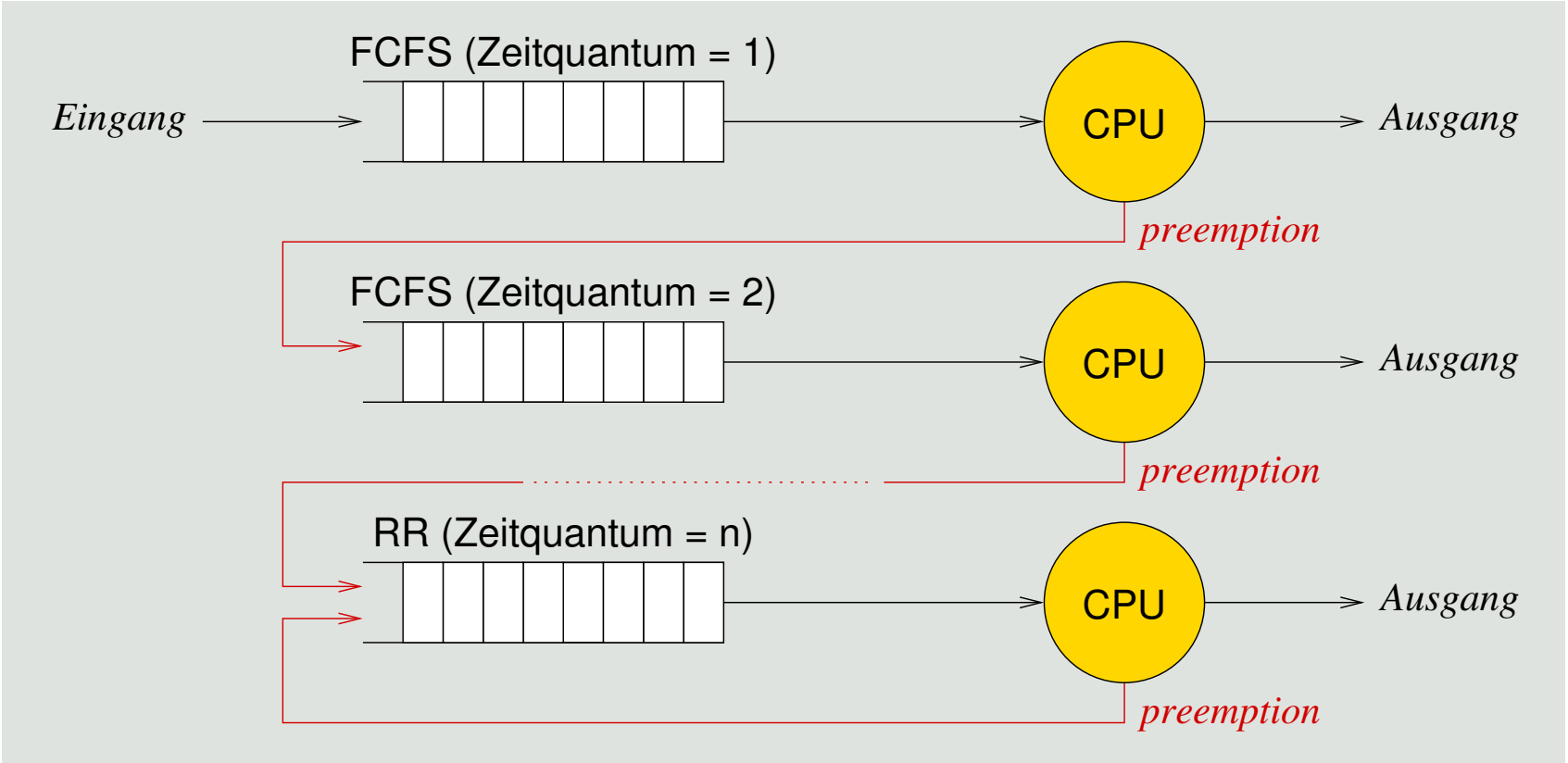
MLFQ: Bestrafung



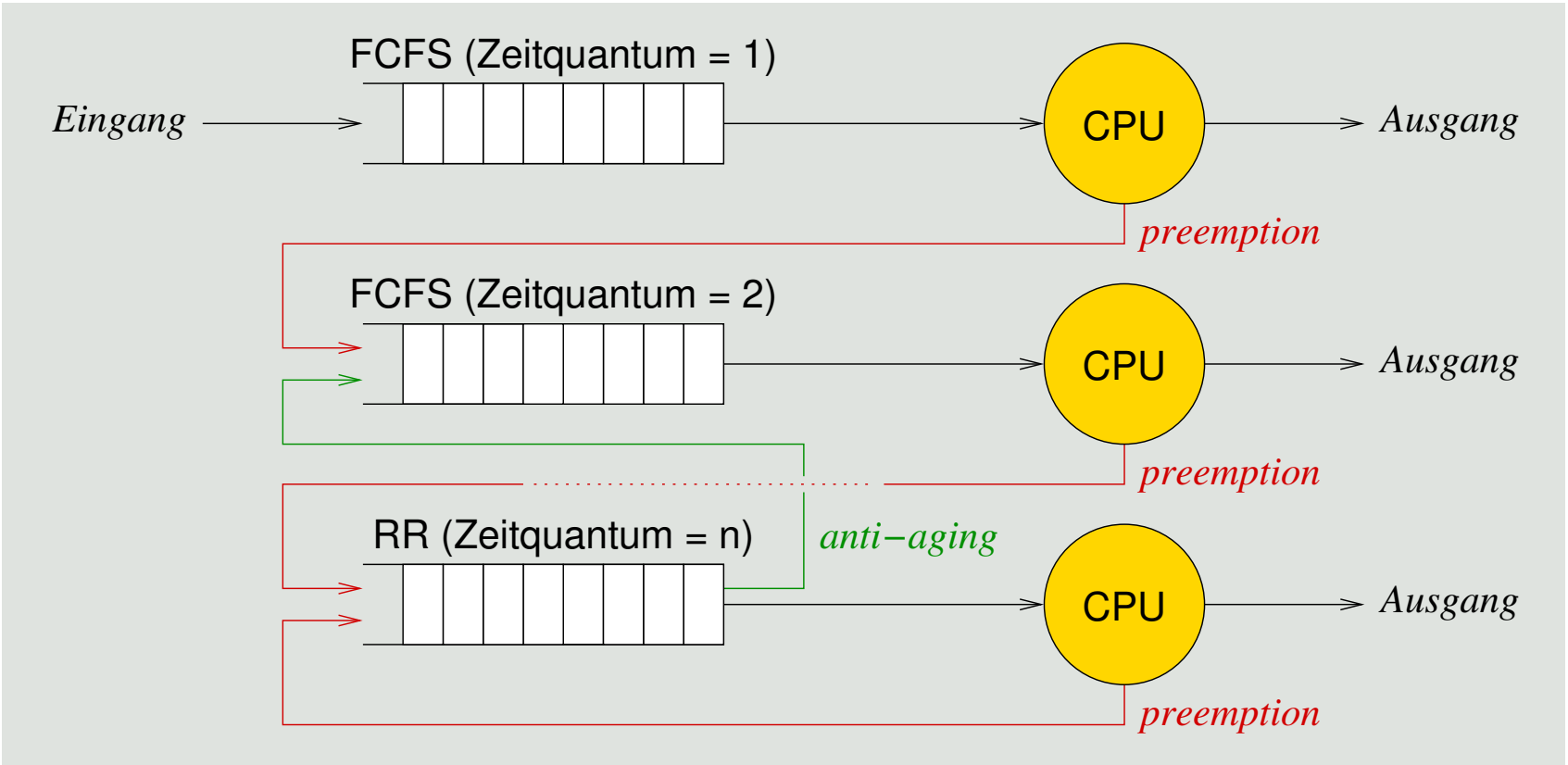
MLFQ: Bestrafung



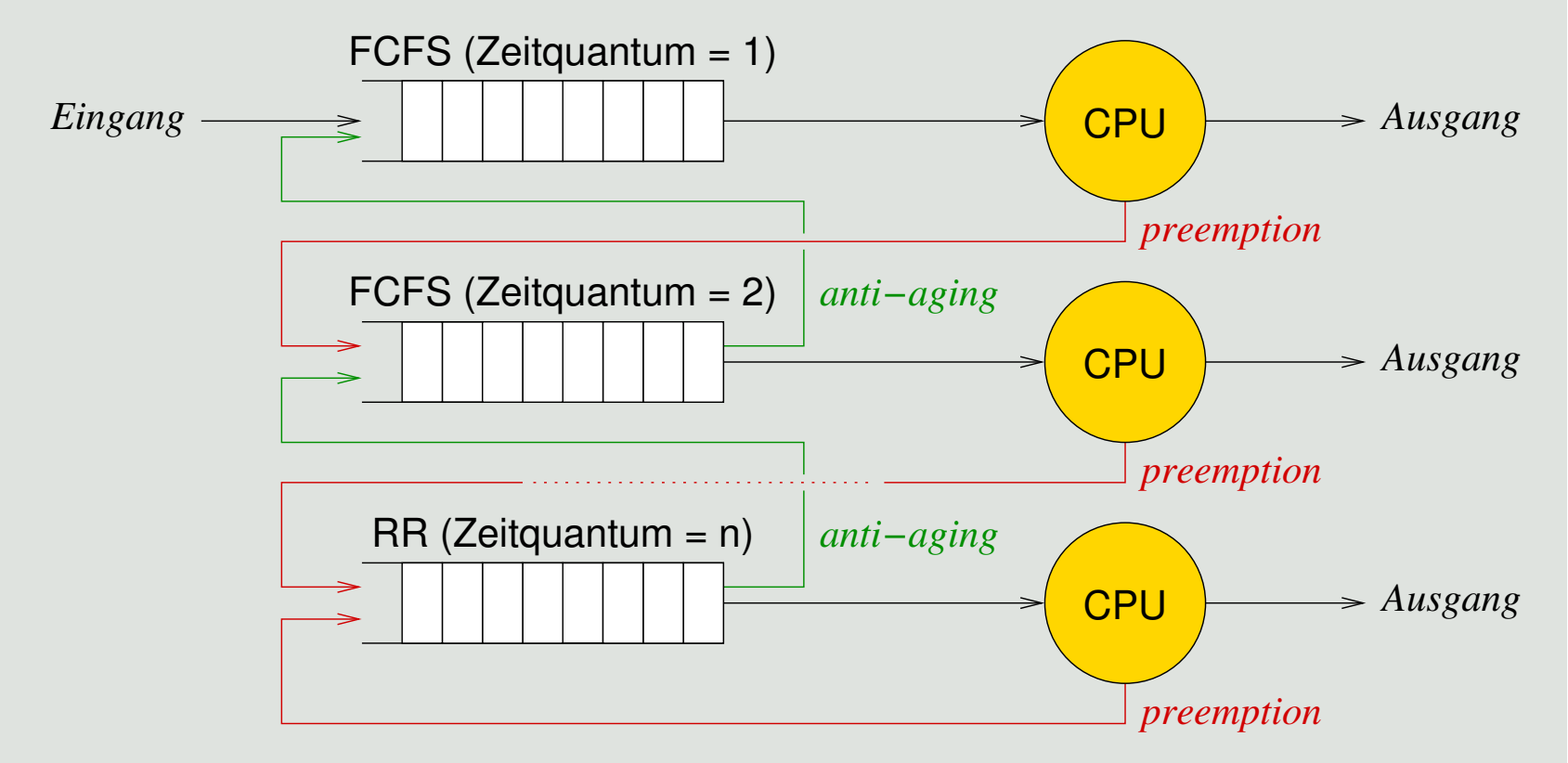
MLFQ: Bestrafung



MLFQ: Bestrafung und Bewährung



MLFQ: Bestrafung und Bewährung



feedback (FB)

Gliederung

Gegenüberstellung

	FCFS	RR	VRR	SPN	HRRN	SRTF
kooperativ	✓			(✓)	(✓)	
verdrängend		✓	✓			✓
probabilistisch				✓	✓	✓
deterministisch	keine bzw. nicht von sich aus allein $\rightsquigarrow$ EZS [13]					

Gegenüberstellung

	FCFS	RR	VRR	SPN	HRRN	SRTF
kooperativ	✓			(✓)	(✓)	
verdrängend		✓	✓			✓
probabilistisch				✓	✓	✓
deterministisch	keine bzw. nicht von sich aus allein $\rightsquigarrow$ EZS [13]					

