

# Systeme I: Betriebssysteme

## **Kapitel 7** **Scheduling**

Wolfram Burgard



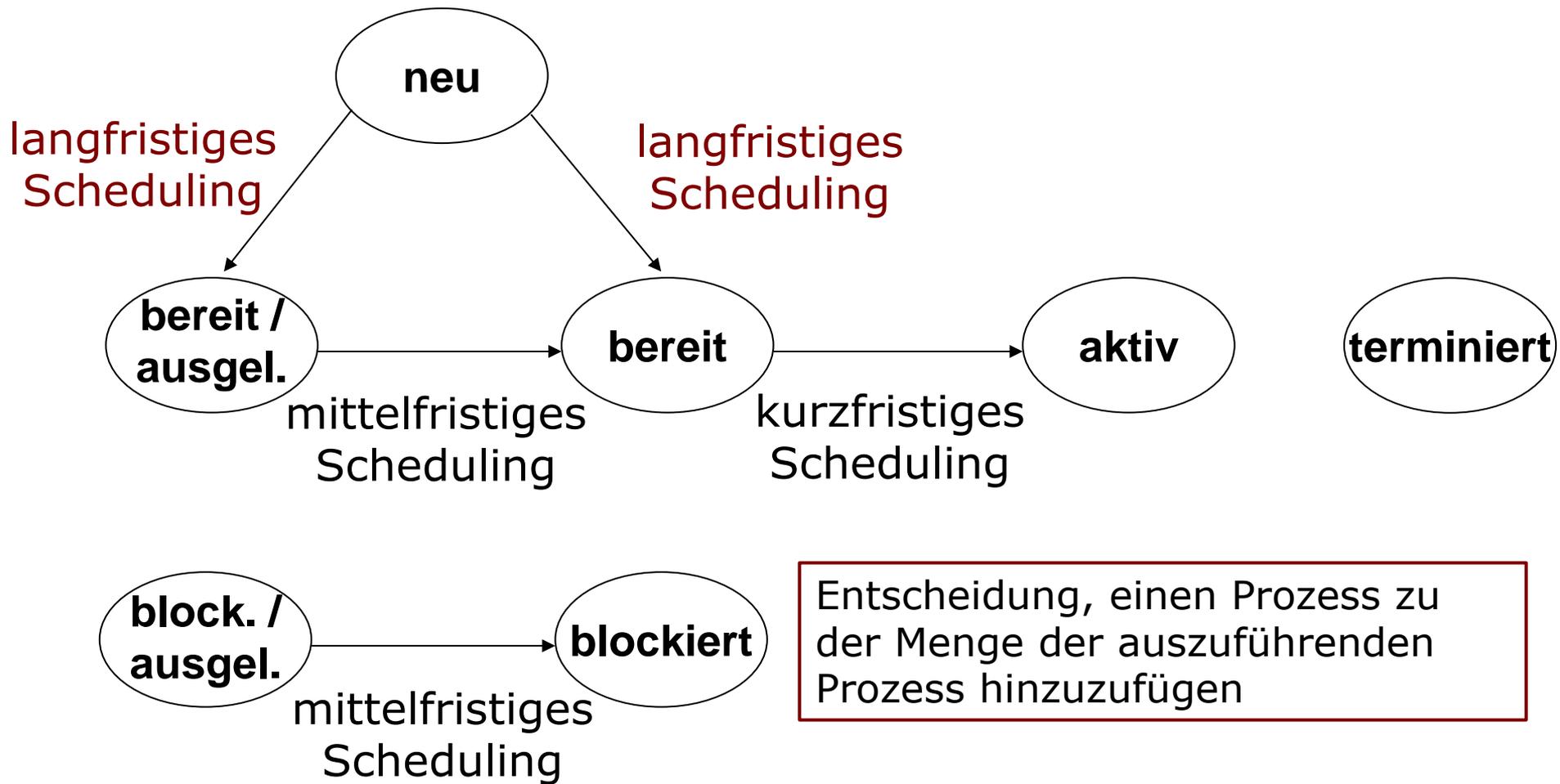
# Inhalt Vorlesung

- Aufbau einfacher Rechner
- Überblick: Aufgabe, Historische Entwicklung, unterschiedliche Arten von Betriebssystemen
- Verschiedene Komponenten / Konzepte von Betriebssystemen
  - Dateisysteme
  - Prozesse
  - Nebenläufigkeit und wechselseitiger Ausschluss
  - Deadlocks
  - **Scheduling**
  - Speicherverwaltung

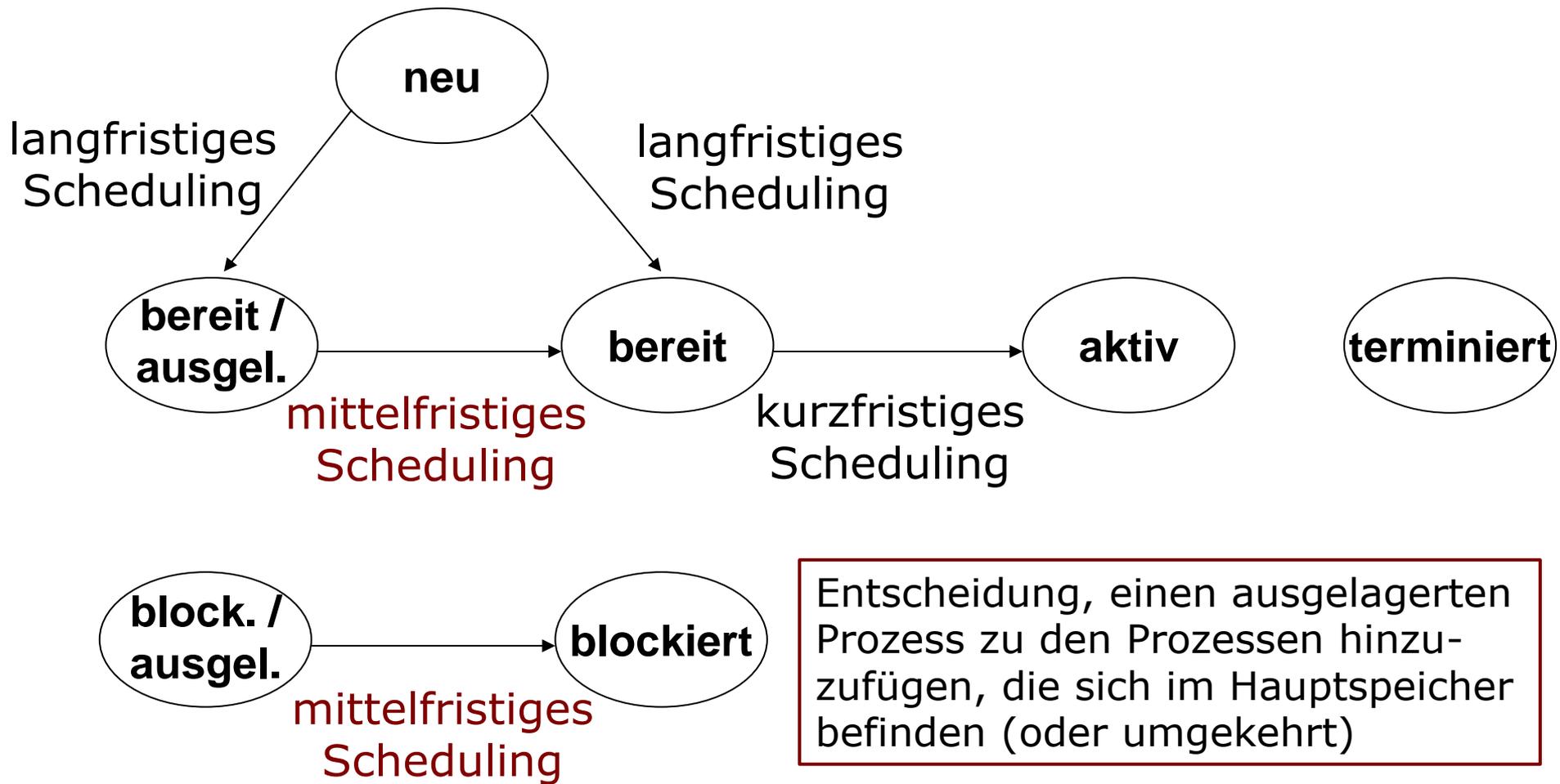
# Einführung

- Mehrprogrammsystem: Im Hauptspeicher werden mehrere Prozesse verwaltet
- Jeder Prozess wird entweder vom Prozessor bearbeitet oder wartet auf ein Ereignis
- Scheduling: Betriebssystem muss entscheiden, welche Prozesse auf den CPU-Kernen Rechenzeit beanspruchen dürfen

