3. Scheduling

- Überblick
  - 3.1 Einführung
  - 3.2 Schedulingstrategien
  - 3.3 Multilevel-Scheduling
  - 3.4 Scheduling mit Sollzeitpunkten
  - 3.5 Fallstudien Unix und Windows

3.1 Einführung

Jobs haben Eigenschaften: periodisch/einmalig, Dauer bekannt (oder unbekannt), benötigte Ressourcen, unterbrechbar oder nicht, Deadline (hart oder weich), Abhängigkeiten von anderen Jobs

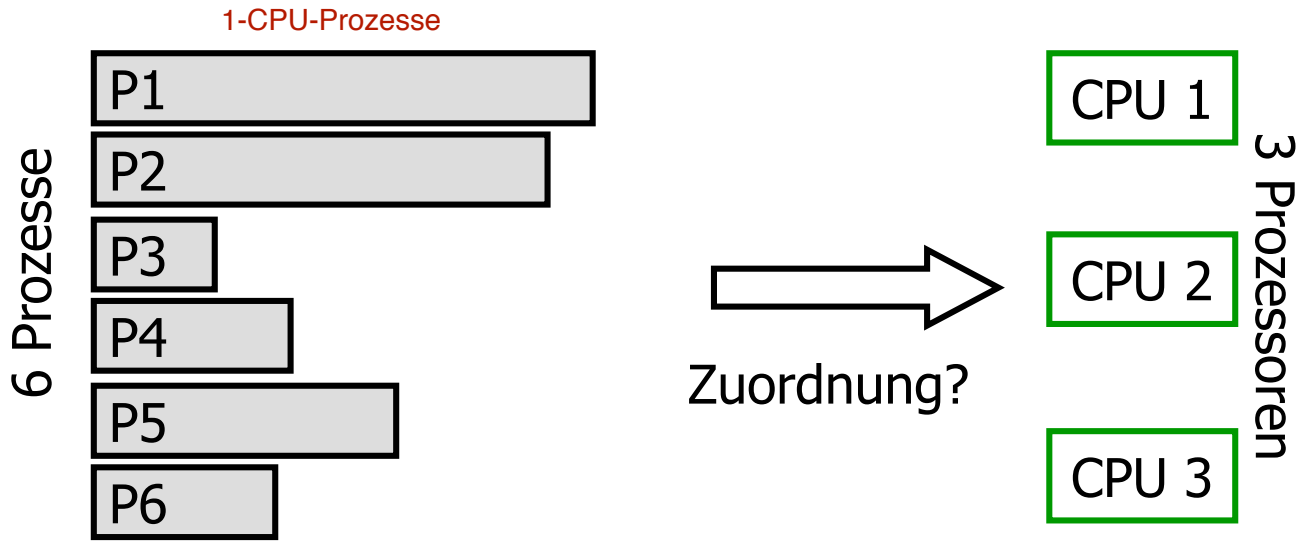
-> je nach Eigenschaften sind verschiedene Verfahren sinnvoll

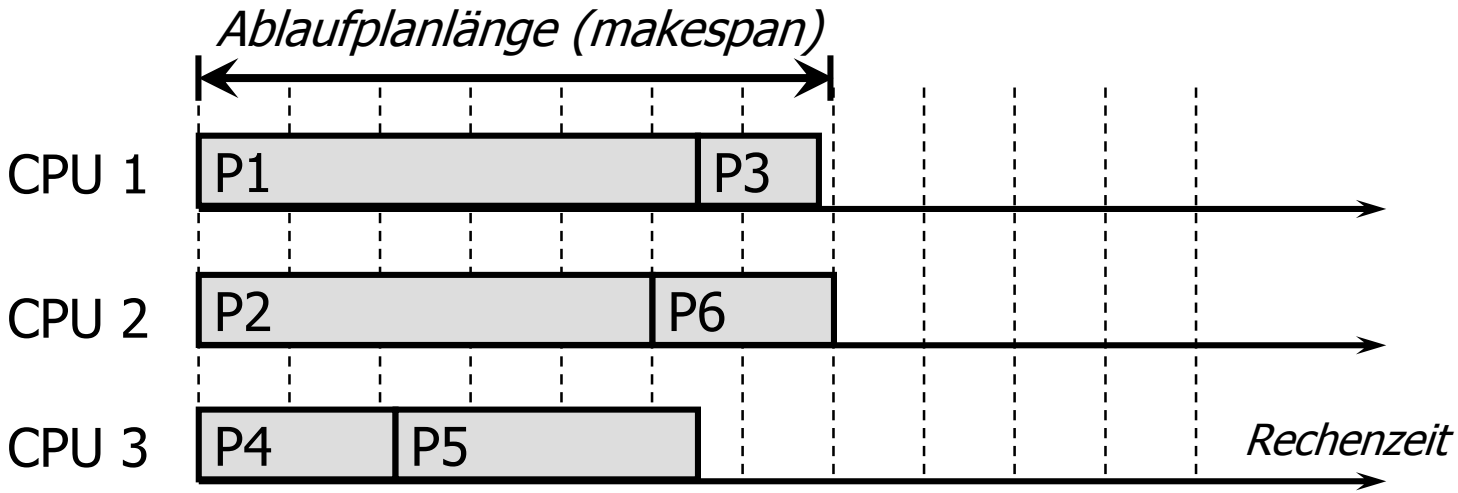
- Scheduling (Ablaufplanung)
  - Räumliche und zeitliche Zuordnung von Aktivitäten zu Instanzen, welche diese Aktivitäten durchführen können
- Welcher Prozess soll ausgeführt werden? Auswahl durch
  - Scheduler als Teil der Systemsoftware anhand der
  - Scheduling-Strategie: Zuordnungsalgorithmus Prozesse → Prozessoren
    - Short-Term-Scheduler: Prozesszuteilung unter bereiten Prozessen, einfache und effizient ausführbare Strategie
    - Long-Term-Scheduler: Längerfristige, komplexe Planung der Ressourcenzuteilung, etwa Speicherauslagerung
- Scheduling wichtig für subjektive Wahrnehmung der Rechnerleistung
  - P1: Schließen eines Fensters, aktualisieren der Oberfläche (2 sec)
  - P2: Senden einer E-mail (2 sec)
  - Reihenfolge P2P1: Benutzer nimmt die Verzögerung sofort wahr, empfindet den Rechner als langsam
  - Reihenfolge P1P2: Verzögerung des E-mail-Versands um 2 Sekunden bleibt wahrscheinlich unbemerkt

Folie 3 neu: Zeitskala

Klassisches Scheduling-Problem

- Zuordnung von Prozessen zu Prozessoren so, dass die Ausführungszeit minimiert wird





Gantt-Diagramm

Gestaltungsparameter für Scheduling

- Prozessmenge statisch oder dynamisch?

  auch: geschlossenes System, z.B. Embedded Systems

  - Statische Prozessmenge: Keine weiteren Prozesse kommen hinzu, d.h. alle Prozesse sind gegeben und ablauffähig
  - Dynamische Prozessmenge: Während der Ausführung können neue Prozesse hinzukommen, auf die reagiert werden muss

  statisch: alle Möglichen Kombinationen (theoretisch) bekannt
  dynamisch: unbekannt, was gleichzeitig oder überhaupt geschehen kann

- On-line oder Off-line Scheduling
  - Off-line (irgendwann vorher): Alle Prozesse – auch zukünftige Ankünfte – sind bekannt ⇒ vollständige Information liegt vor
  - On-line (zur Laufzeit): Lediglich aktuelle Prozesse bekannt ⇒ Entscheidungsfindung auf Grund unvollständiger Information

  Off-Line = vor der Laufzeit geplant, dafür müssen alle Informationen bekannt sein
  On-Line: Planung zur Laufzeit -> jeweils unvollständige Information (z.B. wie lange ein Ausfall dauern wird)

Gestaltungsparameter für Scheduling (2)

- Plattform: Einprozessor/Mehrprozessor
  - Mehrprozessor: Identische oder unterschiedliche Prozessoren
    $\Rightarrow$ evtl. Variation der Ausführungsgeschwindigkeiten
- Verdrängung möglich?
  - Mit Verdrängung können Schedulingziele besser erreicht werden

Zu jedem Zeitpunkt kann der Zeitplan verändert werden, wenn sich die Prioritäten während der Laufzeit verändern.
-> Prozesse, die nicht unterbrochen werden können, sind schwer zu planen

- Abhängigkeiten zwischen Prozessen?
  - Reihenfolgebeziehung (partielle Ordnung)
  - Synchronisierte Zuordnung der parallelen Prozesse eines Programms
  - Kommunikationszeiten zu berücksichtigen?
  - Rüstzeiten (Umschaltzeiten) zu berücksichtigen?

Gestaltungsparameter für Scheduling (3)

- Sind Prioritäten zu berücksichtigen?
  - Statische Prioritäten: a-priori (von außen) vorgegeben
  - Dynamische Prioritäten: Bestimmung während der Ausführung

- Sollzeitpunkte (= Deadlines) zu berücksichtigen? (oft bei Realzeitsystemen)
  - Prozesse müssen zu bestimmten Zeitpunkten beendet werden

- Existieren periodische, regelmäßig wiederkehrende Prozesse?