- MLQ und MLFQ erlauben eine Kombination dieser Verfahren, jedoch abgestuft und nicht alle zusammen auf derselben Ebene
  - dadurch wird letztlich eine **Priorisierung** der Strategien vorgenommen
    - entsprechend der globalen Strategie, die den Ebenenwechsel steuert
  - teilweise wird so speziellen Anwendungsbedürfnissen entgegengekommen
    - z.B. FCFS priorisieren $\rightsquigarrow$ „*number crunching*“ fördern

Gegenüberstellung

	FCFS	RR	VRR	SPN	HRRN	SRTF
kooperativ	✓			(✓)	(✓)	
verdrängend		✓	✓			✓
probabilistisch				✓	✓	✓
deterministisch	keine bzw. nicht von sich aus allein $\rightsquigarrow$ EZS [13]					

- MLQ und MLFQ erlauben eine Kombination dieser Verfahren, jedoch abgestuft und nicht alle zusammen auf derselben Ebene
  - dadurch wird letztlich eine **Priorisierung** der Strategien vorgenommen
    - entsprechend der globalen Strategie, die den Ebenenwechsel steuert
  - teilweise wird so speziellen Anwendungsbedürfnissen entgegengekommen
    - z.B. FCFS priorisieren $\rightsquigarrow$ „*number crunching*“ fördern
- jedes dieser Verfahren stellt bestimmte **Gütemerkmale** [7] in den Vordergrund und vergibt damit indirekt Prioritäten an Prozesse

Prioritäten setzende Verfahren

Statische Prioritäten (MLQ) vs. dynamische Prioritäten (VRR, SPN, SRTF, HRRN, MLFQ).

- **Prozessvorrang** bedeutet die bevorzugte Einlastung von Prozessen mit höherer Priorität

Prioritäten setzende Verfahren

Statische Prioritäten (MLQ) vs. dynamische Prioritäten (VRR, SPN, SRTF, HRRN, MLFQ).

- **Prozessvorrang** bedeutet die bevorzugte Einlastung von Prozessen mit höherer Priorität und wird auf zwei Arten bestimmt:

  statisch
  - Zeitpunkt der **Prozesserzeugung** $\rightsquigarrow$ Laufzeitkonstante
  - wird im weiteren Verlauf nicht mehr verändert
  - erzwingt die deterministische Ordnung zw. Prozessen

  dynamisch
  - „jederzeit“ im **Prozessintervall** $\rightsquigarrow$ Laufzeitvariable
  - die Berechnung erfolgt durch das Betriebssystem
    - ggf. in Kooperation mit den Anwendungsprogrammen
  - erzwingt keine deterministische Ordnung zw. Prozessen

Prioritäten setzende Verfahren

Statische Prioritäten (MLQ) vs. dynamische Prioritäten (VRR, SPN, SRTF, HRRN, MLFQ).

- **Prozessvorrang** bedeutet die bevorzugte Einlastung von Prozessen mit höherer Priorität und wird auf zwei Arten bestimmt:
  - **statisch**
    - Zeitpunkt der **Prozesserzeugung** $\rightsquigarrow$ Laufzeitkonstante
    - wird im weiteren Verlauf nicht mehr verändert
    - erzwingt die deterministische Ordnung zw. Prozessen
  - **dynamisch**
    - „jederzeit“ im **Prozessintervall** $\rightsquigarrow$ Laufzeitvariable
    - die Berechnung erfolgt durch das Betriebssystem
      - ggf. in Kooperation mit den Anwendungsprogrammen
    - erzwingt keine deterministische Ordnung zw. Prozessen
- damit ist allerdings noch nicht **Echtzeitverarbeitung** garantiert, bei der Prozessvorrang eine maßgebliche Rolle spielt
  - die **Striktheit von Terminvorgaben** ist einzuhalten: weich, fest, hart
    - entsprechend der jeweiligen Anforderungen der Anwendungsdomäne
  - keines der behandelten Verfahren sichert dies dem Anwendungssystem zu

Resümee

Resümee

- Prozesseinplanung unterliegt einer breit gefächerten **Einordnung**
  - kooperativ/verdrängend
  - deterministisch/probabilistisch
  - statisch/dynamisch
  - asymmetrisch/symmetrisch

Resümee

- die entsprechenden **Verfahrensweisen** sind z.T. sehr unterschiedlich
  - FCFS: kooperativ
  - RR, VRR: verdrängend
  - SPN, HRRN, SRTF: probabilistisch
  - MLQ, MLFQ (FB): mehrstufig

Resümee

- Prioritäten setzende Verfahren legen einen **Prozessvorrang** fest
  - FCFS: Ankunftszeit
  - RR: Ankunftszeit, VRR: Ankunftszeit nach Ein-/Ausgabestoßende
  - SPN: Rechenstoß, HRRN: Verhältniswert, SRTF: Rechenstoßrest

Resümee

- eine weitere Dimension ist die **Striktheit von Terminvorgaben**
  - die jedoch keins der behandelten Verfahren an sich berücksichtigt…

Resümee

…Fokus lag auf Durschnittsleistung (*average performance*)

- Prozesseinplanung unterliegt einer breit gefächerten **Einordnung**
  - kooperativ/verdrängend
  - deterministisch/probabilistisch
  - statisch/dynamisch
  - asymmetrisch/symmetrisch
- die entsprechenden **Verfahrensweisen** sind z.T. sehr unterschiedlich
  - FCFS: kooperativ
  - RR, VRR: verdrängend
  - SPN, HRRN, SRTF: probabilistisch
  - MLQ, MLFQ (FB): mehrstufig
- Prioritäten setzende Verfahren legen einen **Prozessvorrang** fest
  - FCFS: Ankunftszeit
  - RR: Ankunftszeit, VRR: Ankunftszeit nach Ein-/Ausgabestoßende
  - SPN: Rechenstoß, HRRN: Verhältniswert, SRTF: Rechenstoßrest
- eine weitere Dimension ist die **Striktheit von Terminvorgaben**
  - die jedoch keins der behandelten Verfahren an sich berücksichtigt…

Zusammenfassung

Bibliographie

Literaturverzeichnis (1)

[1] Bayer, R. :
Symmetric binary B-Trees: Data structure and maintenance algorithms.
In: *Acta Informatica* 1 (1972), Dezember, S. 290–306

[2] Coffman, E. G. ; Denning, P. J.:
***Operating System Theory*.**
Prentice Hall, Inc., 1973

[3] Conway, R. W. ; Maxwell, L. W. ; Millner, L. W.:
***Theory of Scheduling*.**
Addison-Wesley, 1967