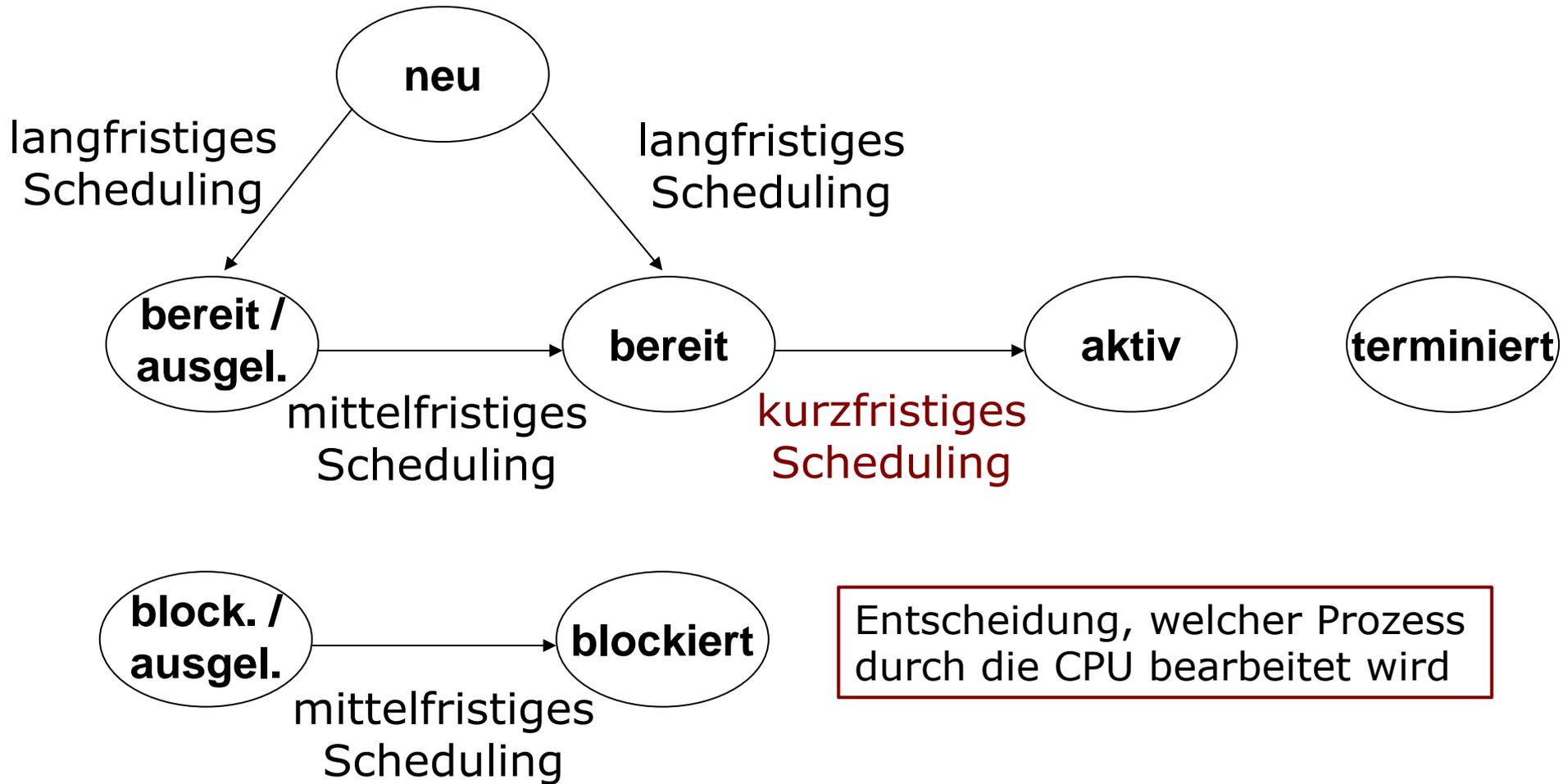
# Drei Arten von Scheduling



# Drei Arten von Scheduling



# Drei Arten von Scheduling



# Kurzfristiges Scheduling

- Rechenzeit wird Prozessen so zugewiesen, dass "optimale Performance" erreicht wird
- Verschiedene Scheduling-Algorithmen existieren für verschiedene Optimierungsziele
- Beachte: Kontextwechsel kosten Rechenzeit

# Kriterien für das kurzfristige Scheduling (1)

- **Benutzerorientiert:**
  - Minimale Antwortzeit bei interaktivem System
  - Minimale Zeit zwischen Eingang und Abschluss eines Prozesses (Durchlaufzeit)
  - Gute Vorhersagbarkeit (unabhängig von Systemauslastung ähnliche Zeit)
- **Systemorientiert:**
  - Maximale Anzahl von Prozessen, die pro Zeiteinheit abgearbeitet werden (Durchsatz, z.B. pro Stunde)
  - Maximale CPU-Auslastung (aktive Zeit)

# Durchsatz vs. Durchlaufzeit

- Durchsatz: Anzahl der Prozesse, die vom System z.B. pro Stunde erledigt werden
- Durchlaufzeit: Zeit von Start bis Abschluss
- Hoher Durchsatz heißt nicht unbedingt niedrige Durchlaufzeit
- Für Benutzer ist eher niedrige Durchlaufzeit interessant

# Kriterien für das kurzfristige Scheduling (2)

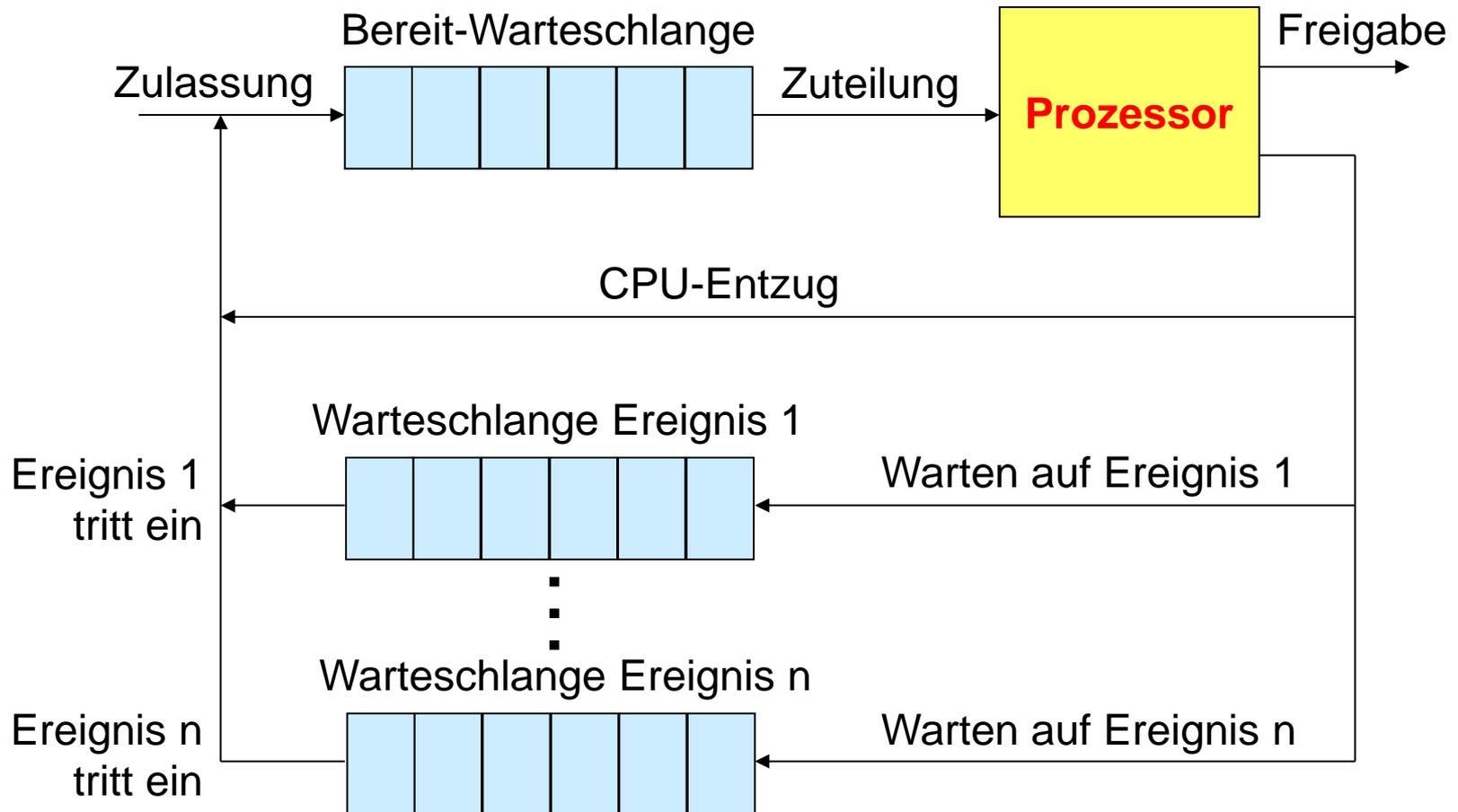
- **Allgemein:**
  - Fairness: Jeder Prozess erhält CPU irgendwann
  - Prioritäten müssen eingehalten werden
  - Effizienz: Möglichst wenig Aufwand für Scheduling selbst
- **Echtzeitsysteme:**
  - Vorhersehbares Verhalten
  - Einhalten von Deadlines

# Kriterien für das kurzfristige Scheduling (3)

- Abhängigkeiten zwischen den Kriterien
- Beispiel:
  - Kurze Antwortzeit: Viele Wechsel zwischen Prozessen
  - Aber dann: Niedrigerer Durchsatz und mehr Aufwand durch Prozesswechsel
- Scheduling-Strategie muss Kompromiss schließen