- Zu erreichendes Ziel, d.h. zu optimierende Zielfunktion, wie z.B.
  - Länge des Ablaufplans (min) — Gesamtdauer minimieren
  - Maximale Antwortzeit (min) — möglichst kurze Wartezeit für alle, z.B. bei Hotlines
  - Mittlere (gewichtete) Antwortzeit (min) — ähnlich max. Antwortzeit, aber Dauer "fairer" verteilt
  - Anzahl Prozessoren (min)
  - Durchsatz (max) — Aufgaben / Zeit erledigt
  - Prozessorauslastung (max)
  - Maximale Verspätung (min) — wenn Verspätung nötig, dann soll diese minimal sein

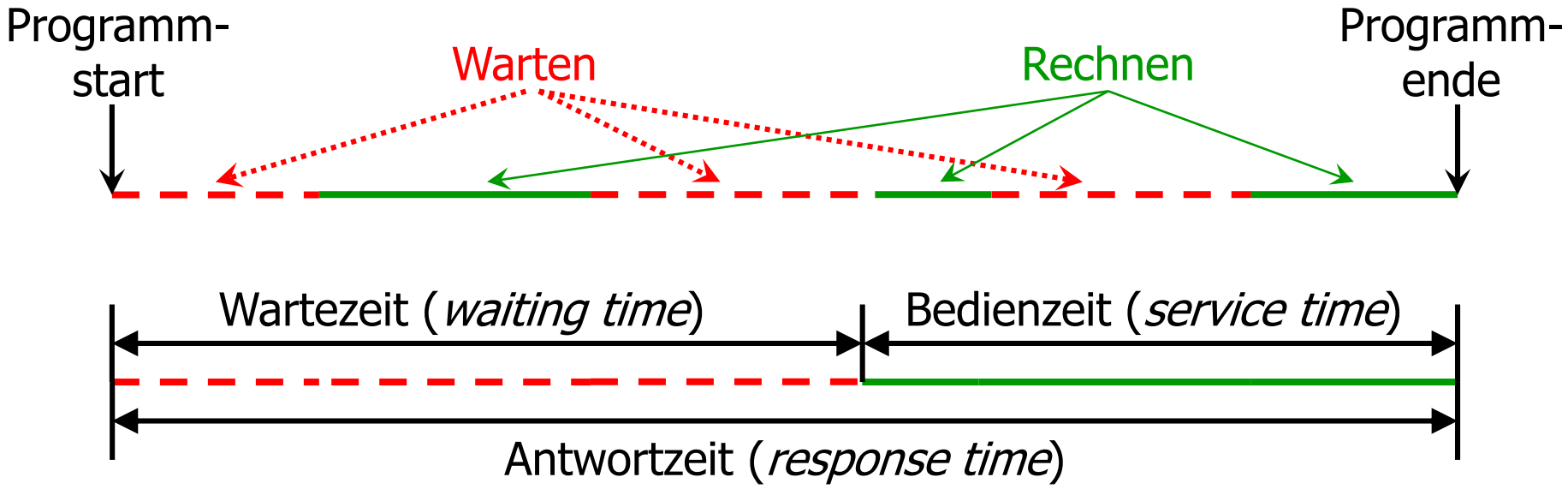
Problem: Ziele sind meistens nicht kompatibel, d.h. man muss sich entscheiden

Detaillierte Betrachtung der Schedulingziele

- Ziele für alle Systemarten
  - Fairness: gerechte Verteilung der Rechenzeiten an Bewerber
  - Policy Enforcement: Transparente Entscheidungskriterien
  - Balance: Alle Teile des Systems sind ausgelastet
- Ziele bei Stapelverarbeitungssystemen Batch-Systeme
  - Maximiere Durchsatz, d.h. Jobs/Zeiteinheit, z.B. Jobs/Stunde
  - Minimiere Turnaround-Zeit: Zeit zwischen Auftragsstart und Auftragende
- Interaktive Systeme
  - Minimiere die Antwortzeit für Anfragen
  - Proportionalität: Anpassung des Systemverhaltens auf das aktuelle Benutzerprofil
- Echtzeitsysteme
  - Einhaltung von Sollzeitpunkten

3.2 Scheduling in Mehrprogrammsystemen

- Zu erreichende Ziele
  - Hohe Effizienz ⇒ Hohe Auslastung des Prozessors
  - Geringe Antwortzeit bei interaktiven Prozessen und hoher Durchsatz bei Stapelbetrieb (*batch processing*)
  - Fairness, d.h. gerechte Verteilung der Prozessorleistung und der Wartezeit unter den Prozessen



Schedulingstrategien in Mehrprogrammsystemen

- Annahmen
  - Homogenes (symmetrisches) Multiprozessorsystem
  - Dynamische Prozessmenge
  - Keine Abhängigkeiten zwischen Prozessen
  - Dynamisches On-line-Scheduling
  - Keine Sollzeitpunkte
- Strategiealternativen
  - Ohne / mit Verdrängung des Prozesses
  - Ohne / mit Prioritäten
  - Unabhängig / abhängig von der Bedienzeit

Standardstrategien

Gegeben seien die folgenden fünf Prozesse:

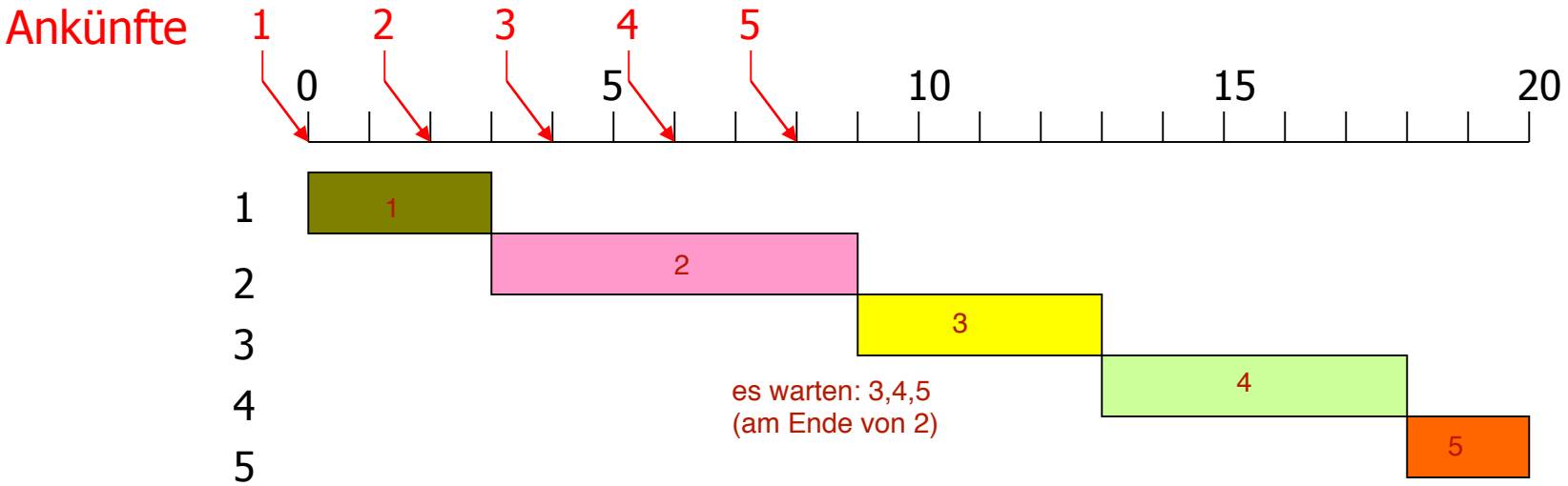
ab wann kann der Prozess arbeiten? Service Time

Nr.	Ankunft	Bedienzeit	Priorität
1	0	3	2
2	2	6	4
3	4	4	1
4	6	5	5
5	8	2	3

FCFS (First Come First Served) oder FIFO (First In First Out)

- Arbeitsweise
  - Bearbeitung der Prozesse in der Reihenfolge ihrer Ankunft in der Bereitliste
  - Prozessorbesitz bis zum Ende oder zur freiwilligen Aufgabe
  - Anmerkung: Entspricht der Alltagserfahrung.

-> Prioriäten werden ignoriert

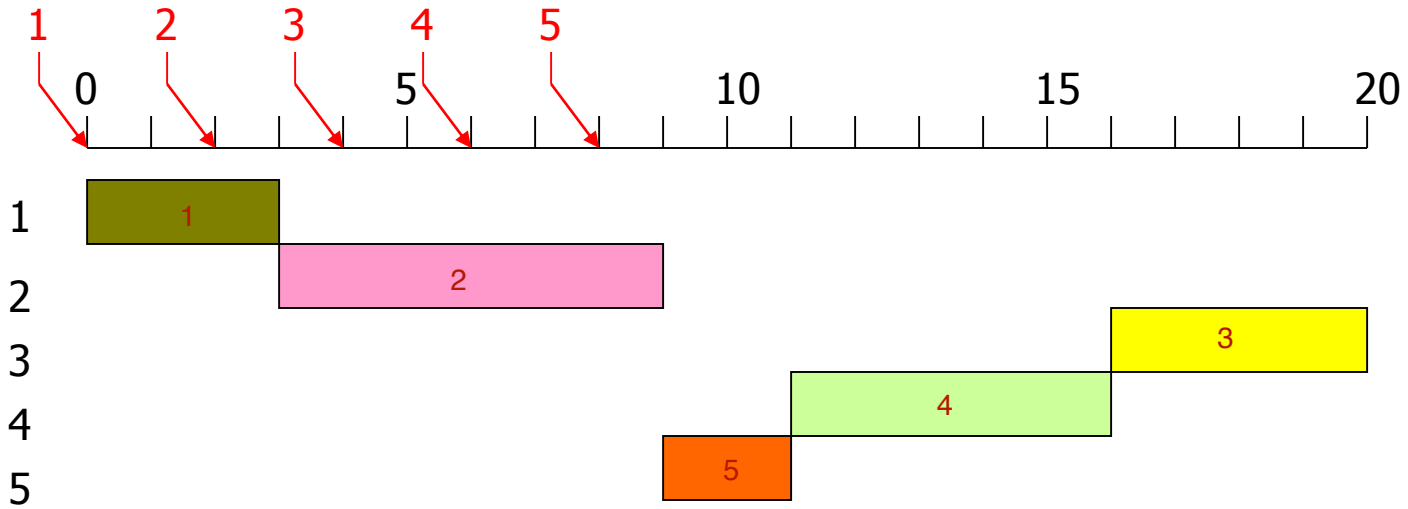


Vorteile: kein Overhead (optimale Gesamtzeit), transparent
Nachteile: lange Wartezeiten einzelner Prozesse (hier Prozess 5, sehr kurz, muss aber sehr lange warten)