[4] Guibas, L. J. ; Sedgewick, R. :
A dichromatic framework for balanced trees.
In: *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (SFCS 1978)*, IEEE, 1978, S. 8–21

Literaturverzeichnis (2)

[5] Hönig, T. :
Der O(1)-Scheduler im Kernel 2.6.
In: *Linux Magazin* (2004), Februar, Nr. 2

[6] Kleinöder, J. ; Schröder-Preikschat, W. :
Dialog- und Echtzeitverarbeitung.
In: [9], Kapitel 7.2

[7] Kleinöder, J. ; Schröder-Preikschat, W. :
Einplanungsgrundlagen.
In: [9], Kapitel 9.1

[8] Kleinöder, J. ; Schröder-Preikschat, W. :
Prozesse.
In: [9], Kapitel 6.1

Literaturverzeichnis (3)

[9] Kleinöder, J. ; Schröder-Preikschat, W. ; Lehrstuhl Informatik 4 (Hrsg.):
Systemprogrammierung.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien)

[10] Kleinrock, L. :
***Queuing Systems.* Bd. I: Theory.**
John Wiley & Sons, 1975

[11] Kornai, J. :
Economics of Shortage.
North-Holland Publishing Company, 1980

[12] Lister, A. M. ; Eager, R. D.:
Fundamentals of Operating Systems.
The Macmillan Press Ltd., 1993. –
ISBN 0–333–59848–2

Literaturverzeichnis (4)

[13] Liu, J. W. S.:
Real-Time Systems.
Prentice-Hall, Inc., 2000. –
ISBN 0–13–099651–3

Anhang

Verfahrensweisen: Fallstudien

UNIX klasssich

UNIX klasssich

- zweistufig, Anwortzeiten minimierend, Interaktivität fördernd

UNIX klasssich

- zweistufig, Anwortzeiten minimierend, Interaktivität fördernd:

  low-level kurzfristig; präemptiv, MLFQ, **dynamische Prioritäten**

  - einmal pro Sekunde: $prio = cpu\_usage + p\_nice + base$
  - CPU-Nutzungsrecht mit jedem „Tick“ ($1/10\,s$) verringert
    - Prioritätswert kontinuierlich um „Tickstand“ erhöhen
    - je höher der Wert, desto niedriger die Priorität
  - über die Zeit gedämpftes CPU-Nutzungsmaß: *cpu_usage*
    - der Dämpfungsfilter variiert von UNIX zu UNIX

UNIX klasssich

- zweistufig, Anwortzeiten minimierend, Interaktivität fördernd:

high-level mittelfristig; mit **Umlagerung** (*swapping*) arbeitend

UNIX klasssich

- zweistufig, Anwortzeiten minimierend, Interaktivität fördernd:

  low-level kurzfristig; präemptiv, MLFQ, **dynamische Prioritäten**
  - einmal pro Sekunde: $prio = cpu\_usage + p\_nice + base$
  - CPU-Nutzungsrecht mit jedem „Tick“ ($1/10\,s$) verringert
    - Prioritätswert kontinuierlich um „Tickstand“ erhöhen
    - je höher der Wert, desto niedriger die Priorität
  - über die Zeit gedämpftes CPU-Nutzungsmaß: *cpu_usage*
    - der Dämpfungsfilter variiert von UNIX zu UNIX

  high-level mittelfristig; mit **Umlagerung** (*swapping*) arbeitend

- Prozesse können relativ zügig den Betriebssystemkern verlassen
  - gesteuert über die beim Schlafenlegen einstellbare **Aufweckpriorität**

UNIX 4.3 BSD

- MLFQ (32 Warteschlangen, RR), dynamische Prioritäten (0–127)

UNIX 4.3 BSD

- MLFQ (32 Warteschlangen, RR), dynamische Prioritäten (0–127):

  Berechnung der **Benutzerpriorität** bei jedem vierten Tick (40 ms)

  - $p\_usrpri = PUSER + \left[\frac{p\_cpu}{4}\right] + 2 \cdot p\_nice$
    - mit $p\_cpu = p\_cpu + 1$ bei jedem Tick (10 ms)
    - **Gewichtungsfaktor** $-20 \leq p\_nice \leq 20$ (vgl. `nice(2)`)
  - Prozess mit Priorität P kommt in Warteschlange $P/4$

UNIX 4.3 BSD

- MLFQ (32 Warteschlangen, RR), dynamische Prioritäten (0–127):

Berechnung der **Benutzerpriorität** bei jedem vierten Tick (40 ms)

- $p\_usrpri = PUSER + \left[\frac{p\_cpu}{4}\right] + 2 \cdot p\_nice$
  - mit $p\_cpu = p\_cpu + 1$ bei jedem Tick (10 ms)
  - **Gewichtungsfaktor** $-20 \leq p\_nice \leq 20$ (vgl. `nice(2)`)
- Prozess mit Priorität P kommt in Warteschlange $P/4$

Glättung des Wertes der **Prozessornutzung** (p_cpu), sekündlich

- $p\_cpu = \frac{2 \cdot load}{2 \cdot load + 1} \cdot p\_cpu + p\_nice$
- Sonderfall: Prozesse schliefen länger als eine Sekunde
  - $p\_cpu = \left[\frac{2 \cdot load}{2 \cdot load + 1}\right]^{p\_slptime} \cdot p\_cpu$

- Annahme 1:
  - mittlere Auslastung (*load*) sei 1

$$\frac{2 \cdot load}{2 \cdot load + 1} = \frac{2}{3} = 0.66$$

- Annahme 1:
  - mittlere Auslastung (*load*) sei 1

$$\frac{2 \cdot load}{2 \cdot load + 1} = \frac{2}{3} = 0.66 \rightsquigarrow p\_cpu = 0.66 \cdot p\_cpu + p\_nice$$

- Annahme 1:
  - mittlere Auslastung (*load*) sei 1

$$\frac{2 \cdot load}{2 \cdot load + 1} = \frac{2}{3} = 0.66 \rightsquigarrow p\_cpu = 0.66 \cdot p\_cpu + p\_nice$$