# Erinnerung: Warteschlangen

Warteschlangen für bereite Prozesse und für Prozesse, die auf Ereignisse warten

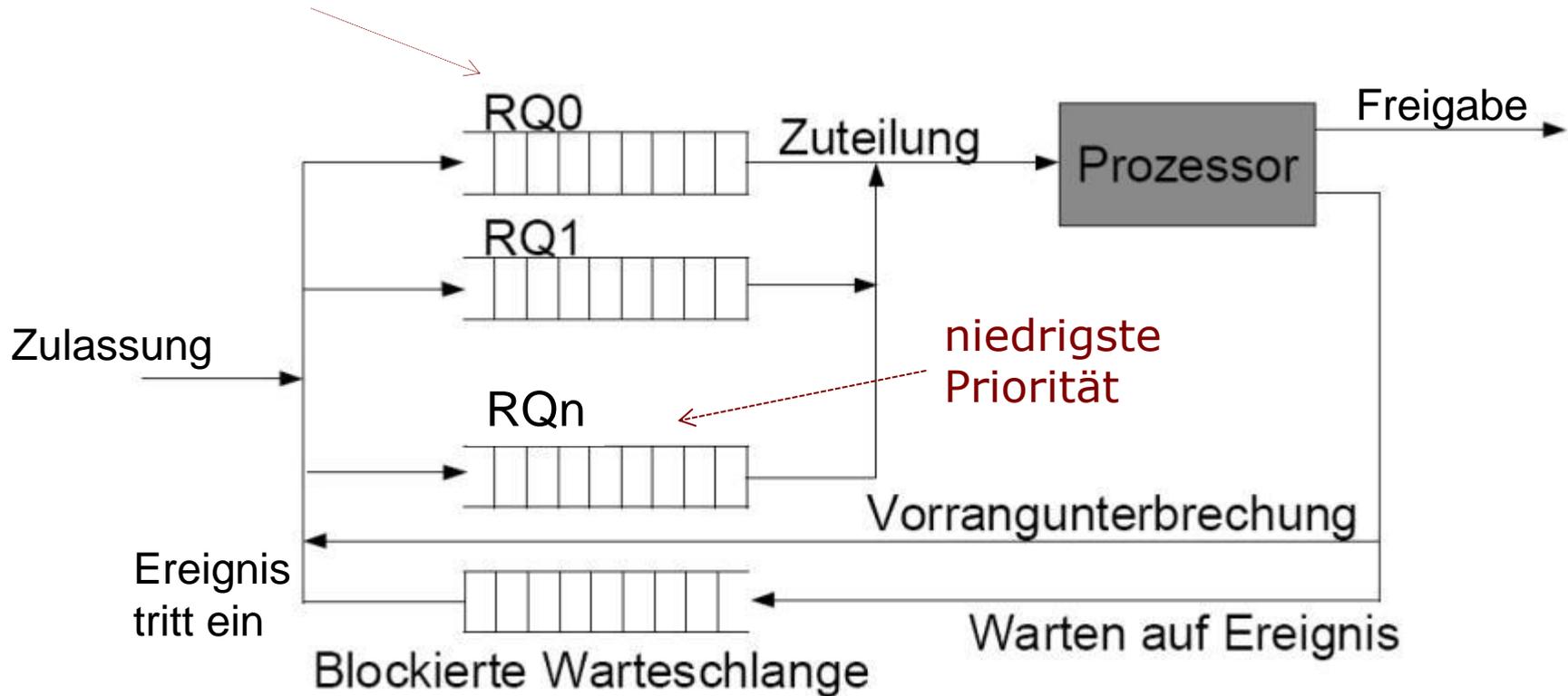


# Prioritäten (1)

- Prioritäten: Mehrere Warteschlangen mit bereiten Prozessen verschiedener Priorität
- Bei Entscheidung der Ablaufplanung: Scheduler beginnt mit der Warteschlange, die bereite Prozesse enthält und die höchste Priorität hat
- Innerhalb Warteschlange: Scheduling-Strategie

# Prioritäten (2)

höchste Priorität



# Prioritäten (3)

- Bereiter Prozess in Warteschlange mit höchster Priorität erhält Rechenzeit
- Problem: Verhungern von Prozessen mit geringer Priorität
- Lösung: Ändere Prioritäten entsprechend Alter (später mehr dazu)

# Scheduling-Algorithmen: Prozessauswahl

- Auswahlfunktion legt fest, welcher der bereiten Prozesse als nächstes aktiv wird
- Basierend auf Prioritäten oder auch Ausführungseigenschaften
- Drei Größen von Bedeutung:
  - $w$  (Wartezeit auf CPU seit Erzeugung)
  - $e$  (bisher verbrauchte CPU-Zeit)
  - $s$  (insgesamt benötigte CPU-Zeit, geschätzt)

# Scheduling-Algorithmen: Zeitpunkt der Auswahlentscheidung

- **Nicht-präemptives Scheduling:**  
CPU kann einem Prozess nur entzogen werden, wenn er beendet oder blockiert ist
- **Präemptives Scheduling:**  
Aktueller Prozess kann vom Betriebssystem unterbrochen werden, wenn dies richtig erscheint

# First Come First Served (FCFS)

- Nicht-präemptive Strategie
- **Strategie:**  
Wenn ein Prozess beendet oder blockiert ist: Bereiter Prozess, der schon am längsten wartet, wird aktiv
- Auswahlfunktion:  $\max(w)$
- Implementiert durch einfache Warteschlange