LCFS (Last Come First Served)

- Arbeitsweise
  - Bearbeitung der Prozesse in der umgekehrten Reihenfolge ihrer Ankunft in der Bereitliste
  - Prozessorbesitz bis zum Ende oder zur freiwilligen Aufgabe
  - Anmerkung: In dieser reinen Form selten benutzt

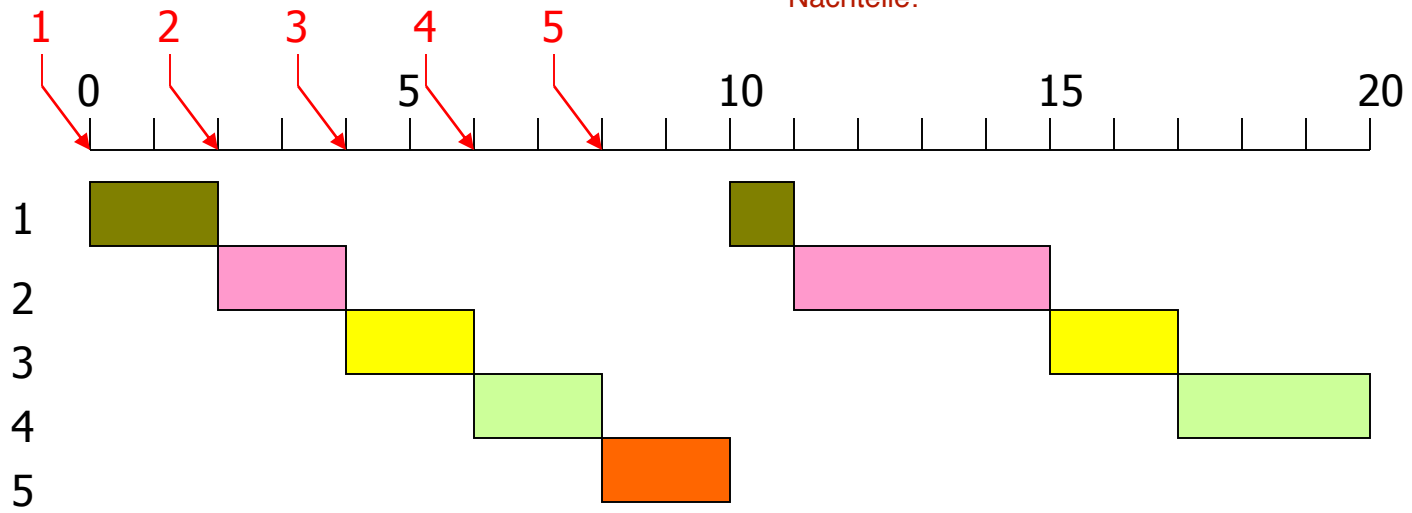
Nachteile: “unfair”, Prozesse unten im Stapel warten lange oder kommen evtl. nie dran (verhungern)
Vorteil: Hier bessere maximale Wartezeit



LCFS-PR (Last Come First Served - Preemptive Resume)

- Neuankömmling in Bereitliste verdrängt den rechnenden Prozess, verdrängter Prozess wird hinten in die Warteschlange eingereiht
- Ziel ist die Bevorzugung kurzer Prozesse.
  - Kurzer Prozess hat die Chance, schnell (vor der nächsten Ankunft) fertig zu werden
  - Ein langer Prozess wird u.U. mehrfach verdrängt
- Nach dem Eintreffen aller Prozesse wird die Warteschlange nach FIFO abgearbeitet

Vorteile: kurze Prozesse werden schnell fertig
Nachteile:

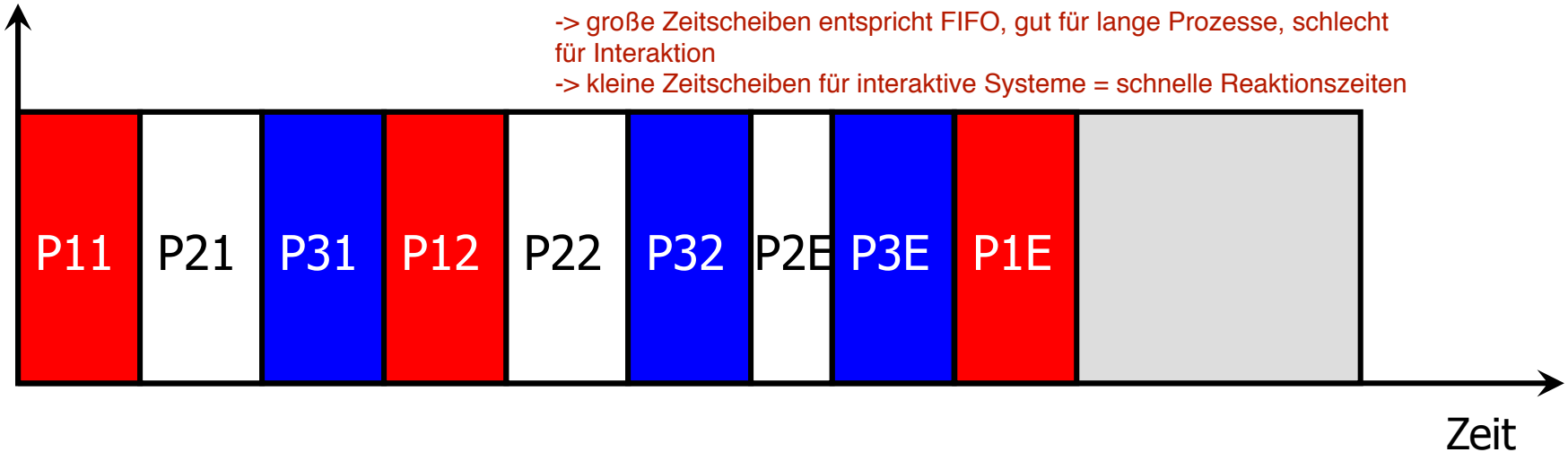


Zeitscheibenbetrieb (Time Sharing Mode)

Round Robin

- Abwicklung taktgesteuert und nebenläufig
  - Jeder Prozess erhält im festen Takt ein Zeitfenster zugeteilt
  - Ist er am Ende des Zeitfensters nicht fertig, dann wird er unterbrochen und in einer Warteschlange hinten angestellt
  - Änderung der Taktung durch Prioritäten, Warten auf Fertigstellung eines E/A-Auftrags, ...
- Voraussetzung: Prozesse unterbrechbar und Mechanismen zur Prozessumschaltung vorhanden

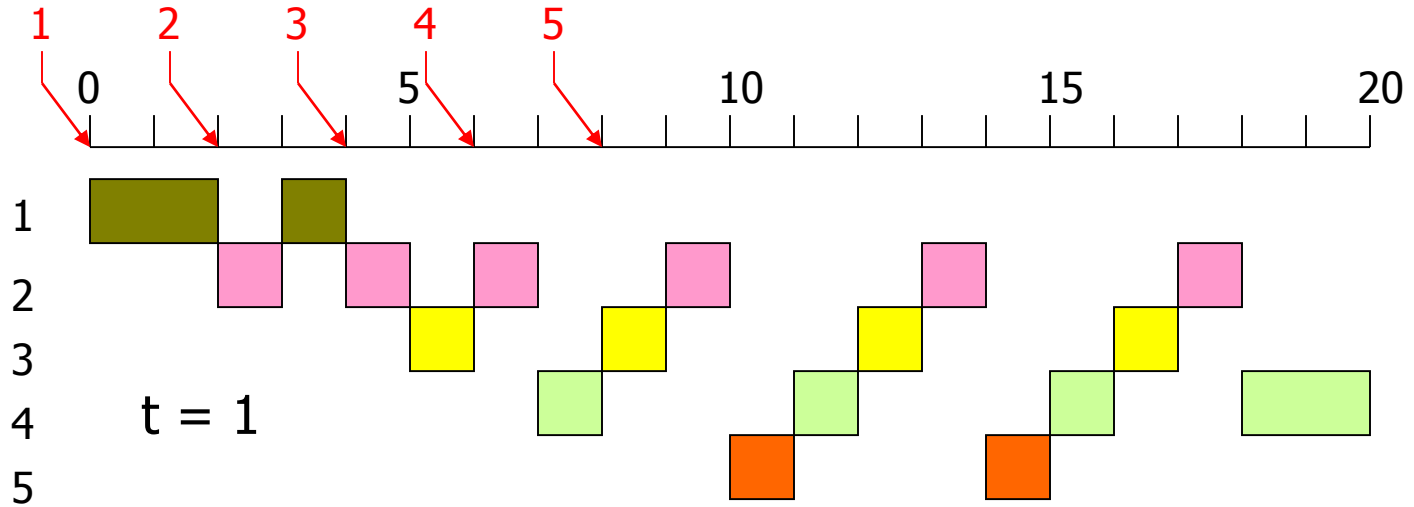
Nachteile: nicht die insgesamt schnellste Zeit durch Unterbrechungen & extra Zeit durch Wechsel
-> große Zeitscheiben entspricht FIFO, gut für lange Prozesse, schlecht für Interaktion
-> kleine Zeitscheiben für interaktive Systeme = schnelle Reaktionszeiten