- Annahme 2:
  - Prozess sammelt T_i Ticks im Zeitinterval i an, $p\_nice = 0$:

$$\begin{aligned} p\_cpu &= 0.66 \cdot T_0 \\ &= 0.66 \cdot (T_1 + 0.66 \cdot T_0) = 0.66 \cdot T_1 + 0.44 \cdot T_0 \\ &= 0.66 \cdot T_2 + 0.44 \cdot T_1 + 0.30 \cdot T_0 \\ &= 0.66 \cdot T_3 + \cdots + 0.20 \cdot T_0 \\ &= 0.66 \cdot T_4 + \cdots + 0.13 \cdot T_0 \end{aligned}$$

- Annahme 1:
  - mittlere Auslastung (*load*) sei 1

$$\frac{2 \cdot load}{2 \cdot load + 1} = \frac{2}{3} = 0.66 \rightsquigarrow p\_cpu = 0.66 \cdot p\_cpu + p\_nice$$

- Annahme 2:
  - Prozess sammelt T_i Ticks im Zeitinterval i an, $p\_nice = 0$:

$$\begin{aligned} p\_cpu &= 0.66 \cdot T_0 \\ &= 0.66 \cdot (T_1 + 0.66 \cdot T_0) = 0.66 \cdot T_1 + 0.44 \cdot T_0 \\ &= 0.66 \cdot T_2 + 0.44 \cdot T_1 + 0.30 \cdot T_0 \\ &= 0.66 \cdot T_3 + \cdots + 0.20 \cdot T_0 \\ &= 0.66 \cdot T_4 + \cdots + 0.13 \cdot T_0 \end{aligned}$$

  - nach fünf Sekunden gehen nur noch etwa 13 % der „Altlast“ ein

UNIX Solaris

- MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

UNIX Solaris

- MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

quantum	*tqexp*	*slpret*	*maxwait*	*lwait*	Ebene
200	0	50	0	50	0
200	0	50	0	50	1
		. . .			
40	34	55	0	55	44
40	35	56	0	56	45
40	36	57	0	57	46
40	37	58	0	58	47
40	38	58	0	58	48
40	39	58	0	59	49
40	40	58	0	59	50
40	41	58	0	59	51
40	42	58	0	59	52
40	43	58	0	59	53
40	44	58	0	59	54
40	45	58	0	59	55
40	46	58	0	59	56
40	47	58	0	59	57
40	48	58	0	59	58
20	49	59	32000	59	59

`/usr/sbin/dispadmin -c TS -g`

UNIX Solaris

- MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

quantum	*tqexp*	*slpret*	*maxwait*	*lwait*	Ebene
200	0	50	0	50	0
200	0	50	0	50	1
		. . .			
40	34	55	0	55	44
40	35	56	0	56	45
40	36	57	0	57	46
40	37	58	0	58	47
40	38	58	0	58	48
40	39	58	0	59	49
40	40	58	0	59	50
40	41	58	0	59	51
40	42	58	0	59	52
40	43	58	0	59	53
40	44	58	0	59	54
40	45	58	0	59	55
40	46	58	0	59	56
40	47	58	0	59	57
40	48	58	0	59	58
20	49	59	32000	59	59

`/usr/sbin/dispadmin -c TS -g`

MLQ (Klasse)		Priorität
time-sharing	TS	0–59
interactive	IA	0–59
system	SYS	60–99
real time	RT	100–109

UNIX Solaris

- MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

quantum	*tqexp*	*slpret*	*maxwait*	*lwait*	Ebene
200	0	50	0	50	0
200	0	50	0	50	1
		. . .			
40	34	55	0	55	44
40	35	56	0	56	45
40	36	57	0	57	46
40	37	58	0	58	47
40	38	58	0	58	48
40	39	58	0	59	49
40	40	58	0	59	50
40	41	58	0	59	51
40	42	58	0	59	52
40	43	58	0	59	53
40	44	58	0	59	54
40	45	58	0	59	55
40	46	58	0	59	56
40	47	58	0	59	57
40	48	58	0	59	58
20	49	59	32000	59	59

`/usr/sbin/dispadmin -c TS -g`

MLQ (Klasse)		Priorität
time-sharing	TS	0–59
interactive	IA	0–59
system	SYS	60–99
real time	RT	100–109

MLFQ in Klasse TS bzw. IA:

quantum Zeitscheibe (ms)

tqexp Ebene bei Bestrafung

slprt Ebene nach Deblockierung

maxwait ohne Bedienung (s)

lwait Ebene bei Bewährung

UNIX Solaris

- MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

quantum	*tqexp*	*slpret*	*maxwait*	*lwait*	Ebene
200	0	50	0	50	0
200	0	50	0	50	1
		...			
40	34	55	0	55	44
40	35	56	0	56	45
40	36	57	0	57	46
40	37	58	0	58	47
40	38	58	0	58	48
40	39	58	0	59	49
40	40	58	0	59	50
40	41	58	0	59	51
40	42	58	0	59	52
40	43	58	0	59	53
40	44	58	0	59	54
40	45	58	0	59	55
40	46	58	0	59	56
40	47	58	0	59	57
40	48	58	0	59	58
20	49	59	32000	59	59

`/usr/sbin/dispadmin -c TS -g`

MLQ (Klasse)		Priorität
time-sharing	TS	0–59
interactive	IA	0–59
system	SYS	60–99
real time	RT	100–109

MLFQ in Klasse TS bzw. IA:

- ***quantum*** Zeitscheibe (ms)
- ***tqexp*** Ebene bei Bestrafung
- ***slprt*** Ebene nach Deblockierung
- ***maxwait*** ohne Bedienung (s)
- ***lwait*** Ebene bei Bewährung

- Besonderheit: *dispatch table* (TS, IA) kapselt alle Entscheidungen
  - kunden-/problemspezifische Lösungen durch verschiedene Tabellen

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß
9	50	1	E/A-Stoß

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß
9	50	1	E/A-Stoß
10	58	1	E/A-Stoß

Beispiel:

- 1 × CPU-Stoß à 1000 ms
- 5 × E/A-Stoß → CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß
9	50	1	E/A-Stoß
10	58	1	E/A-Stoß
11	58	1	E/A-Stoß

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß
9	50	1	E/A-Stoß
10	58	1	E/A-Stoß
11	58	1	E/A-Stoß
12	58	1	E/A-Stoß

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

	…		
7	0	20	*anti-aging*

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

…			
7	0	20	*anti-aging*
8	50	40	Zeitscheibe

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

…			
7	0	20	*anti-aging*
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

		…	
7	0	20	*anti-aging*
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe
10	30	80	Zeitscheibe