# Beispiel FCFS

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



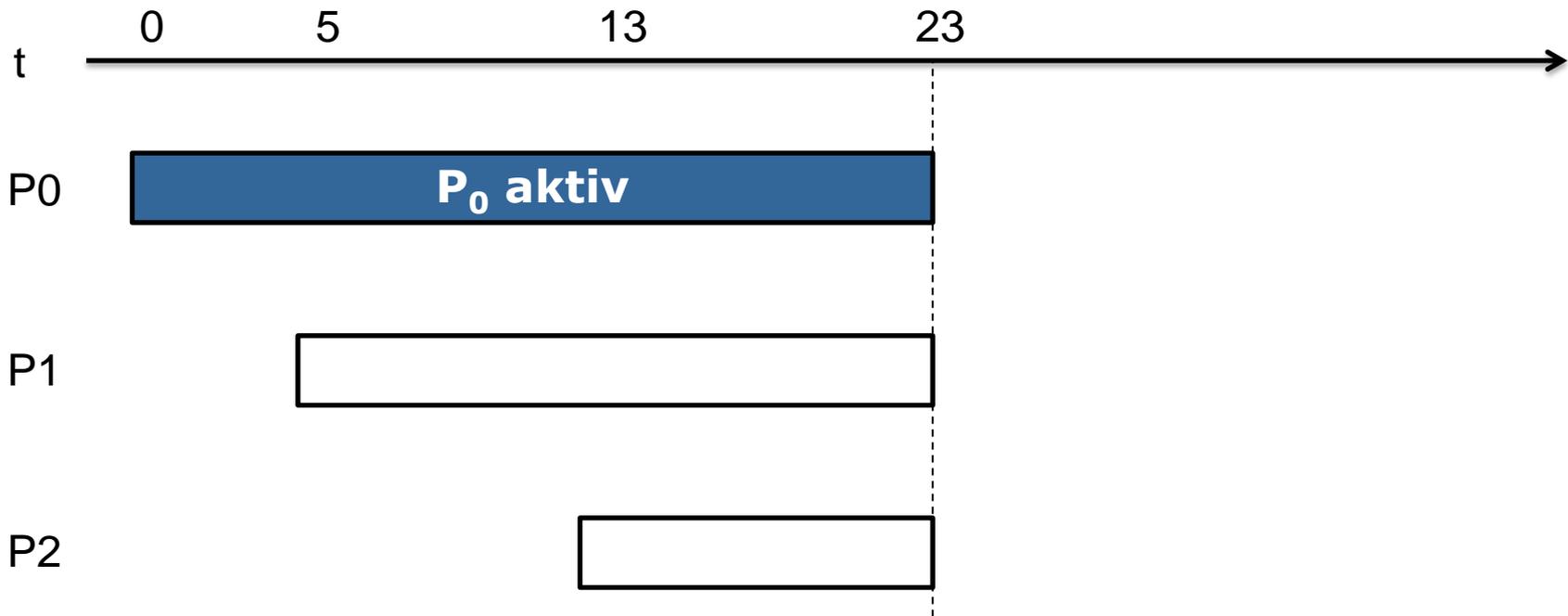
P0

P1

P2

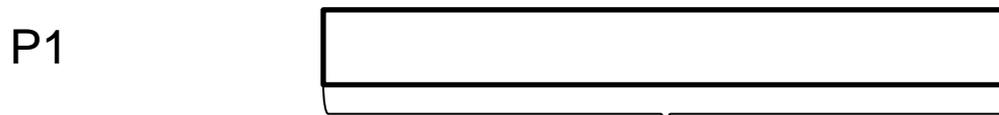
# Beispiel FCFS

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel FCFS

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



$w_1=18$

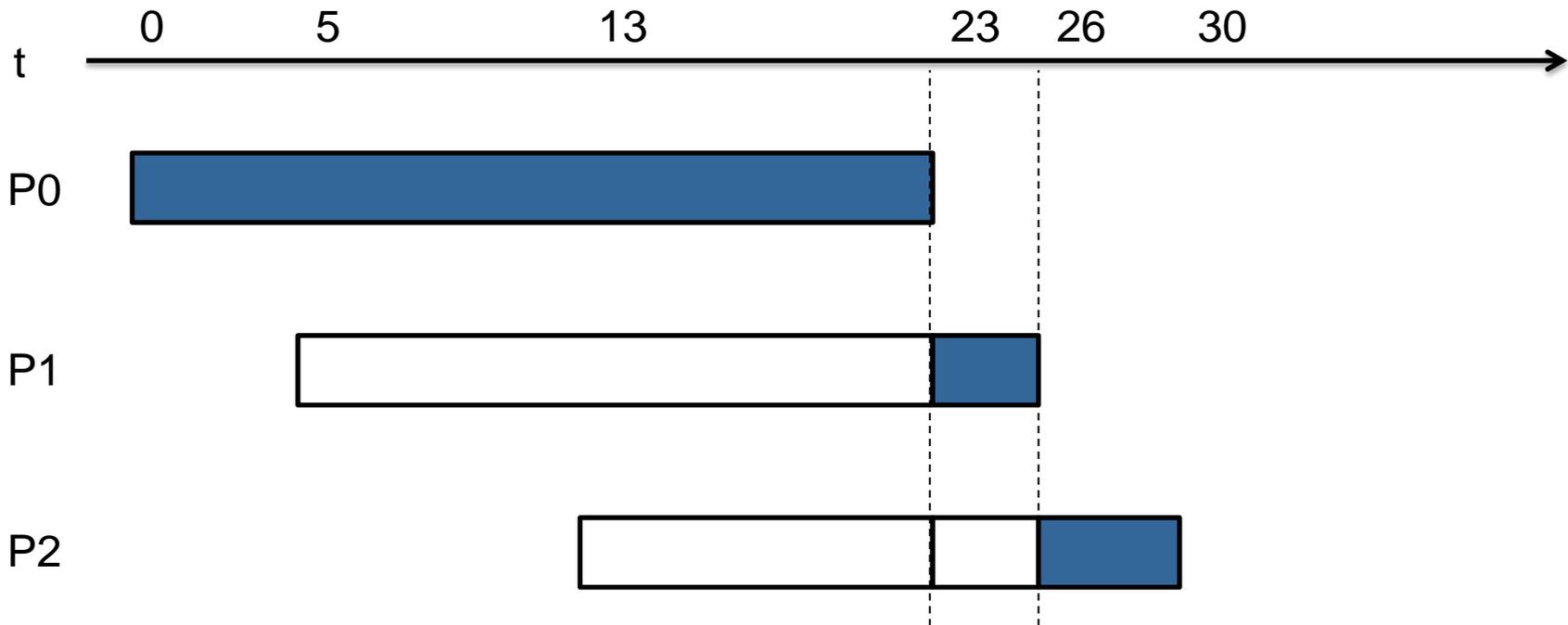


$w_2=10$

Auswahlstrategie:  
 $\max(w)$

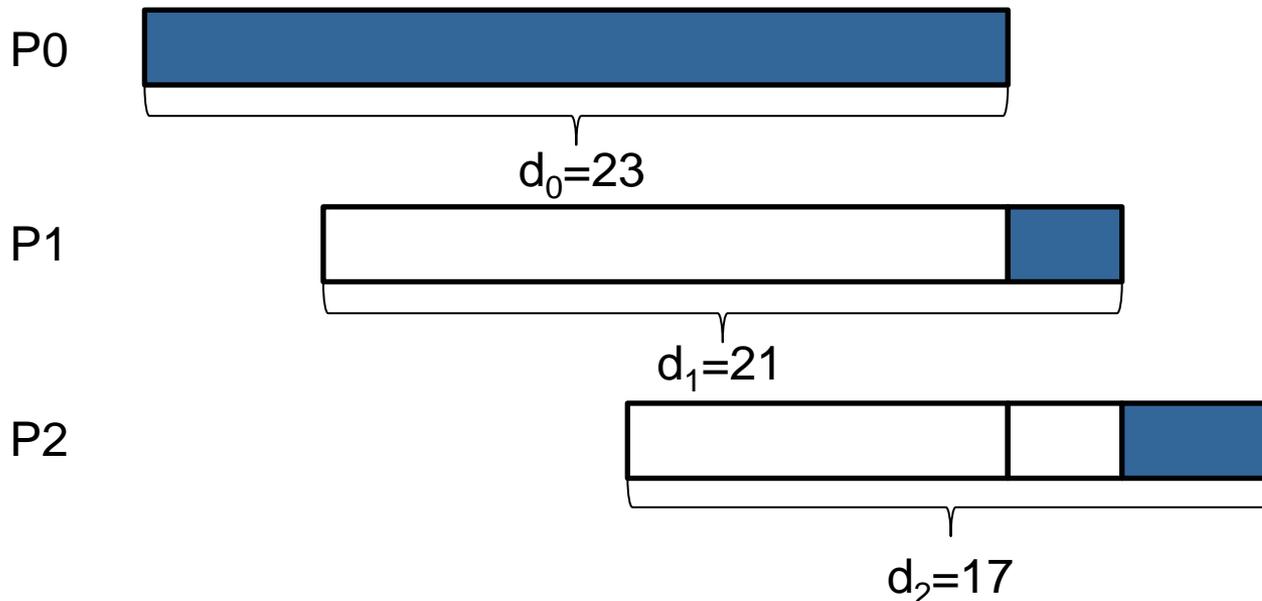
# Beispiel FCFS

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel FCFS - Durchlaufzeiten

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



Mittlere Durchlaufzeit:  
 $d^*=20,3$

# First Come First Served (FCFS)

## Analyse

- Begünstigt lange Prozesse, kurze Prozesse können durch lange Prozesse stark verzögert werden
- Begünstigt Prozesse ohne Ein-/Ausgabe (die den Prozessor vor Beendigung nicht abgeben)
- Alleine nicht sehr interessant, aber kann mit Prioritätsverfahren kombiniert werden

# Round Robin (RR)

- Präemptive Strategie
- **Strategie:**
  - Scheduler wird nach Ablauf fester Zeitdauer immer wieder aktiviert
  - Laufender Prozess wird dann in eine Warteschlange eingefügt
  - Der am längsten wartende Prozess wird aktiviert