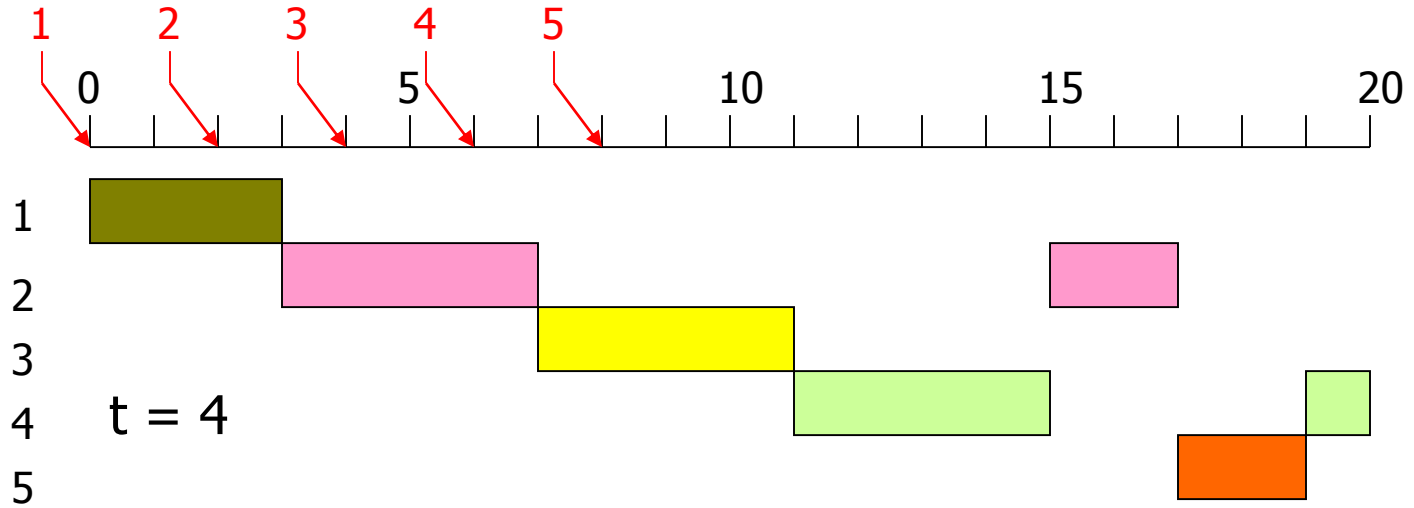


Round Robin: Zeitscheibenlänge

- Ziel des Verfahrens ist die gleichmäßige Verteilung der Prozessorkapazität und der Wartezeit auf die Prozesse
- Wahl der Zeitscheibenlänge t ist Optimierungsproblem
  - Für großes t nähert sich RR der Reihenfolgestrategie FCFS
  - Für kleines t schlägt der Aufwand für das häufige Umschalten negativ zu Buche
- Üblich sind Zeiten im msec-Bereich

RR Round Robin



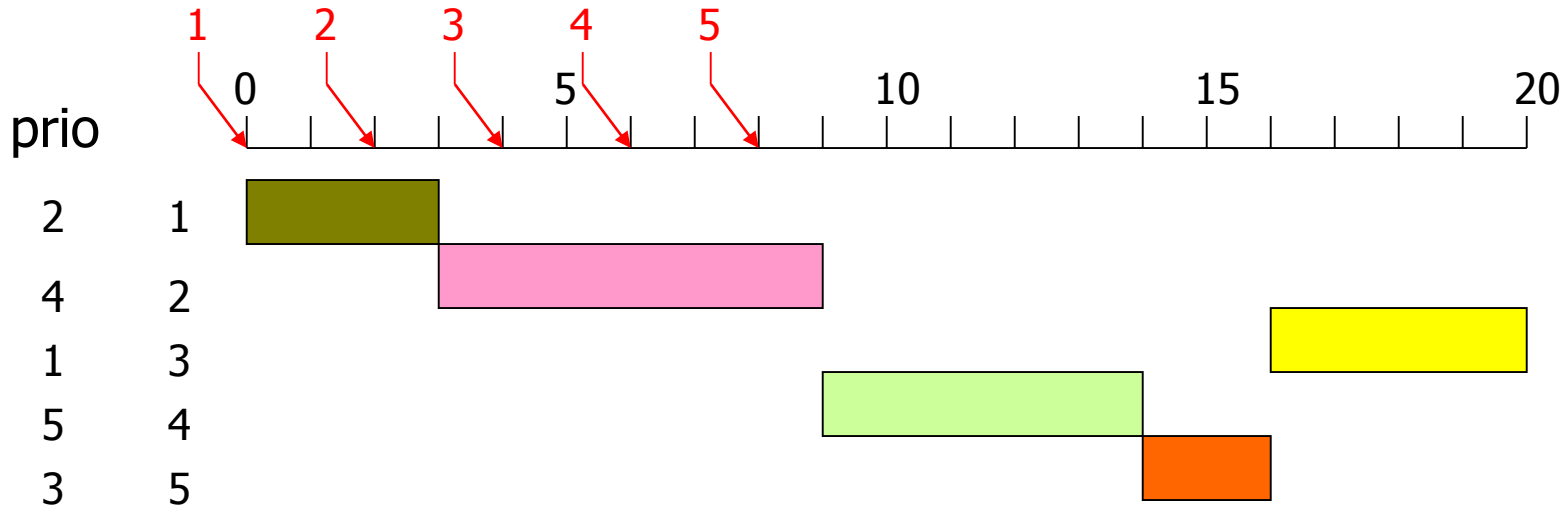


PRIO-NP
(Priorities – non preemptive)

- Neuankömmlinge werden nach ihrer Priorität in die Bereitliste eingeordnet
  - Prozessorbesitz bis zum Ende oder zur freiwilligen Aufgabe

PrioQueue ohne Verdrängung von laufenden Prozessen

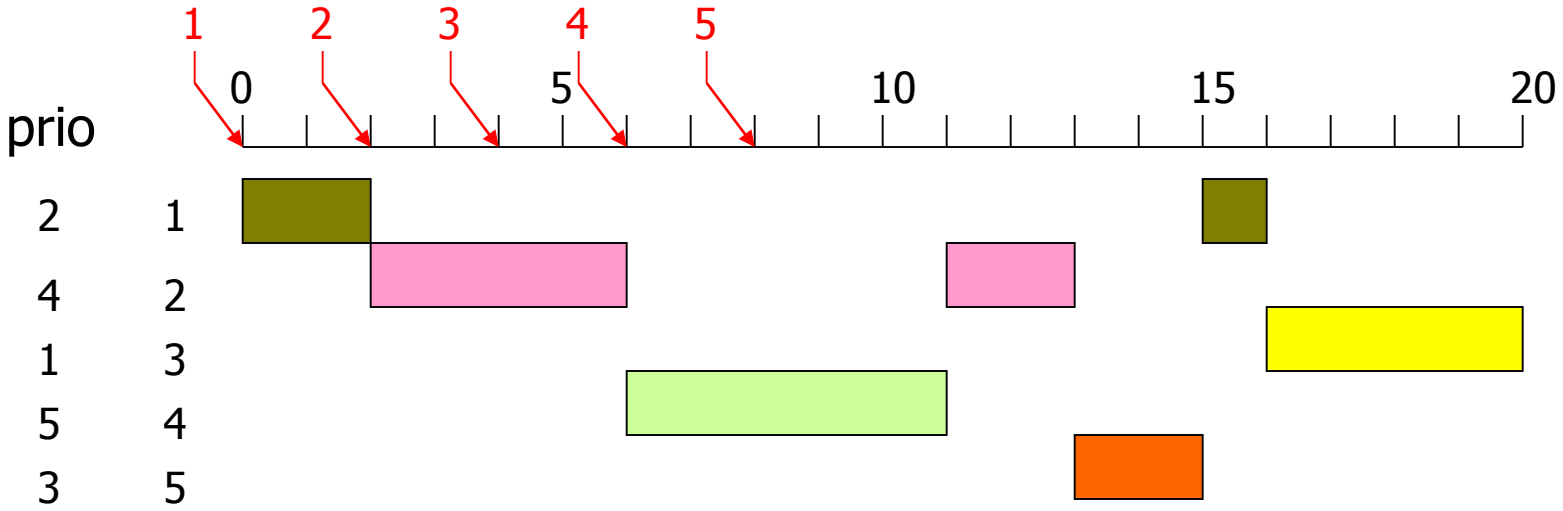
Problem: Prioritäten müssen definiert werden
im OS: Systemprozesse > User Prozesse



PRIO-P (Priorities – Preemptive)

- Wie PRIO-NP, jedoch findet Verdrängungsprüfung statt
  ⇒der rechnende Prozess wird verdrängt, wenn er eine geringere Priorität hat als der Neuankömmling