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

		…	
7	0	20	*anti-aging*
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe
10	30	80	Zeitscheibe
11	20	120	Zeitscheibe

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

	…		
7	0	20	*anti-aging*
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe
10	30	80	Zeitscheibe
11	20	120	Zeitscheibe
12	10	80	E/A-Stoß

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- $1 \times$ CPU-Stoß à 1000 ms
- $5 \times$ E/A-Stoß $\rightarrow$ CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe

Variante: nach 640 ms…

- Prozess wird verdrängt, muss auf die erneute Einlastung warten
- Alterung des wartenden Prozesses wird durch Prioritätsanhebung entgegengewirkt (*anti-aging*)
- die höhere Ebene erreicht, sinkt der Prozess danach wieder ab

…			
7	0	20	*anti-aging*
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe
10	30	80	Zeitscheibe
11	20	120	Zeitscheibe
12	10	80	E/A-Stoß
13	50	1	E/A-Stoß
…			

- Prozessen zugewiesene Prozessorzeit ist in **Epochen** unterteilt:

  beginnen alle lauffähige Prozess haben ihr Zeitquantum erhalten

  enden alle lauffähigen Prozesse haben ihr Zeitquantum verbraucht

- Prozessen zugewiesene Prozessorzeit ist in **Epochen** unterteilt:

  beginnen alle lauffähige Prozess haben ihr Zeitquantum erhalten

  enden alle lauffähigen Prozesse haben ihr Zeitquantum verbraucht

- **Zeitquanten** (Zeitscheiben) variieren mit Prozessen und Epochen:
  - jeder Prozess besitzt eine einstellbare Zeitquantumbasis (`nice(2)`)
    - 20 Ticks $\approx$ 210 ms
    - das Zeitquantum eines Prozesses nimmt periodisch (Tick) ab
  - beide Werte addiert liefert die dynamische Priorität eines Prozesses
    - dynamische Anpassung: $\mathit{quantum} = \mathit{quantum}/2 + (20 - \mathit{nice})/4 + 1$

- **Zeitquanten** (Zeitscheiben) variieren mit Prozessen und Epochen:
  - jeder Prozess besitzt eine einstellbare Zeitquantumbasis (`nice(2)`)
    - 20 Ticks $\approx$ 210 ms
    - das Zeitquantum eines Prozesses nimmt periodisch (Tick) ab
  - beide Werte addiert liefert die dynamische Priorität eines Prozesses
    - dynamische Anpassung: $\mathit{quantum} = \mathit{quantum}/2 + (20 - \mathit{nice})/4 + 1$

- **Echtzeitprozessse** besitzen statische Prioritäten: 0–99
  - je kleiner der Wert, desto höher die Priorität
  - schwache Echtzeit (vgl. [6, S. 16])

- Prozesseinplanung unterscheidet zwischen drei ***Scheduling*-Klassen**:

FIFO	verdrängbare, kooperative Echtzeitprozesse	} eine
RR	Echtzeitprozesse derselben Priorität	
other	konventionelle („*time-shared*“) Prozesse	

Bereitliste

- Prozesseinplanung unterscheidet zwischen drei ***Scheduling*-Klassen**:

FIFO	verdrängbare, kooperative Echtzeitprozesse	} eine
RR	Echtzeitprozesse derselben Priorität	
other	konventionelle („*time-shared*“) Prozesse	

Bereitliste

- Prozessauswahl greift auf eine **Gütefunktion** zurück: $O(n)$

$v = -1000$	der Prozess ist *Init*	$-$
$v = 0$	der Prozess hat sein Zeitquantum verbraucht	$-$
$0 < v < 1000$	der Prozess hat sein Zeitquantum nicht verbraucht	$+$
$v \geq 1000$	der Prozess ist ein Echtzeitprozess	$++$

- Prozessauswahl greift auf eine **Gütefunktion** zurück: $O(n)$

$v = -1000$	der Prozess ist *Init*	$-$
$v = 0$	der Prozess hat sein Zeitquantum verbraucht	$-$
$0 < v < 1000$	der Prozess hat sein Zeitquantum nicht verbraucht	$+$
$v \geq 1000$	der Prozess ist ein Echtzeitprozess	$++$

- Prozesse können bei der Auswahl einen **Bonus** („*boost*“) erhalten
  - sofern sie sich mit dem Vorgänger den Adressraum teilen

- Prozessplanung hat **konstante Berechnungskomplexität** [5]

- Prozessplanung hat **konstante Berechnungskomplexität** [5]:
  Prioritätsfelder zwei Tabellen pro CPU: *active*, *expired*
  - jedes Feld eine Bitkarte (*bitmap*) von n Einträgen
  - mit n gleich der Anzahl von Prioritätsebenen

- Prozessplanung hat **konstante Berechnungskomplexität** [5]:

  Prioritätsfelder zwei Tabellen pro CPU: *active*, *expired*
  - jedes Feld eine Bitkarte (*bitmap*) von n Einträgen
  - mit n gleich der Anzahl von Prioritätsebenen

  Prioritätsebenen 140 Ebenen $=$ Einträge pro Tabelle
  - 0–99 für Echtzeit-, 100–139 für sonstige Prozesse
  - pro Ebene eine (doppelt verkettete) Bereitliste

- Prozessplanung hat **konstante Berechnungskomplexität** [5]:
  Prioritätsfelder zwei Tabellen pro CPU: *active*, *expired*
  - jedes Feld eine Bitkarte (*bitmap*) von n Einträgen
  - mit n gleich der Anzahl von Prioritätsebenen

  Prioritätsebenen 140 Ebenen $=$ Einträge pro Tabelle
  - 0–99 für Echtzeit-, 100–139 für sonstige Prozesse
  - pro Ebene eine (doppelt verkettete) Bereitliste
- ist Bitkartenposition i gesetzt (1, *true*), dann ist wenigstens ein Prozess auf der Bereitliste von Ebene i verzeichnet
- zur Listenauswahl wird die Bitkarte von Anfang ($i = 0$) an abgesucht
  - ggf. unter Zuhilfenahme spezieller Bitoperationen des Prozessors (x86: BSF)

- Prozessplanung hat **konstante Berechnungskomplexität** [5]:

Prioritätsebenen 140 Ebenen

- 0–99 für Echtzeit-, 100–139 für sonstige Prozesse
- pro Ebene eine (doppelt verkettete) Bereitliste