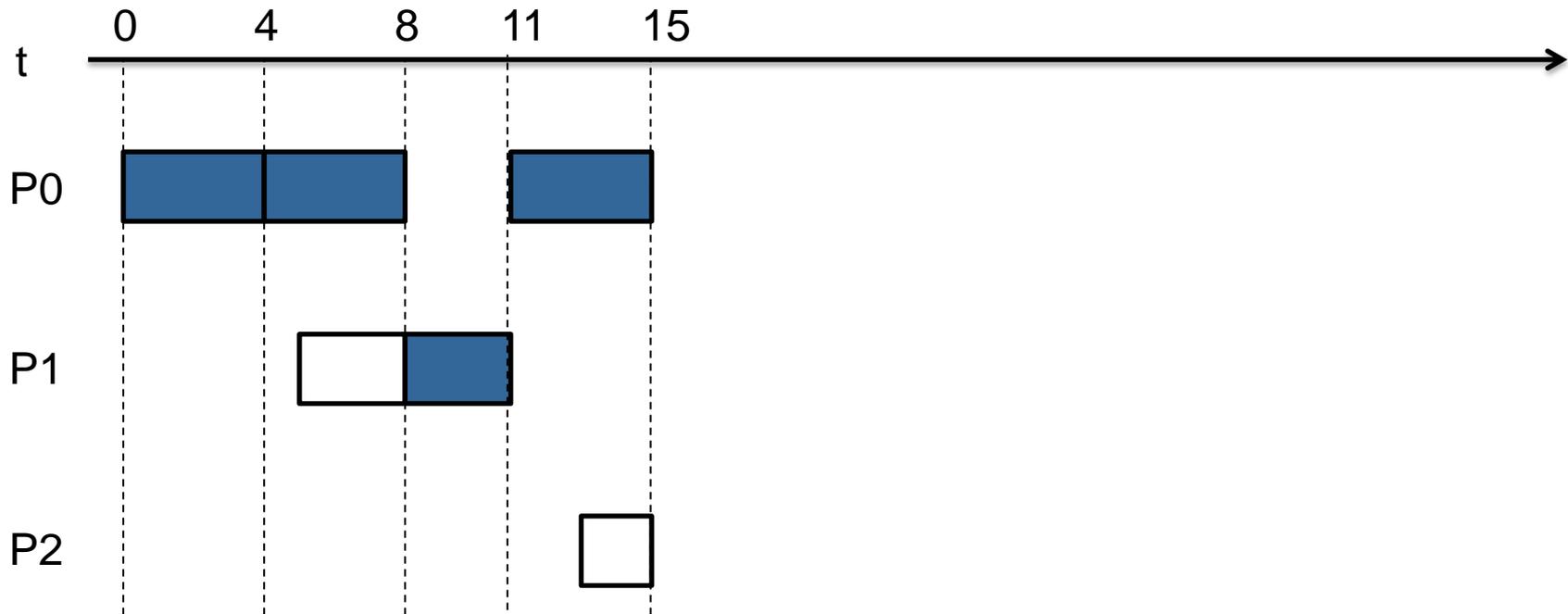
# Beispiel Round Robin

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



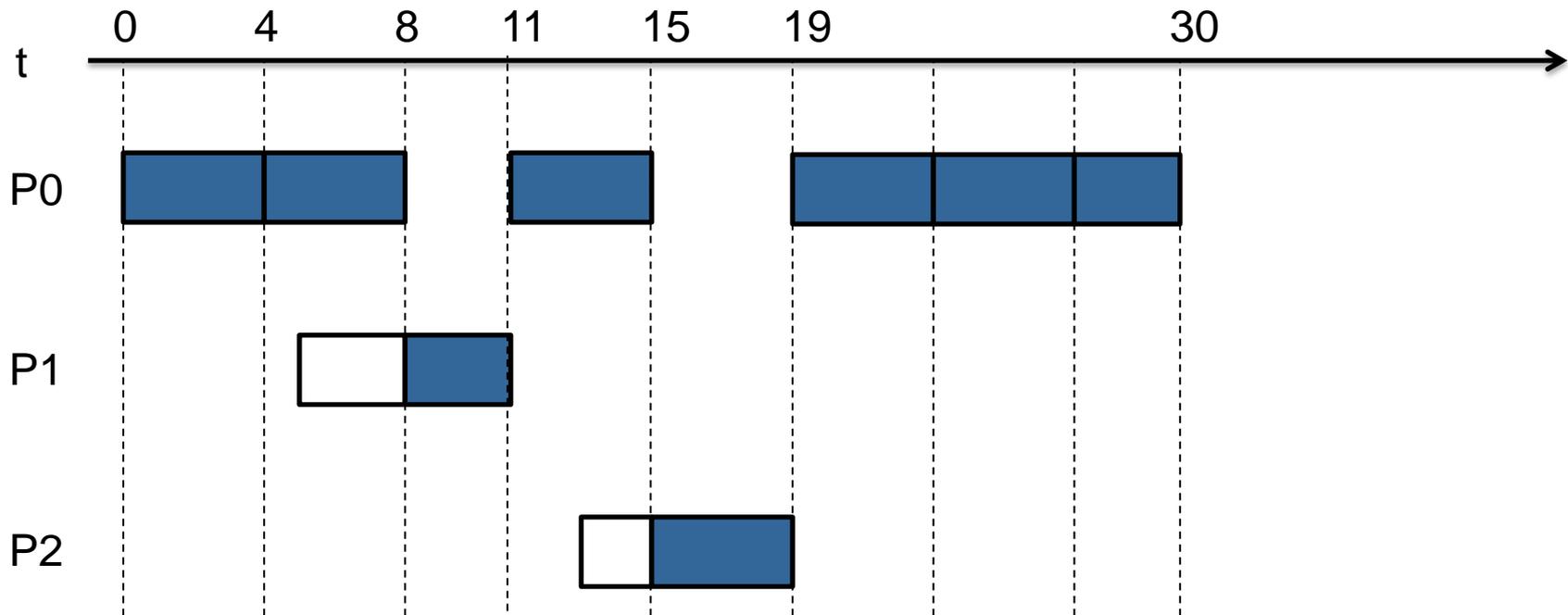
# Beispiel Round Robin

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel Round Robin

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel Round Robin

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



$d_0=30$



$d_1=6$

Mittlere Durchlaufzeit:  
 $d^*=14$



$d_2=6$

# Round Robin (RR)

## Analyse

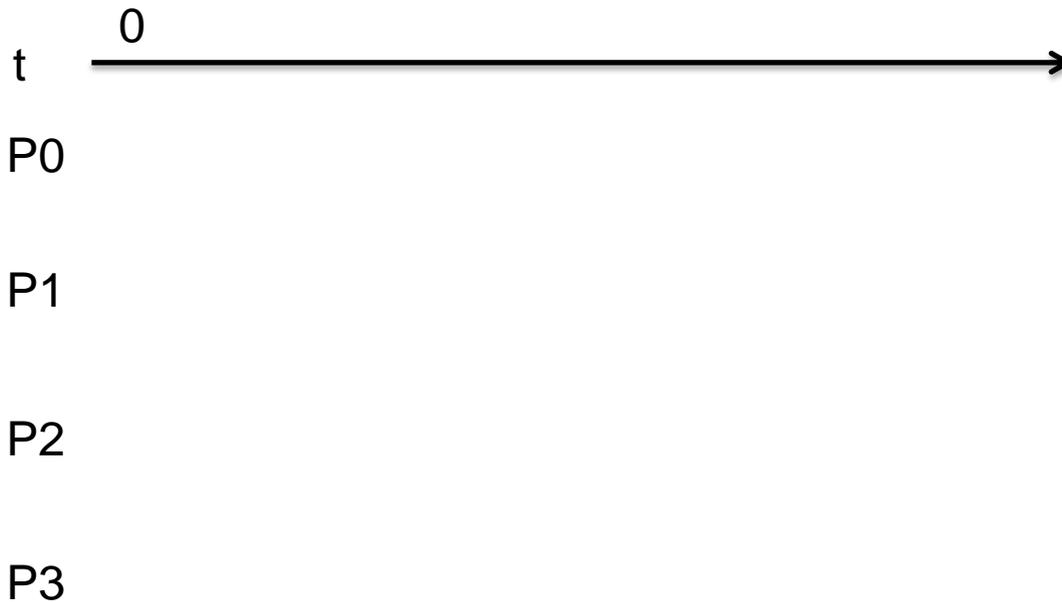
- Länge des Zeitintervalls ist essentiell
- Zu kurz: Aufwand für viele Prozesswechsel
- Zu lang: Ähnlich FCFS
- Sinnvoll: Entsprechend durchschnittlich benötigter CPU-Zeit
- Prozesse ohne Ein-/Ausgabe etwas begünstigt
- Prozesse mit E/A geben CPU vor Ablauf Zeitintervall ab und sind dann erst blockiert

# Shortest Job First (SJF) (1)

- Nicht-präemptive Strategie
- Auswahlfunktion:  $\min(s)$
- **Strategie:**
  - Benutzt Abschätzungen der Gesamtlaufzeit von Prozessen
  - Prozess mit kürzester geschätzter Laufzeit erhält CPU als erstes

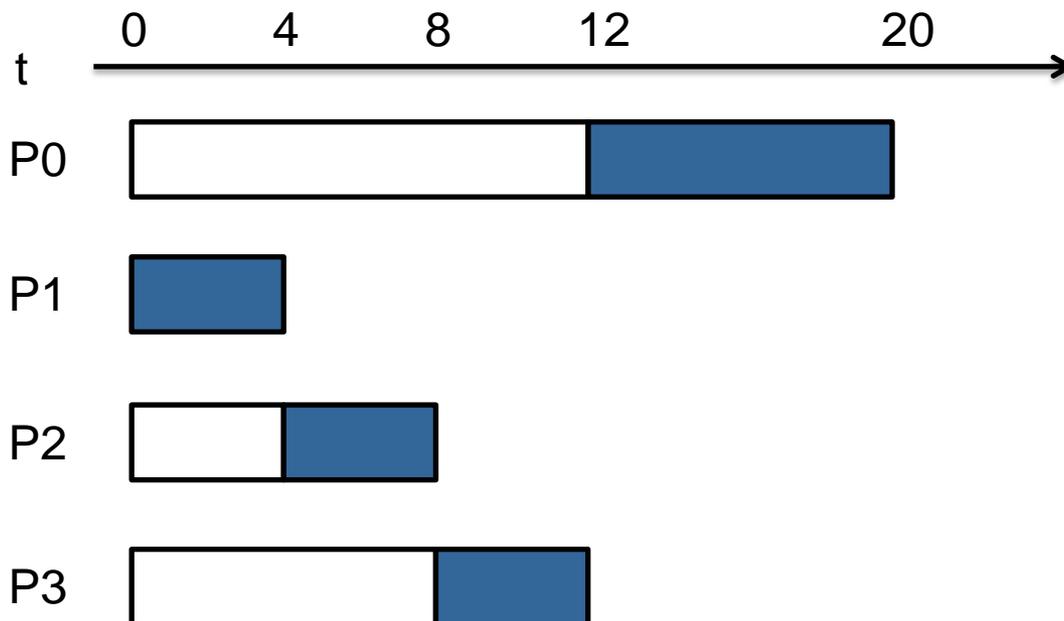
# Shortest Job First (2) – Beispiel

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	0	4
P2	0	4
P3	0	4



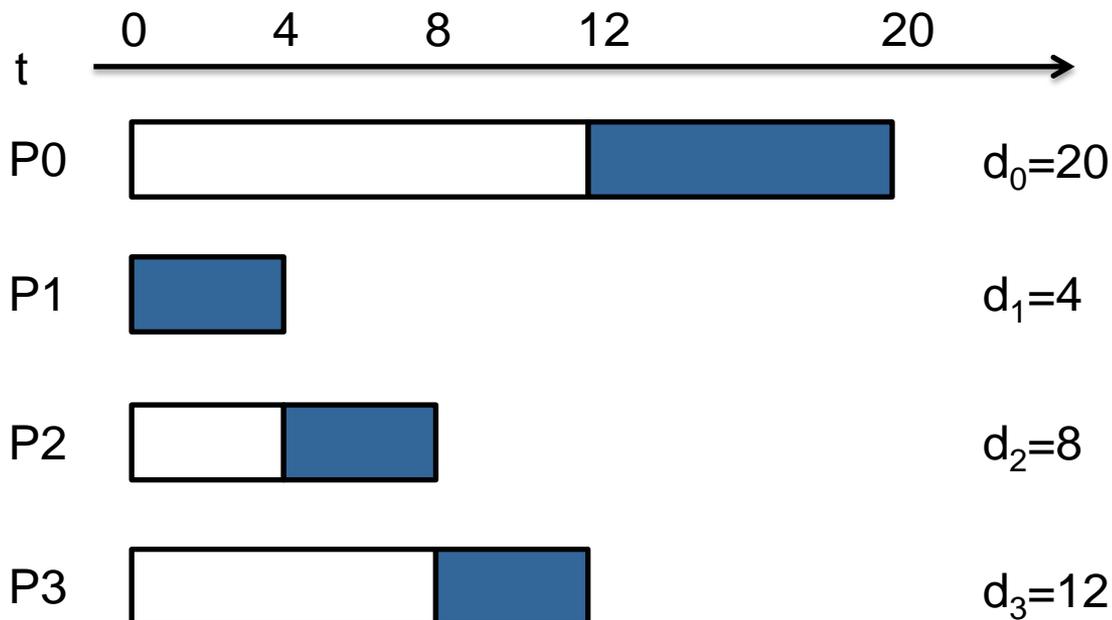
# Shortest Job First (2) – Beispiel

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	0	4
P2	0	4
P3	0	4



# Shortest Job First (2) – Beispiel

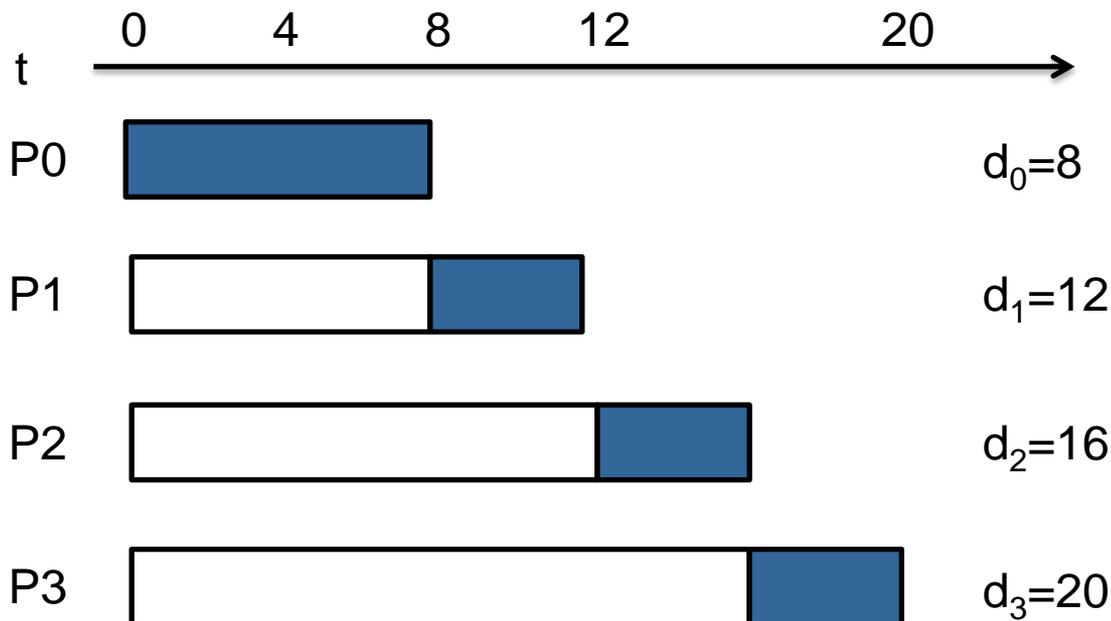
Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	0	4
P2	0	4
P3	0	4



Mittlere Durchlaufzeit:  
 $d^*=11$

# Andere Reihenfolge

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	0	4
P2	0	4
P3	0	4



Mittlere Durchlaufzeit:  
 $d^*=56/4=14$

# Optimalität von Shortest Job First

## Satz:

- Seien  $n$  Prozesse  $P_1, \dots, P_n$  mit Laufzeiten  $t_1, \dots, t_n$  gegeben und alle zur Zeit  $t = 0$  bereit
- Dann erzielt SJF die minimale durchschnittliche Durchlaufzeit

# Shortest Job First (SJF) (5)

**Beweis:**

Annahme: Ausführungsreihenfolge  $P_1, P_2, \dots, P_n$

Berechne für alle Prozesse  $P_i$  die Durchlaufzeiten  $d_i$ :

$$d_1 = t_1$$

$$d_2 = d_1 + t_2 = t_1 + t_2$$

$$d_3 = d_2 + t_3 = t_1 + t_2 + t_3$$

...