PrioQueue mit Verdrängung des laufenden Prozesses

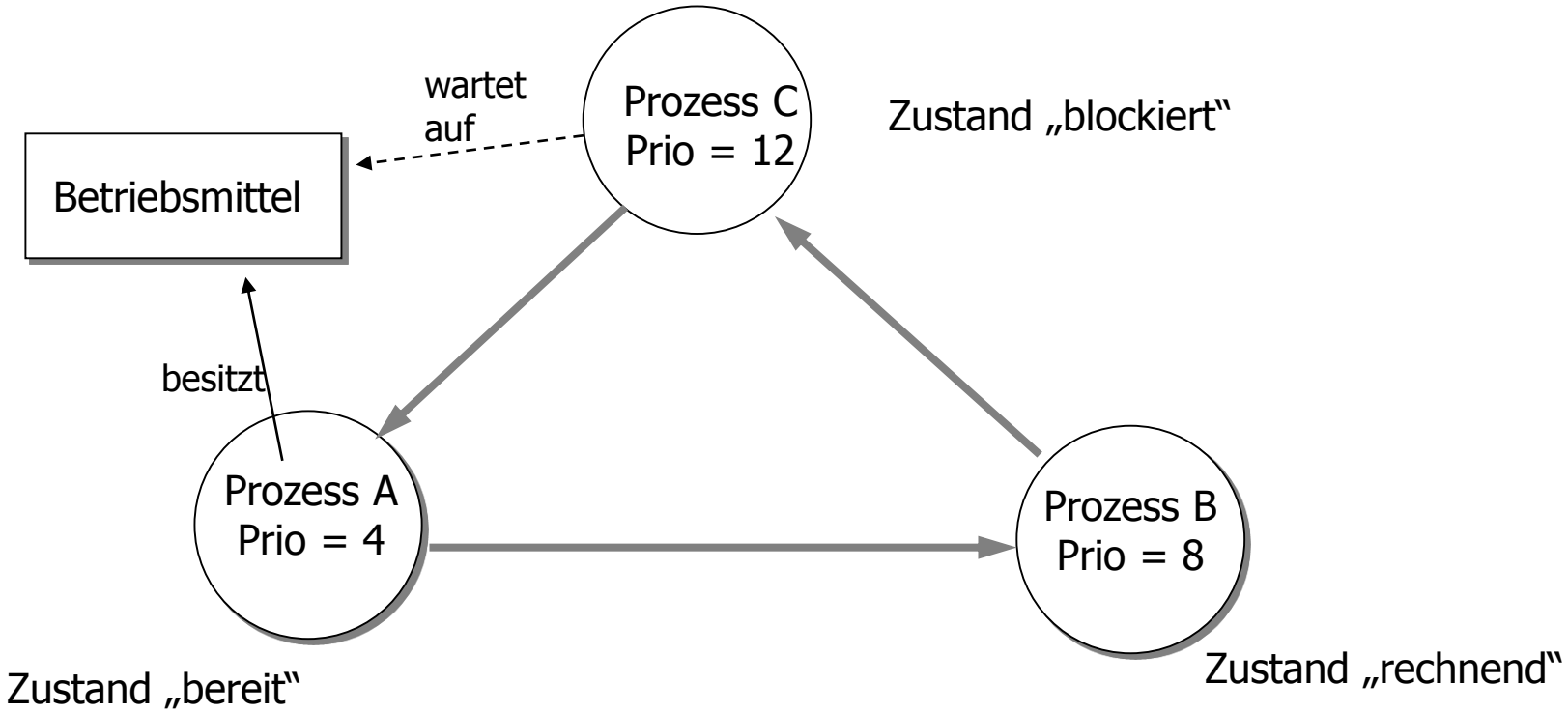


Prioritätsinvertierung

- Prozess C wird „verhungern", obwohl er der dringlichste ist
  - B blockiert die CPU, A kommt nicht dran und kann das Betriebsmittel nicht freigeben,
  - ⇒ C bleibt auf unbestimmte Zeit blockiert

Grund: Entziehen von Betriebsmittel ist gefährlich, z.B. können korrupte Daten oder Inkonsistenzen in Datenbanken entstehen
-> OS entzieht üblicherweise keine Betriebsmittel

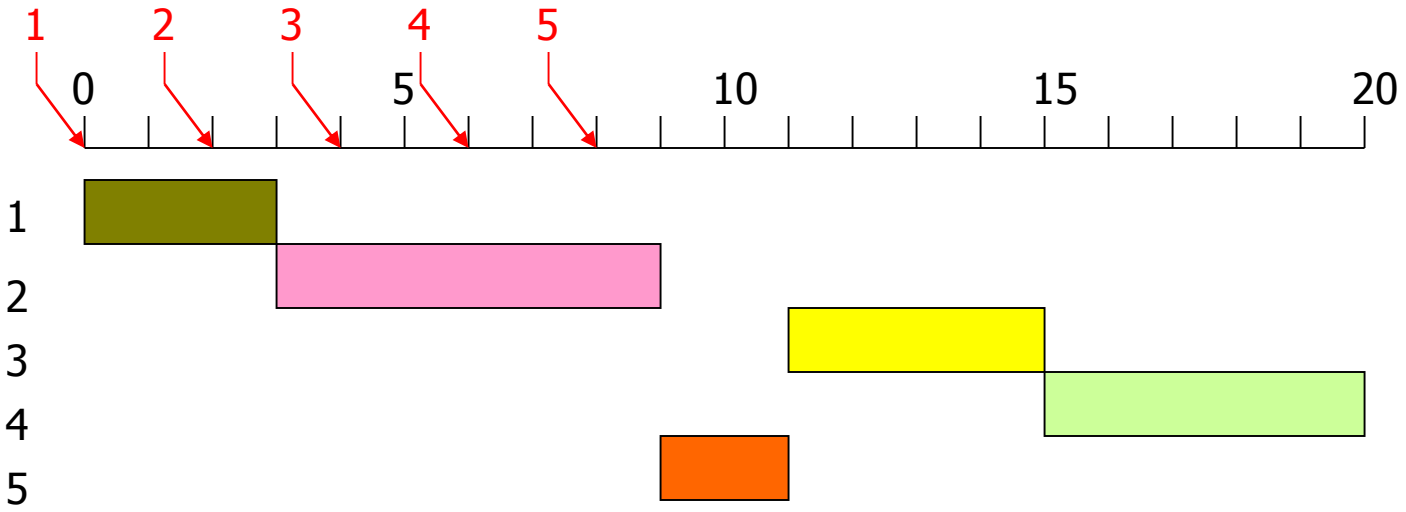
- Lösung?
  - A bekommt Priorität von C, solange A das Betriebsmittel besitzt



SJN (Shortest Job Next)

- Prozess mit kürzester Bedienzeit wird als nächster bis zum Ende oder zur freiwilligen Aufgabe bearbeitet.
  - Wie PRIO-NP mit Bedienzeit als Prioritätskriterium
  - Bevorzugt kurze Prozesse

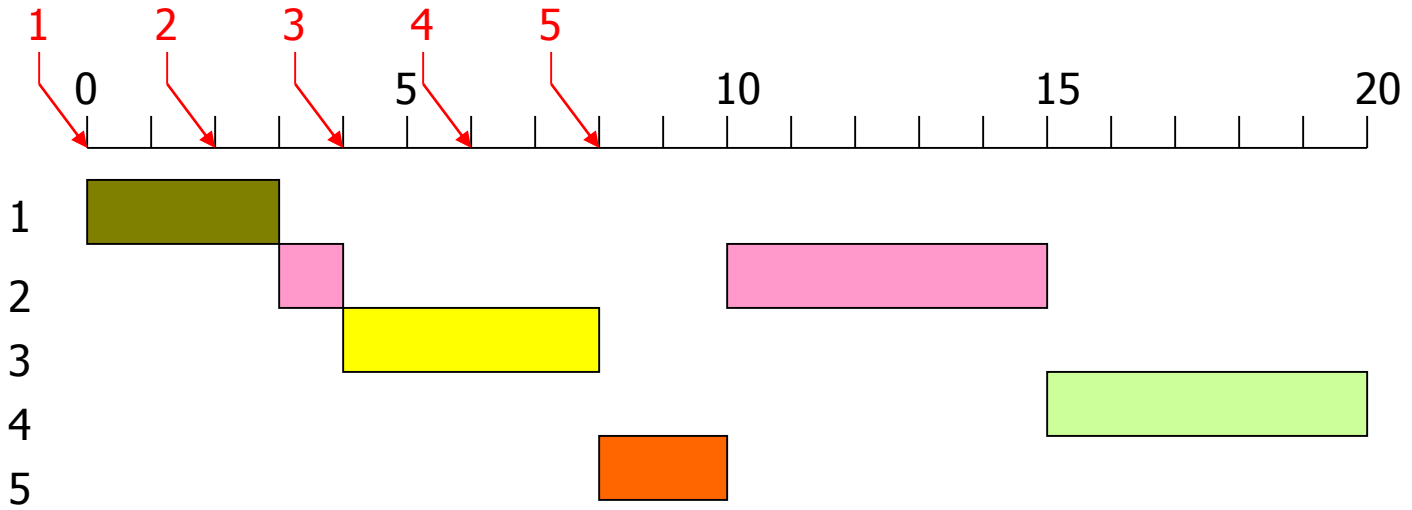
praktisch kaum relevant



SRTN (Shortest Remaining Time Next)