- ist Bitkartenposition i gesetzt (1, *true*), dann ist wenigstens ein Prozess auf der Bereitliste von Ebene i verzeichnet
- zur Listenauswahl wird die Bitkarte von Anfang ($i = 0$) an abgesucht
  - ggf. unter Zuhilfenahme spezieller Bitoperationen des Prozessors (x86: BSF)

- Prioritäten gemeiner Prozesse skalieren je nach Interaktivitätsgrad
  - **Bonus** (-5) für interaktive Prozesse, **Strafe** ($+5$) für rechenintensive
  - berechnet am Zeitscheibenende: $prio = MAX\_RT\_PRIO + nice + 20$

- Prozessplanung hat **konstante Berechnungskomplexität** [5]:
  Prioritätsfelder zwei Tabellen pro CPU: *active*, *expired*

  - ist Bitkartenposition i gesetzt (1, *true*), dann ist wenigstens ein Prozess auf der Bereitliste von Ebene i verzeichnet
  - zur Listenauswahl wird die Bitkarte von Anfang ($i = 0$) an abgesucht
    - ggf. unter Zuhilfenahme spezieller Bitoperationen des Prozessors (x86: BSF)

- Ablauf des Zeitquantums befördert aktiven Prozess ins „*expired*“-Feld
  - zum Epochenwechsel werden die Tabellen ausgetauscht: Zeigerwechsel

- der Planer verzichtet auf Prozessheuristik und feste Zeitscheiben

- der Planer verzichtet auf Prozessheuristik und feste Zeitscheiben
  - vielmehr garantiert er jedem Prozess einen bestimmten **Prozessoranteil**
    - $1/N$ Zeiteinheiten, mit N gleich der Anzahl der laufbereiten Prozesse
    - der Gesamtanteil wächst und schrumpft mit der Gesamtzahl dieser Prozesse
    - der relative Anteil variiert mit der Priorität (`nice(2)`) eines Prozesses

- der Planer verzichtet auf Prozessheuristik und feste Zeitscheiben
  - vielmehr garantiert er jedem Prozess einen bestimmten **Prozessoranteil**
    - $1/N$ Zeiteinheiten, mit N gleich der Anzahl der laufbereiten Prozesse
    - der Gesamtanteil wächst und schrumpft mit der Gesamtzahl dieser Prozesse
    - der relative Anteil variiert mit der Priorität (`nice(2)`) eines Prozesses
  - in Bezug auf diesen Anteil gilt dabei für einen laufenden Prozess:
    (a) er kann den Prozessor früher abgeben (blockiert: E/A-intensiver Prozess)
    (b) er darf den Prozessor länger behalten (rechenintensiver Prozess)
    (c) er wird verdrängt, sobald er sein $1/N$-tel Zeitanteil konsumiert hat und ein anderer Prozess wartet, er länger als der kürzeste laufbereite Prozess läuft

- der Planer verzichtet auf Prozessheuristik und feste Zeitscheiben
  - vielmehr garantiert er jedem Prozess einen bestimmten **Prozessoranteil**
    - $1/N$ Zeiteinheiten, mit N gleich der Anzahl der laufbereiten Prozesse
    - der Gesamtanteil wächst und schrumpft mit der Gesamtzahl dieser Prozesse
    - der relative Anteil variiert mit der Priorität (`nice(2)`) eines Prozesses
  - in Bezug auf diesen Anteil gilt dabei für einen laufenden Prozess:
    (a) er kann den Prozessor früher abgeben (blockiert: E/A-intensiver Prozess)
    (b) er darf den Prozessor länger behalten (rechenintensiver Prozess)
    (c) er wird verdrängt, sobald er sein $1/N$-tel Zeitanteil konsumiert hat und ein anderer Prozess wartet, er länger als der kürzeste laufbereite Prozess läuft
  - Fall (c) wird regelmäßig überprüft, durch einen Zeitgeber (*clock tick*)

- der Planer verzichtet auf Prozessheuristik und feste Zeitscheiben
  - vielmehr garantiert er jedem Prozess einen bestimmten **Prozessoranteil**
    - $1/N$ Zeiteinheiten, mit N gleich der Anzahl der laufbereiten Prozesse
    - der Gesamtanteil wächst und schrumpft mit der Gesamtzahl dieser Prozesse
    - der relative Anteil variiert mit der Priorität (`nice(2)`) eines Prozesses

  - Fall (c) wird regelmäßig überprüft, durch einen Zeitgeber (*clock tick*)
- Rechenstoßlängen der Prozesse haben eine **minimale Granularität**

Definition (*minimum granularity*, *MG*)

Die Zeitspanne, die einem Prozess auf dem Prozessor zuzubilligen ist, bevor dem Prozess der Prozessor wieder entzogen werden kann.

  - 1 – 4 ms, die **Periodenlänge** des Zeitgebers; Vorgabe 4 ms

- die **Ziellatenz** des Planers bestimmt den effektiven Prozessoranteil

Definition (*target latency*, *TL*)

Die Mindestzeit, die für jeden laufbereiten Prozess zur Verfügung zu stellen ist, um wenigstens eine Runde des Prozessors zu erhalten.

- die **Ziellatenz** des Planers bestimmt den effektiven Prozessoranteil

Definition (*target latency*, TL)

Die Mindestzeit, die für jeden laufbereiten Prozess zur Verfügung zu stellen ist, um wenigstens eine Runde des Prozessors zu erhalten.

- auch die untere Grenze der **Periodenlänge** des Planers; Vorgabe 20 ms
- $P_{sched} = \begin{cases} TL & \text{wenn } N \leq TL/MG \\ N \times MG & \text{sonst} \end{cases}$

- wartende Prozesse geben den Beginn ihres nächsten Zeitschlitzes vor

Definition (*virtual runtime*, *VR*)

Die aus den jeweiligen Rechenstoßlängen akkumulierte Laufzeit eines Prozesses in *ns*, gewichtet mit seiner „Nettigkeit“ (*niceness*).

- wartende Prozesse geben den Beginn ihres nächsten Zeitschlitzes vor

Definition (*virtual runtime*, *VR*)

Die aus den jeweiligen Rechenstoßlängen akkumulierte Laufzeit eines Prozesses in *ns*, gewichtet mit seiner „Nettigkeit“ (*niceness*).