$$d_n = d_{n-1} + t_n = t_1 + t_2 + t_3 \dots + t_{n-1} + t_n$$

Also  $d_i = \sum_{j=1}^i t_j$

# Shortest Job First (SJF) (6)

Mittlere Durchlaufzeit:

$$\begin{aligned}d^* &= \frac{1}{n} \sum_{i=1}^n d_i \\&= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^i t_j \\&= \frac{1}{n} (t_1 + \underline{t_1 + t_2} + \underline{t_1 + t_2 + t_3} + \dots + \underline{t_1 + t_2 + t_3 + \dots + t_n}) \\&= \frac{1}{n} (n \cdot t_1 + (n-1) \cdot t_2 + (n-2) \cdot t_3 + \dots + t_n) \\&= \frac{1}{n} \cdot \sum_{i=1}^n (n-i+1) \cdot t_i \\&= \sum_{i=1}^n \frac{n-i+1}{n} \cdot t_i\end{aligned}$$

# Shortest Job First (SJF) (7)

- Gewichtete Summe über alle  $t_i$
- Gewicht von  $t_1$  ist  $\frac{n+1-1}{n} = \frac{n}{n} = 1$
- Gewicht von  $t_2$  ist  $\frac{n+1-2}{n} = \frac{n-1}{n}$
- Gewicht von  $t_n$  ist  $\frac{n+1-n}{n} = \frac{1}{n}$

# Shortest Job First (SJF) (7)

- Gewichtete Summe über alle  $t_i$
- Gewicht von  $t_1$  ist  $\frac{n+1-1}{n} = \frac{n}{n} = 1$
- Gewicht von  $t_2$  ist  $\frac{n+1-2}{n} = \frac{n-1}{n}$
- Gewicht von  $t_n$  ist  $\frac{n+1-n}{n} = \frac{1}{n}$
- Gewichtete Summe ist dann am kleinsten, wenn  $t_1 \leq t_2 \leq \dots \leq t_n$
- Also: SJF führt zur geringsten mittleren Durchlaufzeit

# Shortest Job First (SJF) (8)

## Analyse

- Erzielt minimale durchschnittliche Durchlaufzeit, sofern alle Prozesse **gleichzeitig verfügbar**
- Kurze Prozesse bevorzugt
- Gefahr, dass längere Prozesse verhungern
- Abschätzungen der Gesamtlaufzeit von Prozessen müssen gegeben sein

# Shortest Remaining Time (SRT)

- Präemptive Variante von SJF
- Auswahlfunktion:  $\min(s-e)$
- **Strategie:**
  - Prozess mit kürzester geschätzter **Restlaufzeit** erhält CPU
  - Keine Unterbrechung aktiver Prozesse nach bestimmtem Zeitintervall
  - Stattdessen: Auswertung der Restlaufzeiten nur, **wenn ein anderer Prozess bereit wird**