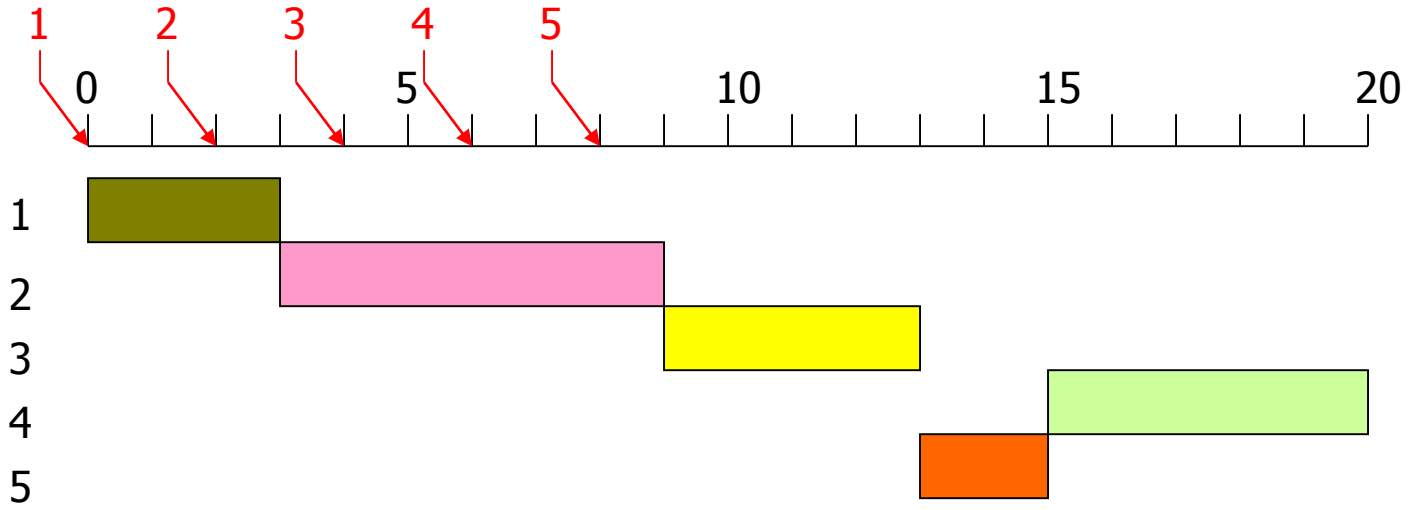
- Prozess mit kürzester Restbedienzeit wird als nächster bearbeitet
  - Rechnender Prozess kann verdrängt werden
  - Nachteil: Schätzung der Bedienzeit stammt vom Benutzer

  ⇒Längere Prozesse können „verhungern“, wenn immer kürzere vorhanden sind



HRRN (Highest Response Ratio Next)

- Arbeitsweise
  - Response Ratio rr ist definiert als $rr := \frac{\text{Wartezeit + Bedienzeit}}{\text{Bedienzeit}}$
  - rr wird dynamisch berechnet und als Priorität verwendet
    - Prozess mit größtem rr -Wert wird als nächster ausgewählt.
    - Strategie ist nicht verdrängend
    - Wie bei SJN: Bevorzugung kurzer Prozesse, aber lange Prozesse können durch Warten „Punkte sammeln".



3.3 Multilevel-Scheduling

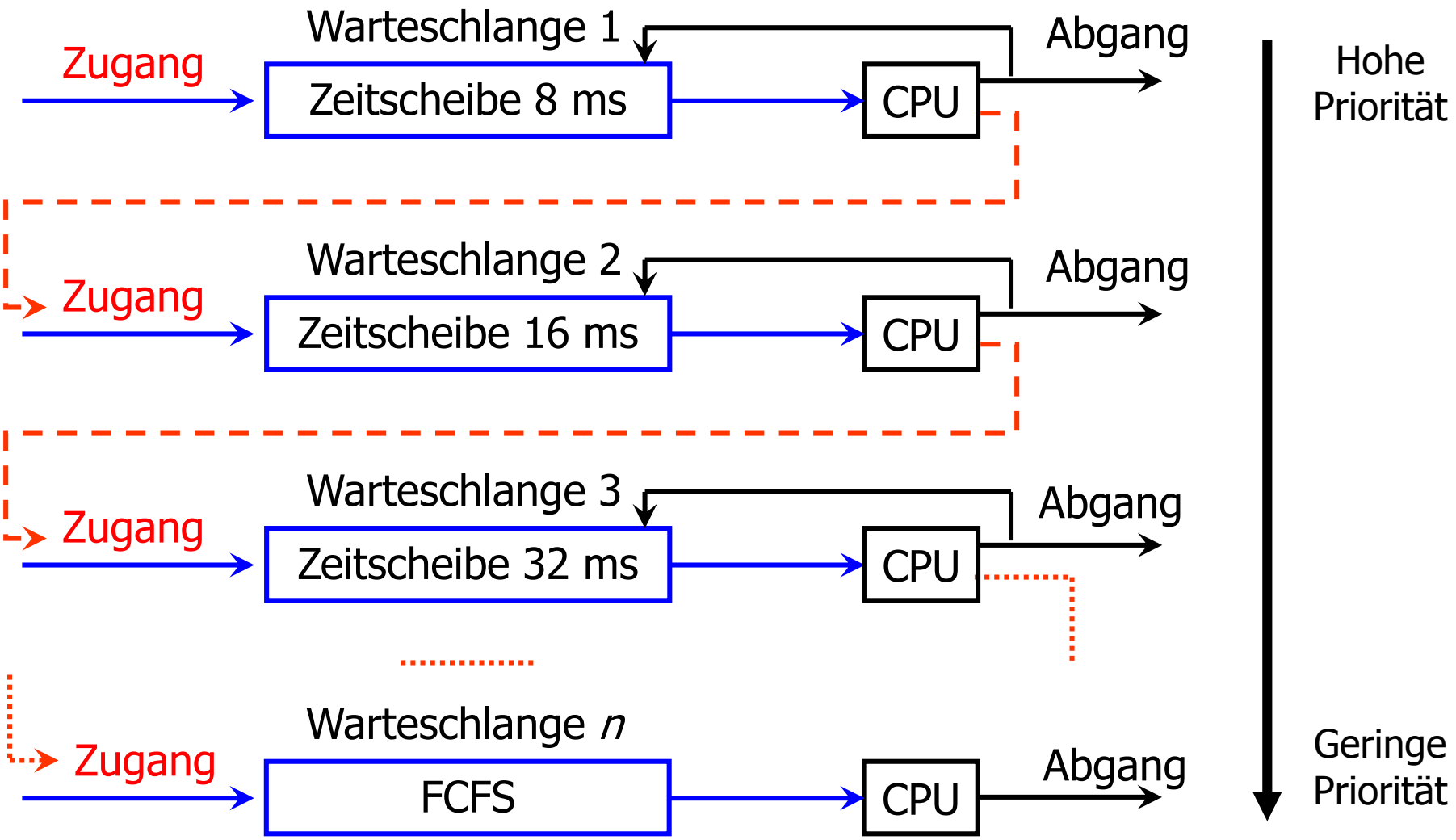
- Kombination unterschiedlicher Scheduling-Verfahren
  - Durch die Verknüpfung können Schedulingstrategien besser auf die Betriebsform (Dialogbetrieb, Stapelbetrieb, …) abgestimmt werden

- Einfachste Realisierung
  - Unterteilung der Liste bereiter Prozesse in mehrere Sublisten
  - Jede Liste kann anhand verschiedener Strategien verwaltet werden
  - Aufträge werden – je nach gewünschter Betriebsform – in die entsprechende Liste einsortiert
  - Mit zusätzlichem Auswahlverfahren wird bestimmt, welcher Prozess aus welcher Subliste zur Ausführung gebracht wird
  - Prioritätsvergabe pro Liste führt dazu, dass z.B. zeitkritsche Prozesse bevorzugt behandelt werden

Feedback-Scheduling

- Grundidee ist die Berücksichtigung von
  - „Warte"-Historie eines bereiten Prozesses
  - Periodizität bei Prozessausführung
    - ⇒Schedulingkriterien werden an den aktuellen Systemzustand angepasst
    - ⇒Beispiel: Aging bevorzugt unverhältnismäßig lang (bzgl. ihrer Ausführungszeit) wartende Prozesse
- Multilevel-Feedback-Scheduling
  - Ausführungszeit und Ausführungsverhalten der Prozesse ist nicht im Voraus bekannt
  - Anpassung an Ausführungsverhalten: Modifikation der Zeitscheibenlänge
  - Anpassung an Ausführungszeit: Stufenweise Prioritätenreduktion

Multilevel-Feedback-Scheduling



Multilevel-Feedback-Scheduling (2)

- Grundlage: RR-Verfahren mit Bereit-Liste, welche in mehrere Teillisten untergliedert wird
  - Unterschiedliche Länge der Zeitscheiben/Liste
  - Unterschiedliche Prioritäten/Liste
  - Unterste Warteschlange funktioniert nach dem FCFS-Prinzip