- wird in jeder Zeitgeberperiode oder bei der Prozessorabgabe aktualisiert
- bestimmt den Moment des Prozessorentzugs und den Bereitlistenplatz
  - der laufende Prozess wird verdrängt, wenn $VR_{running} > VR_{ready}$

- Prozesse vom Prozessor zu verdrängen ist **Mangelwirtschaft** [11] ☹

- Prozesse vom Prozessor zu verdrängen ist **Mangelwirtschaft** [11] ☹
  - es besteht ein Mangel an „Waren“ (Prozessoren)
  - während genug „Geld“ (Prozesse) zum Kauf dieser Waren vorhanden ist

- Prozesse vom Prozessor zu verdrängen ist **Mangelwirtschaft** [11] ☹
  - es besteht ein Mangel an „Waren“ (Prozessoren)
  - während genug „Geld“ (Prozesse) zum Kauf dieser Waren vorhanden ist

Definition (idealer Prozessor)

Ein Prozessor, der jeden Prozess mit genau gleicher Geschwindigkeit parallel ausführen kann, jeweils mit $1/N$-tel Zeiteinheiten (*ns*).

- Prozesse vom Prozessor zu verdrängen ist **Mangelwirtschaft** [11] ☹
  - es besteht ein Mangel an „Waren“ (Prozessoren)
  - während genug „Geld“ (Prozesse) zum Kauf dieser Waren vorhanden ist

Definition (idealer Prozessor)

Ein Prozessor, der jeden Prozess mit genau gleicher Geschwindigkeit parallel ausführen kann, jeweils mit $1/N$-tel Zeiteinheiten (*ns*).

- alle Prozesse erhalten den Prozessor für die gleiche relative Laufzeit

Definition (*maximum execution time*, *MET*)

Die Zeit, die ein Prozess auf einen idealen Prozessor erwarten würde.

- Prozesse vom Prozessor zu verdrängen ist **Mangelwirtschaft** [11] ☹
  - es besteht ein Mangel an „Waren“ (Prozessoren)
  - während genug „Geld“ (Prozesse) zum Kauf dieser Waren vorhanden ist

Definition (idealer Prozessor)

Ein Prozessor, der jeden Prozess mit genau gleicher Geschwindigkeit parallel ausführen kann, jeweils mit $1/N$-tel Zeiteinheiten (ns).

- alle Prozesse erhalten den Prozessor für die gleiche relative Laufzeit

Definition (*maximum execution time*, MET)

Die Zeit, die ein Prozess auf einen idealen Prozessor erwarten würde.

  - für den nächsten Prozess auf der Bereitliste ist dies die Wartezeit auf den Prozessor, normalisiert auf die Gesamtzahl der lauffähigen Prozesse
  - nach $MET = P_{sched}/N$ Zeiteinheiten wird der Prozess mit der kürzesten VR den Prozessor zugeteilt erhalten

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$
  - n gleicht der Anzahl der Knoten im Baum, d.h., der laufbereiten Prozesse

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$
  - n gleicht der Anzahl der Knoten im Baum, d.h., der laufbereiten Prozesse
  - sortiert nach der „**Vorkaufszeit**“ eines Prozesses für den Prozessor in *ns*

Definition (*preemption time*)

Der Zeitpunkt eines (auf der Bereitliste) wartenden Prozesses zur Ausübung des Vorkaufsrechts, den Prozessor als erster angeboten zu bekommen.

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$
  - n gleicht der Anzahl der Knoten im Baum, d.h., der laufbereiten Prozesse
  - sortiert nach der „**Vorkaufszeit**“ eines Prozesses für den Prozessor in *ns*

Definition (*preemption time*)

Der Zeitpunkt eines (auf der Bereitliste) wartenden Prozesses zur Ausübung des Vorkaufsrechts, den Prozessor als erster angeboten zu bekommen.

  - die **gewichtete virtuelle Laufzeit** eines vormals gelaufenen Prozesses
  - diese variiert von Prozess zu Prozess auf der Bereitliste
  - ein zunehmender Wert von links nach rechts innerhalb des RS-Baums

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$
  - n gleicht der Anzahl der Knoten im Baum, d.h., der laufbereiten Prozesse
  - sortiert nach der „**Vorkaufszeit**“ eines Prozesses für den Prozessor in *ns*

Definition (*preemption time*)

Der Zeitpunkt eines (auf der Bereitliste) wartenden Prozesses zur Ausübung des Vorkaufsrechts, den Prozessor als erster angeboten zu bekommen.

    - die **gewichtete virtuelle Laufzeit** eines vormals gelaufenen Prozesses
    - diese variiert von Prozess zu Prozess auf der Bereitliste
    - ein zunehmender Wert von links nach rechts innerhalb des RS-Baums
- Prozesse sind nach dem Beginn ihres nächsten Rechenstoßes gelistet
  - frühestens am Ende der laufenden Zeitgeberperiode wird verdrängt
    - erhält der ganz links im RS-Baum verzeichnete Prozess den Prozessor und
    - der verdrängte Prozess wird gemäß seiner *VR* im RS-Baum eingetragen

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])

- die Bereitliste ist ein **balancierter binärer Suchbaum** [1]:[3] $O(\log n)$
  - n gleicht der Anzahl der Knoten im Baum, d.h., der laufbereiten Prozesse
  - sortiert nach der „**Vorkaufszeit**“ eines Prozesses für den Prozessor in *ns*

Definition (*preemption time*)

Der Zeitpunkt eines (auf der Bereitliste) wartenden Prozesses zur Ausübung des Vorkaufsrechts, den Prozessor als erster angeboten zu bekommen.

    - die **gewichtete virtuelle Laufzeit** eines vormals gelaufenen Prozesses
    - diese variiert von Prozess zu Prozess auf der Bereitliste
    - ein zunehmender Wert von links nach rechts innerhalb des RS-Baums
- Prozesse sind nach dem Beginn ihres nächsten Rechenstoßes gelistet
  - frühestens am Ende der laufenden Zeitgeberperiode wird verdrängt
    - erhält der ganz links im RS-Baum verzeichnete Prozess den Prozessor und
    - der verdrängte Prozess wird gemäß seiner *VR* im RS-Baum eingetragen
  - ein erzeugter Prozess erhält die **minimale aktuelle virtuelle Laufzeit**
    - die kleinste vom System festgestellte virtuelle Laufzeit eines Prozesses

[3] Rot-Schwarz-Baum (RS-Baum: *red-black tree* [4])