# Beispiel SRT

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



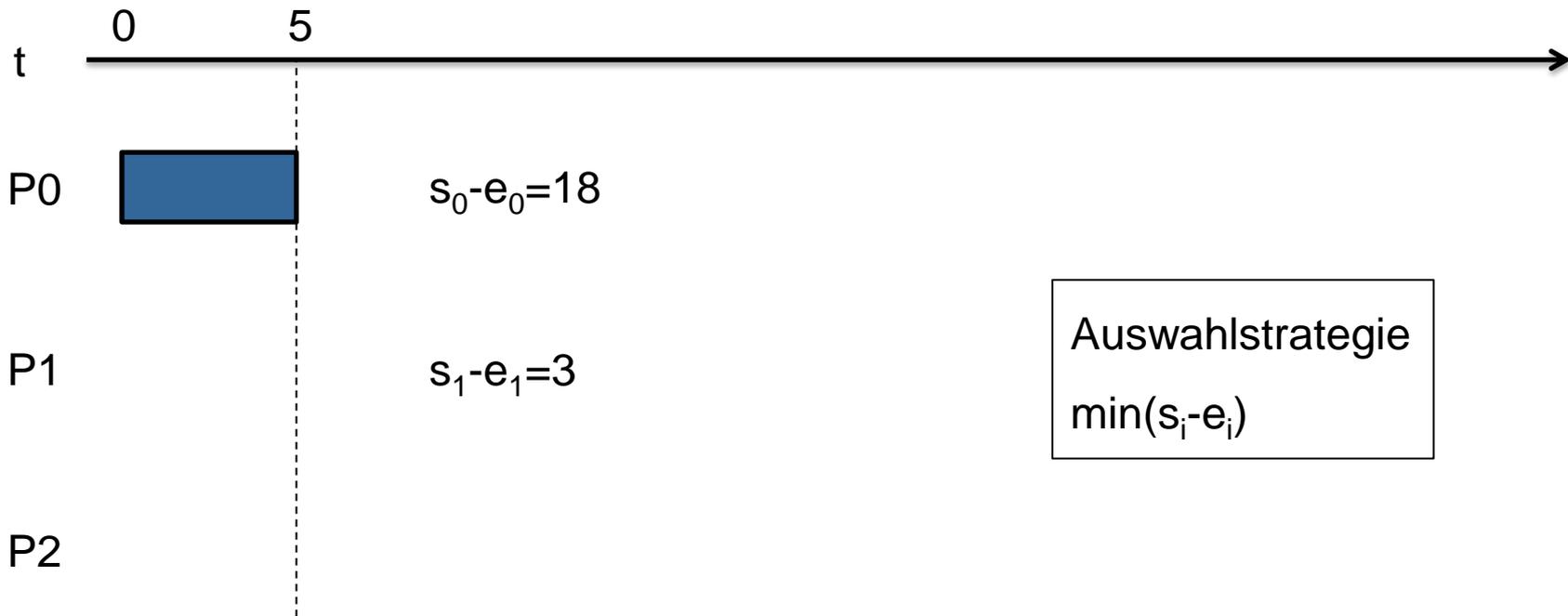
P0

P1

P2

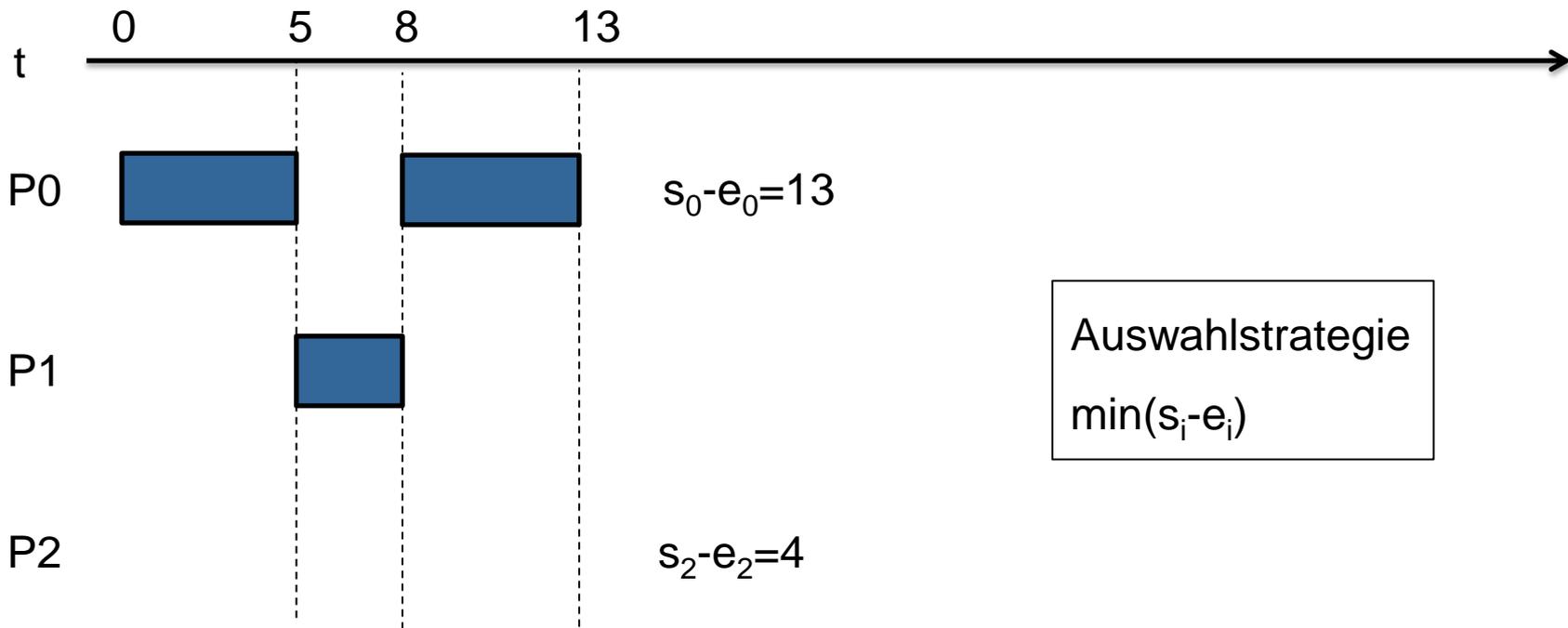
# Beispiel SRT

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



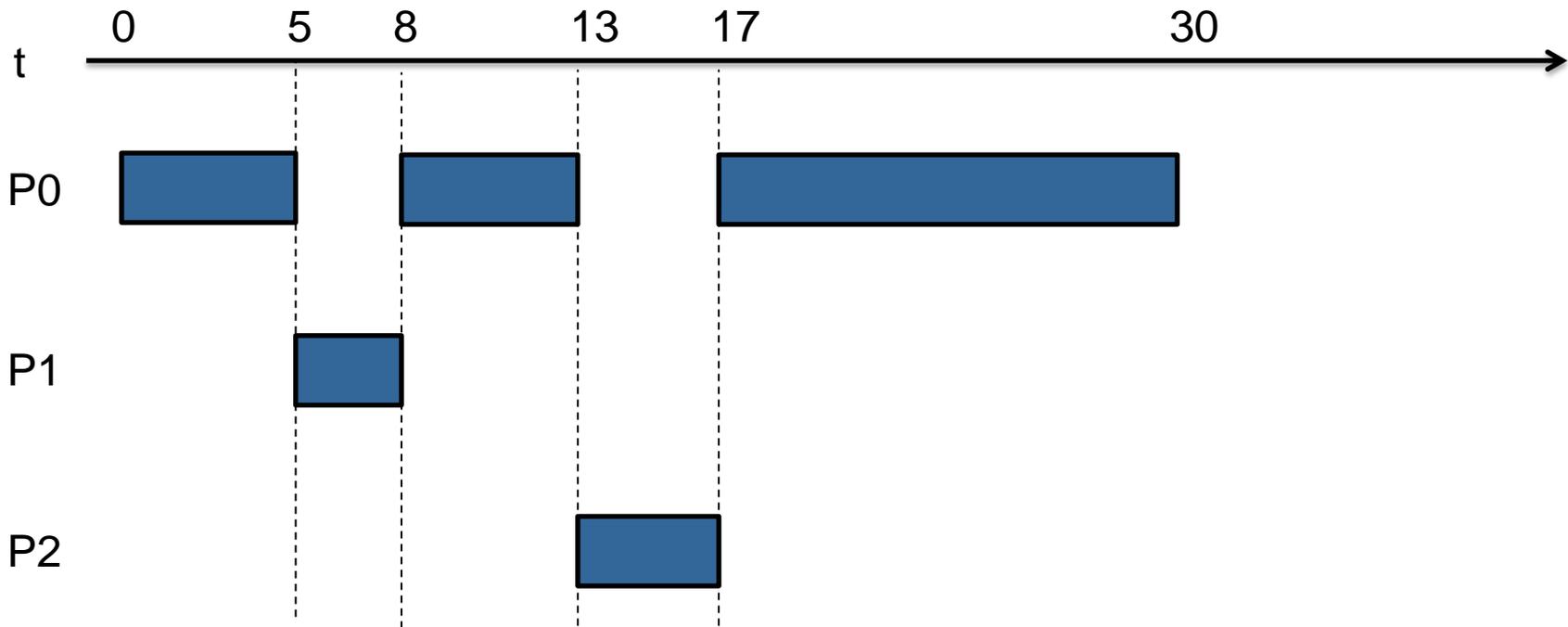
# Beispiel SRT

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel SRT

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



# Beispiel SRT

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	23
P1	5	3
P2	13	4



$d_0=30$

$d_1=3$

$d_2=4$

Mittlere Durchlaufzeit:  
 $d^*=12,3$

# Shortest Remaining Time (SRT)

## Analyse

- Benachteiligt lange Prozesse, auch Verhungern möglich (wie SJF)
- Aufwand für Prozesswechsel und Aufzeichnen von Ausführungszeiten
- Abschätzungen der Gesamtlaufzeit von Prozessen müssen gegeben sein
- Aber u.U. **bessere Durchlaufzeit**, weil kurze bereite Prozesse aktiven längeren Prozessen **sofort** vorgezogen werden

# Highest Response Ratio Next (HRRN)

- Nicht-präemptiv
- Auswahlfunktion:  $\max((w+s)/s)$
- **Strategie:**
  - Basiert auf normalisierter Durchlaufzeit („Response Ratio“)
  - $R = (w+s)/s$
  - Bei Prozessstart:  $R = 1.0$
  - Prozess mit höchstem  $R$  erhält Rechenzeit

# Beispiel HRRN

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	3	4
P2	6	3



P0

$R_0=1$

P1

Auswahlstrategie  
 $\max(R_i)$

P2

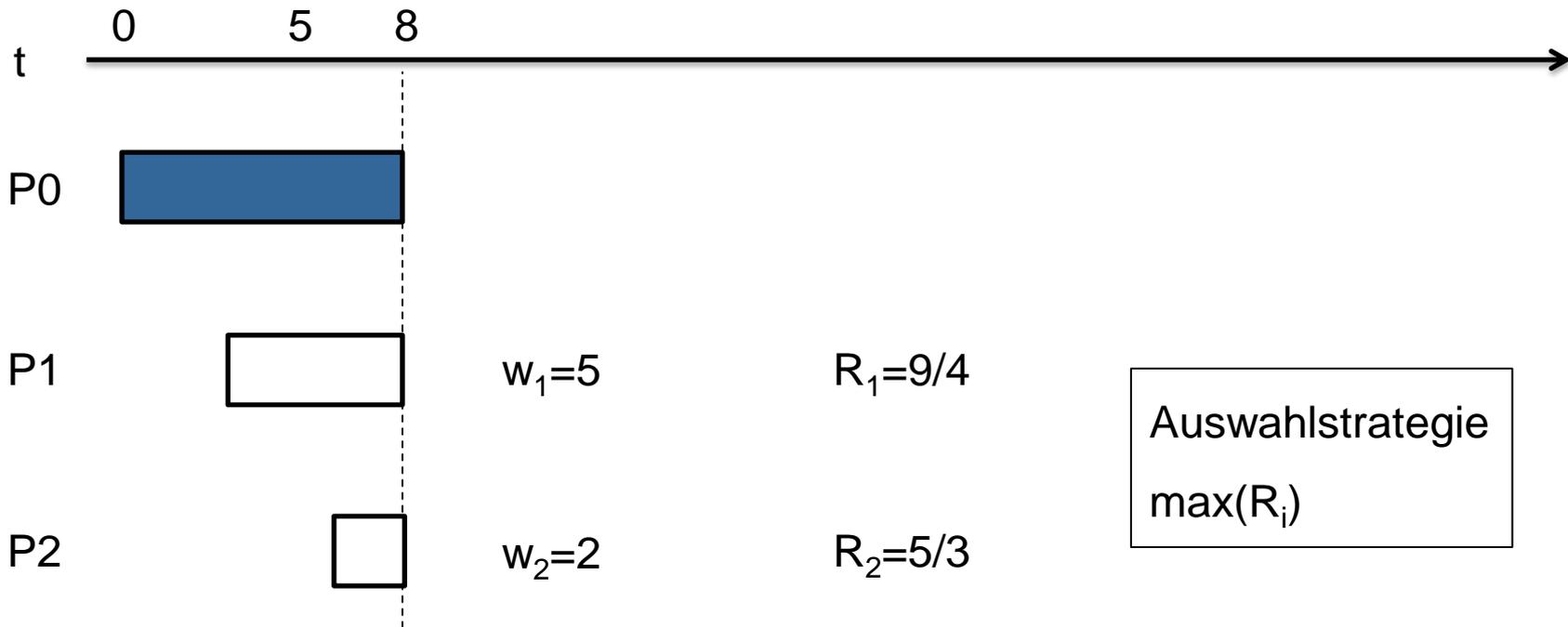
# Beispiel HRRN

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	3	4
P2	6	3



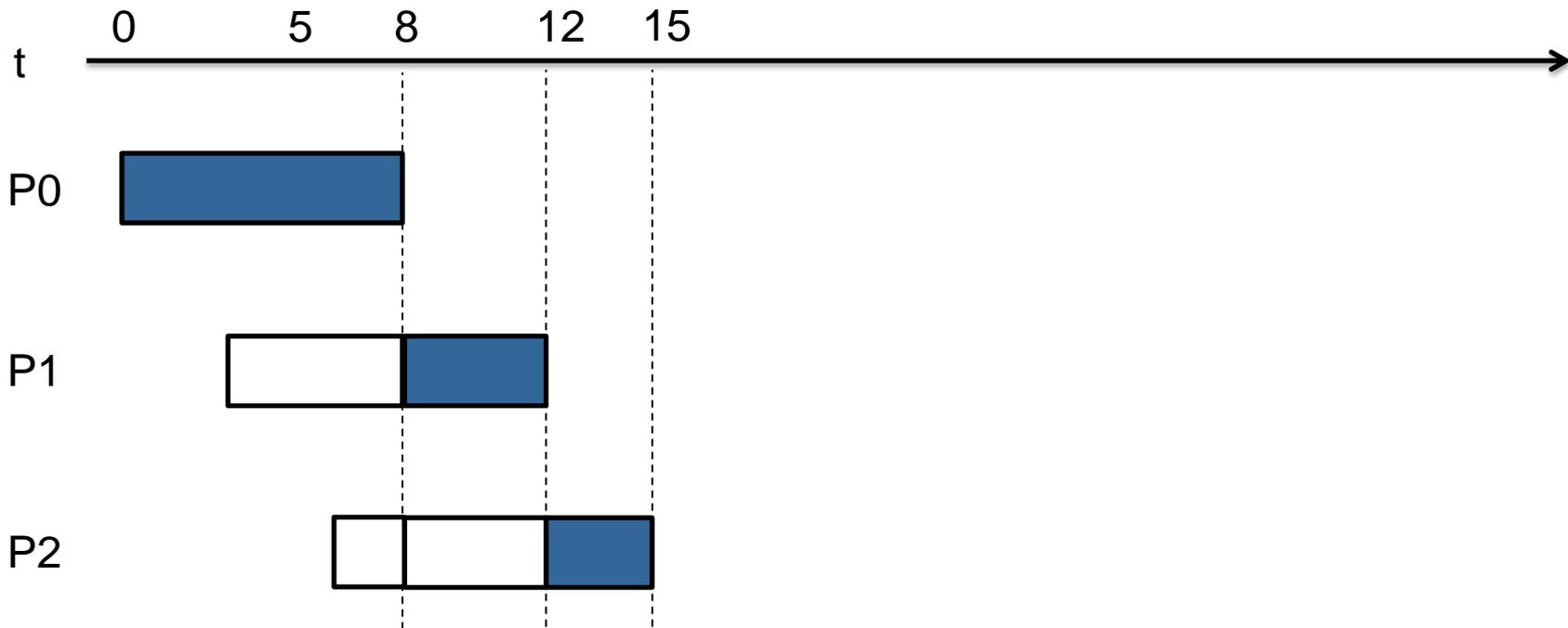
# Beispiel HRRN

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	3	4
P2	6	3



# Beispiel HRRN

Prozess	Erzeugungszeit	Benötigte Zeit s
P0	0	8
P1	3	4
P2	6	3



# Highest Response Ratio Next (HRRN)

## Analyse

- Begünstigt kurze Prozesse: Für kurze, wartende Prozesse wächst  $R$  schnell an
- Aber: Keine Livelocks für längere Prozesse
- Ähnliches Problem wie SJF, SRT: Laufzeitabschätzungen benötigt