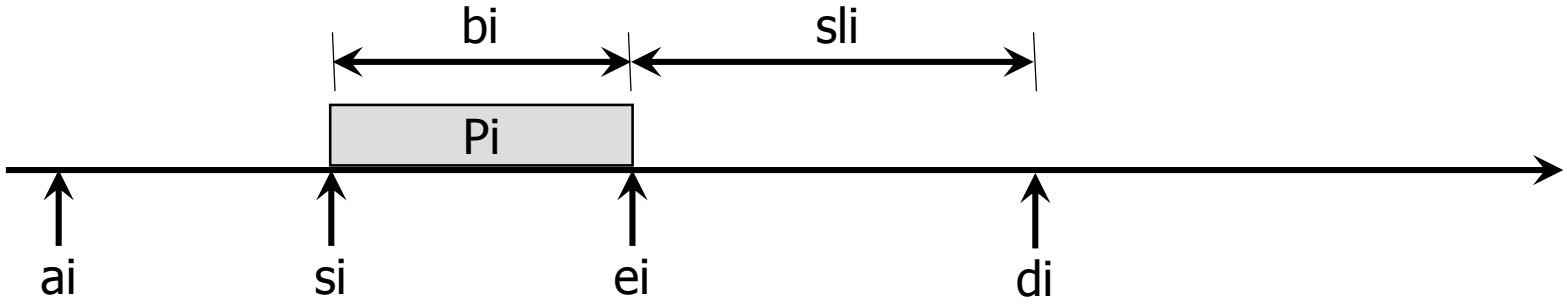
- Vorgehensweise
  - Verdrängte Prozesse (benötigen also mehr Zeit) kommen in eine Bereit-Liste mit längerer Zeitscheibe/geringerer Priorität
  - Prozesse, welche blockierende Operationen aufrufen oder den Prozessor freiwillig abgeben, verbleiben in der Warteschlange
    $\Rightarrow$ Bevorzugung E/A-intensiver Anwendungen
  - Zusätzliche Feedback-Mechanismen ermöglichen eine Hochstufung (kürzere Zeitscheibe/höhere Priorität)

3.4 Scheduling mit Sollzeitpunkten

- Sollzeitpunkte treten primär in Realzeitsystemen wie Steuerrechnern für Fertigungsstraßen, Motorsteuerung … auf
  - Vollständige Bearbeitung eines Prozesses zu einem bestimmten, a-priori festgelegten Sollzeitpunkt (*deadline*), z.B. Auswertung von Messgrößen innerhalb *knapper Zeitschranken*
  - ⇒ Einhaltung der Sollzeitpunkte teilweise kritisch für die Funktion des Gesamtsystems
- Können Verletzungen der Sollzeitpunkte toleriert werden?
  - Strikte Echtzeitsysteme (*Hard real-time systems*)
    - Verletzung wegen Systemausfall untolerierbar (z.B. Airbag, ABS, Öffnung von Ventilen beim Überdruck, …)
    - ⇒ Oft Einsatz von Off-Line-Algorithmen notwendig
  - Schwache Echtzeitsysteme (*Soft real-time systems*)
    - Verletzung zwar tolerierbar, führt aber zu Qualitätsverlusten (z.B. Internet-Telefonie, Videoübertragung im Internet)

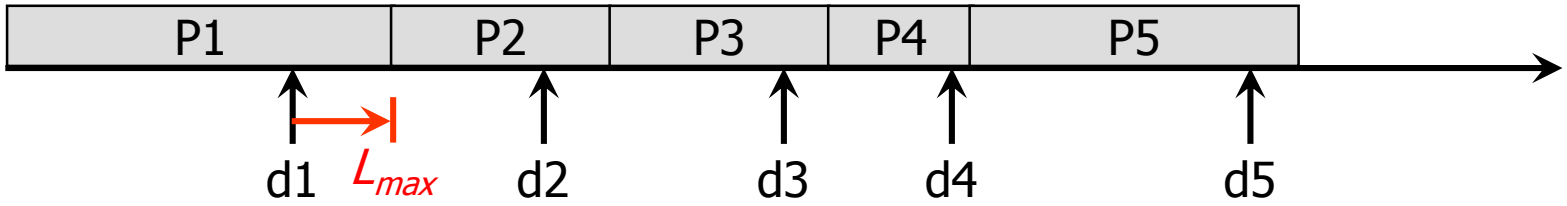
Wichtige Zeitpunkte bei Realzeitprozessen

- Voraussetzung
  - Früheste Anfangszeit und späteste Endzeit sind a-priori vorgegeben und bekannt
- Wichtige Zeitpunkte für einen Prozess Pi:
  - Frühester Startzeitpunkt ai
  - Tatsächlicher Startzeitpunkt si
  - Tatsächlicher Endzeitpunkt ei
  - Spätester Endzeitpunkt di (*Sollzeitpunkt, deadline*)
  - Bedienzeit (*service time*) bi = ei-si
  - Spielraum (*slack-time, laxity*) sli = di-si



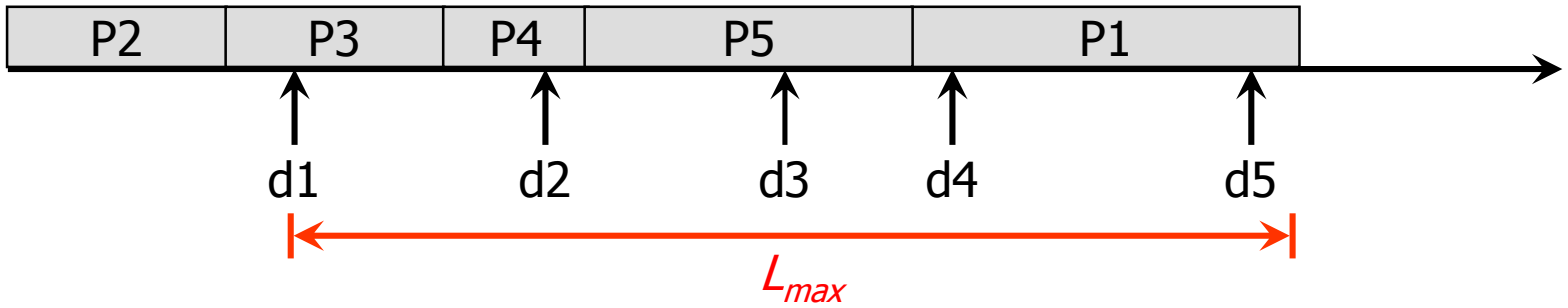
Verspätungen

- Überschreitung der Sollzeit (ei > di): *Verspätung* (*lateness*) L=ei–di
- Mögliche Zielfunktionen bei tolerierbaren Verspätungen (Soft-RTS)
  1. Minimierung der maximalen Verspätung *Lmax*



Ergebnis: Maximale Verspätung minimiert, aber alle Deadlines verpassst

  2. Einhaltung von möglichst vielen Deadlines, ohne Berücksichtigung der maximalen Verspätung



Minimierung der maximalen Verspätung

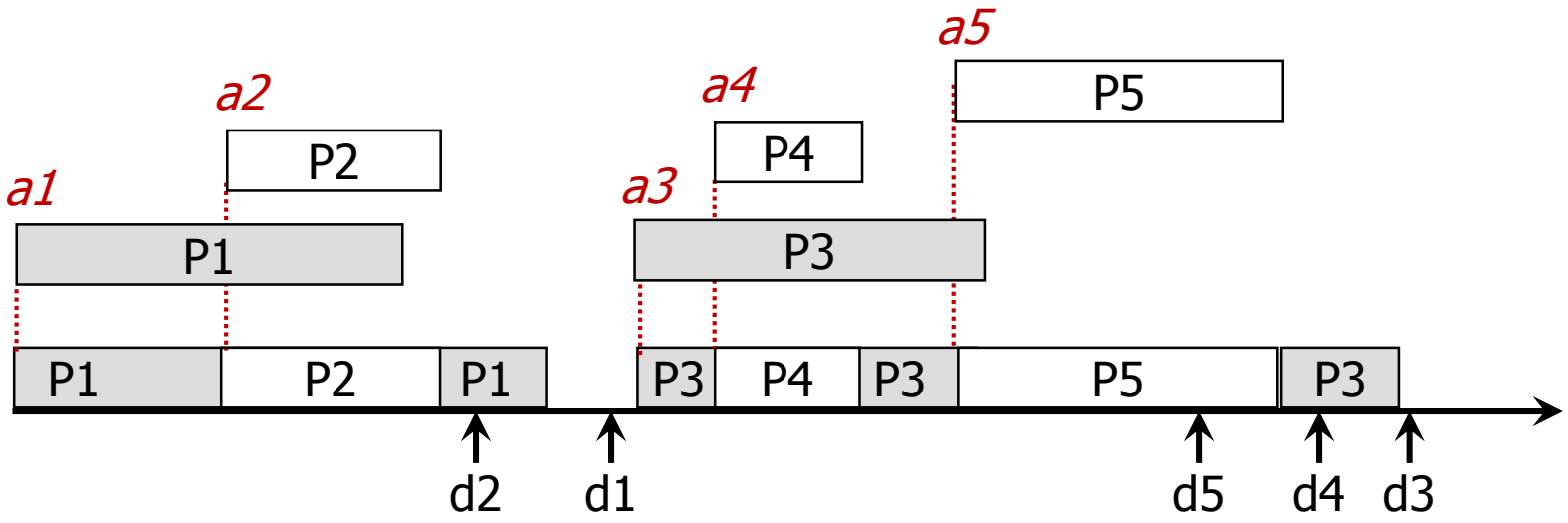
- Bei Einprozessorsystemen ohne Verdrängung und ohne Abhängigkeiten zwischen Prozessen
  - $\Rightarrow$ Scheduling durch Permutation der Prozesse