# Feedback (1)

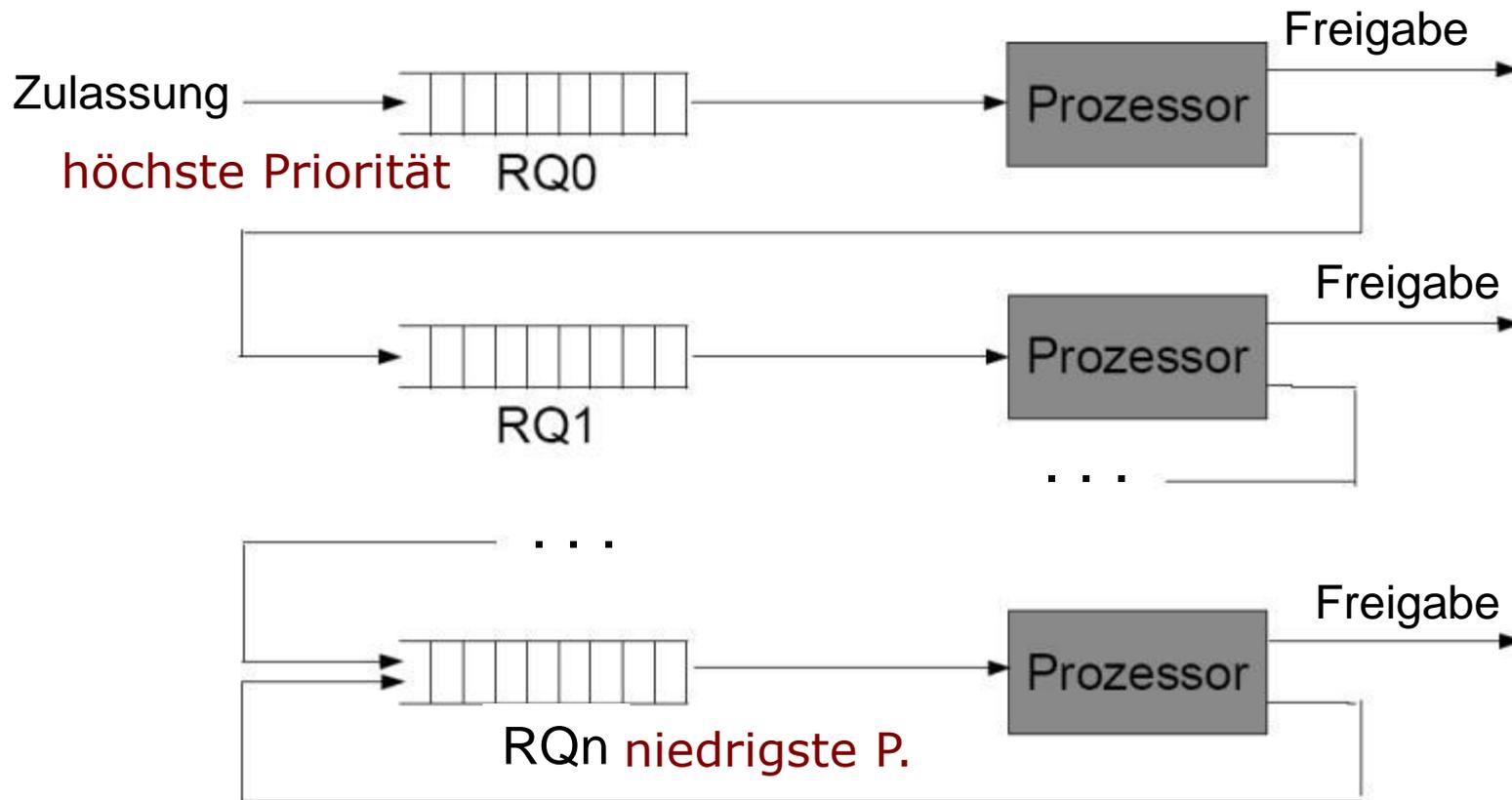
- **Idee:**

Benutze statt Gesamtlaufzeit von Prozessen die bisher verbrauchte CPU-Zeit

- **Strategie:**

- Präemptiv (Zeitintervall), dynamische Prioritäten
- Bei Abgabe der CPU: Einreihen in Warteschlange mit der nächst geringeren Priorität
- Abarbeitung der Warteschlangen nach Priorität
- Dadurch: Verbrauchte CPU-Zeit wird angenähert durch Anzahl **erzwungener CPU-Abgaben**

# Feedback (2)



- Innerhalb Warteschlangen: FCFS
- Bis auf letzte Warteschlange, dort RR

# Feedback (3)

## Analyse

- Bevorzugt E/A-lastige Prozesse
- Prozesse, die in der Vergangenheit viel CPU-Zeit verbraucht haben, werden bestraft
- Lange Prozesse können verhungern

# Feedback (4)

- **Variante 1:**
  - Prozesse aus niedrigeren Warteschlangen erhalten längere Rechenzeiten, wenn sie drankommen, z.B.  $2^i$  Zeiteinheiten für Prozesse aus Warteschlange  $RQ_i$
  - Dadurch auch weniger Kontextwechsel
  - Längere Prozesse können immer noch verhungern
- **Variante 2:**
  - Neuberechnen der Prioritäten von Zeit zu Zeit
  - Wartezeit geht in die Priorität ein (UNIX)

# „Traditionelles“ Unix-Scheduling

- Ziele:
  - Gute Antwortzeiten für interaktive Benutzer
  - Gleichzeitig: Hintergrundaufträge mit geringer Priorität sollen nicht verhungern

# Scheduling bei UNIX (1)

- Es gibt verschiedene Warteschlangen (wie bei Feedback) **mit unterschiedlichen Prioritäten**
- Anfangs: Erster Prozess der nichtleeren **Warteschlange mit höchster Priorität** ausgeführt
- Anschließend: Prozesse höchster Priorität werden untereinander nach **Round Robin** gescheduled

# Scheduling bei UNIX (2)

- Neuberechnung der Prioritäten in regelmäßigen Zeitabständen
- $\text{priority} = \text{CPU\_usage} + \text{nice} + \text{base}$   
(je kleiner der Wert, desto höher die Priorität)
- CPU\_usage
  - Maß für die CPU-Benutzung in der Vergangenheit
  - Berechnet durch exponentielle Glättung:

$$\text{CPU\_usage}(t) := \begin{cases} \text{CPU\_Anteil}(0) = 0 & \text{für } t = 0 \\ e^{-\frac{1}{T}} \cdot \text{CPU\_usage}(t-1) + \left(1 - e^{-\frac{1}{T}}\right) \cdot \text{CPU\_Anteil}(t) & \text{für } t > 0 \end{cases}$$

- CPU\_Anteil: Anteilig verbrauchte Rechenzeit des Prozesses im letzten Zeitschritt
- T: Konstanter Glättungsparameter

# Scheduling bei UNIX (3)

Berechnung CPU\_usage für  $e^{-\frac{1}{T}} = 0.5 \Leftrightarrow T \approx 1.4427$

mit  $a_t := \text{CPU\_Anteil}(t)$  und  $u_t := \text{CPU\_usage}(t)$

- $u_0 = \text{CPU\_usage}(0) = 0$
- $u_1 = 0.5u_0 + 0.5a_1 = 0.5a_1$
- $u_2 = 0.5u_1 + 0.5a_2 = 0.25a_1 + 0.5a_2$
- $u_3 = 0.5u_2 + 0.5a_3 = 0.125a_1 + 0.25a_2 + 0.5a_3$
- $u_4 = 0.5u_3 + 0.5a_4 = 0.0625a_1 + 0.125a_2 + 0.25a_3 + 0.5a_4$

→ Gewichtete Summe: Gewicht der alten Werte nimmt exponentiell ab

# Scheduling bei UNIX (4)

- nice:
  - Durch den Benutzer kontrollierbarer Wert, um einem Prozess eine niedrigere Priorität zu geben
- base:
  - Durch System gewählter Basis-Prioritätswert
  - Einteilung in feste Prioritätsgruppen
  - Höchste Priorität: Swapper
  - Niedrigste Priorität: Benutzerprozesse
  - Bei Benutzerprozessgruppe: Bevorzugung von Prozessen, die durch Abschluss einer E/A-Operation wieder bereit werden gegenüber CPU-lastigen Prozessen

# Scheduling-Algorithmen

## Zusammenfassung (1)

- **First Come First Served**: Prozess, der bereits am längsten wartet, nicht präemptiv
- **Round Robin**: Aktive Prozesse werden nach bestimmter Zeit abgebrochen
- **Shortest Job First**: Prozess mit kürzester erwarteter Rechenzeit; keine Unterbrechung
- **Shortest Remaining Time**: Prozess mit kürzester geschätzter Restlaufzeit; Unterbrechung nur wenn anderer Prozess rechenbereit wird

# Scheduling-Algorithmen

## Zusammenfassung (2)

- **Highest Response Ratio Next:** Prozess mit größter normalisierter Durchlaufzeit; nicht präemptiv
- **Feedback:** Warteschlangen von Prozessen, in die sie u.a. entsprechend ihrer Ausführungsgeschichte eingeteilt werden; Unterbrechung nach bestimmter Zeitdauer

# Thread-Scheduling

- Prozesse können mehrere Threads besitzen
- Erinnerung: „leichtgewichtige“ Prozesse; gemeinsame Nutzung des Adressraumes
- Performanzgewinn z.B. bei rechenintensivem Teil und E/A
- Parallelität in 2 Ebenen: Prozesse / Threads
- Unterscheidung: Threads auf Benutzerebene / auf Systemebene

# Threads auf Benutzerebene

- System weiß nicht Bescheid über die Existenz der Threads eines Benutzerprogramms
- Scheduling findet auf **Prozessebene** statt
- **Thread-Scheduler** entscheidet dann, welcher Thread von gewähltem Prozess laufen soll
- Thread wird nicht unterbrochen innerhalb Zeitintervall für Prozess
- Läuft, bis er warten muss oder fertig ist, oder bis das Zeitintervall abgelaufen ist und ein **anderer Prozess** vom Scheduler gewählt wird

# Threads auf Systemebene

- Scheduling findet auf **Thread-Ebene** statt
- Voller Kontextwechsel u.U. nötig, wenn neuer aktiver Thread zu anderem Prozess gehört
- Zwei Threads gleichwichtig, einer gehört zum gleichen Prozess wie ein gerade blockierter Thread, gib diesem den Vorzug

# Zusammenfassung

- Drei Arten von Scheduling (kurz-, mittel-, langfristig) existieren
- Es gibt eine Vielzahl von Kriterien (Benutzer-, Systemorientiert)
- Es gibt viele verschiedene Strategien für das kurzfristige Scheduling
- Wahl des Algorithmus hängt ab von der Anwendung
- Prioritäten und bisherige Rechenzeit sollten in Auswahlentscheidung mit eingehen