- Theorem EDD (*Earliest Due Date*), Jacksons Regel
  - Relaxation: Alle Prozesse können zu jedem Zeitpunkt beginnen
  - Jeder Ablaufplan, in dem die Prozesse nach nicht fallenden Sollzeitpunkten geordnet ausgeführt werden, ist optimal bezüglich *Lmax*
  - Lediglich Sortiervorgang mit O(nlog n) notwendig
- Praxis
  - Durch die Einführung von unterschiedlichen Startzeitpunkten ($\exists$ i,j: $a_i \neq a_j$) wird das Problem NP-schwer, d.h. es ist nicht in polynomialer Zeit optimal lösbar

Minimierung der maximalen Verspätung (2)

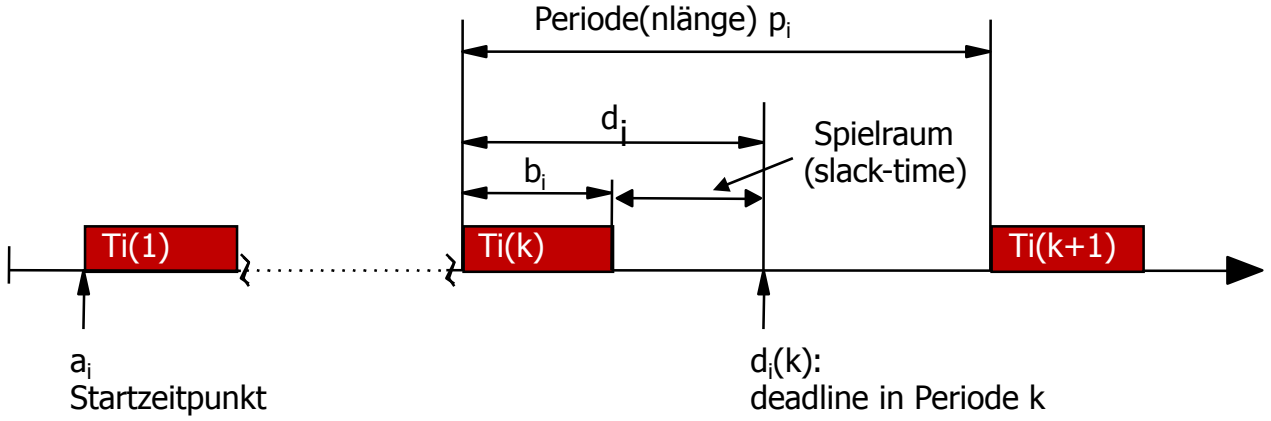
- Voraussetzung
  - Vorgegebene Startzeitpunkte
  - **Verdrängung** möglich

- Theorem EDF (*Earliest Deadline First*)
  - Jeder Ablaufplan, in dem zu jedem Zeitpunkt der Prozess mit dem frühesten Sollzeitpunkt unter allen ablaufbereiten Prozessen zugeordnet wird, ist optimal bezüglich der maximalen Verspätung



Periodische Prozesse

- Periodische Aufgaben mit Deadlines kommen zu bestimmten Zeitpunkten immer wieder
  - ⇒ Jeder Prozess ist gekennzeichnet durch **Periode** bzw. die dazu reziproke **Rate**
- Ist die Schedulingaufgabe lösbar (schedulability test, feasibility test)?
  - Für jeden periodischen Prozess muss gelten: $0 < b_i \leq d_i \leq p_i$
  - Bei mehreren periodischen Prozessen muss gelten: $\sum_i \frac{b_i}{p_i} \leq 1$



Periodische Prozesse

- *Ratenmonotones* Verfahren (rate monotonic scheduling)
  - Statische Priorität für jeden Prozess, die umgekehrt proportional ist zu der Periode
  - $\Rightarrow$ Prozess mit der kleinsten Periode hat die höchste Priorität
- Voraussetzung: Unabhängige Prozesse und Sollzeitpunkte fallen mit den Perioden zusammen
- Satz: Eine Menge von n periodischen Prozessen kann durch ein ratenmonotones Verfahren eingeplant werden, wenn folgendes gilt (hinreichende Bedingung):

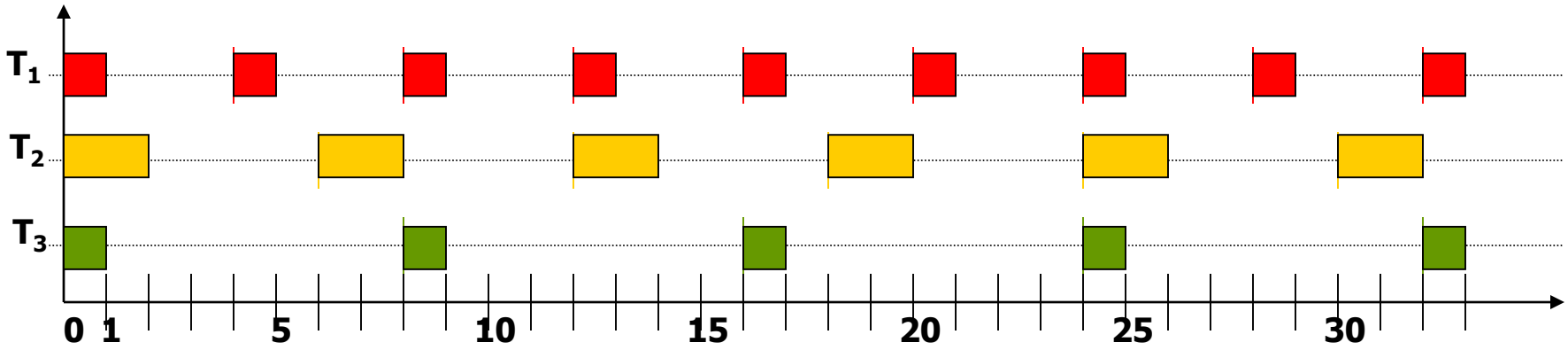
$$\sum_{i=1}^{n} \frac{b_i}{p_i} \leq n \left(2^{\frac{1}{n}} - 1 \right)$$

  - Links: benötigte Prozessorleistung
  - Rechts: obere Schranke für einen zulässigen Schedule
- Bei großen $n \Rightarrow$ CPU-Auslastung höchstens ln 2 $\approx$ 69,3 %

Rate Monotonic Scheduling

- Annahmen
  1. Prozess T_i ist periodisch mit Periodenlänge p_i
  2. Deadline ist $d_i = p_i$
  3. T_i ist unmittelbar nach p_i erneut bereit
  4. T_i hat eine konstante Bedienzeit b_i (<= p_i)
  5. Je kleiner die Periode, desto höher die Priorität

Beispiel: T ={T_1, T_2, T_3}, p={4, 6, 8}, b ={1, 2, 1}

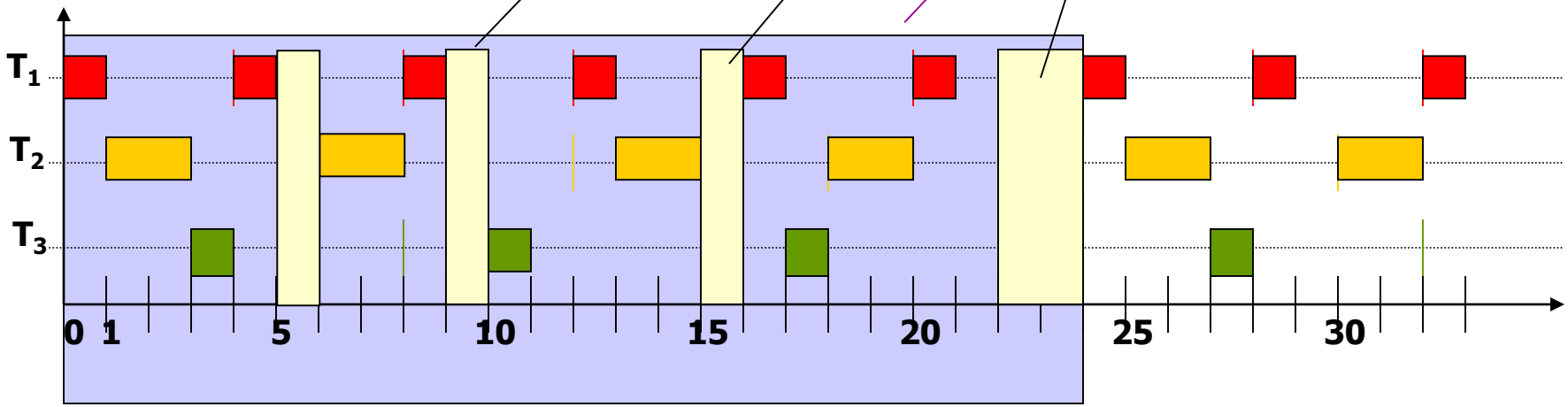


Wie sieht die Einplanung auf einem Prozessor aus?

Rate Monotonic Scheduling (RMS)

- Annahmen
  1. Prozess Ti ist periodisch mit Periodenlänge pi
  2. Deadline ist di = pi
  3. Ti ist unmittelbar nach pi erneut bereit
  4. Ti hat eine konstante Bedienzeit bi (<= pi)
  5. Je kleiner die Periode, desto höher die Priorität

Beispiel: T ={T_1, T_2, T_3}, p={4, 6, 8}, b ={1, 2, 1}



Ergebnis des RMS-Beispiels

Die allgemeine **notwendige Bedingung** ist erfüllt:

$$(1/4 + 2/6 + 1/8) = 17/24 <= 1$$

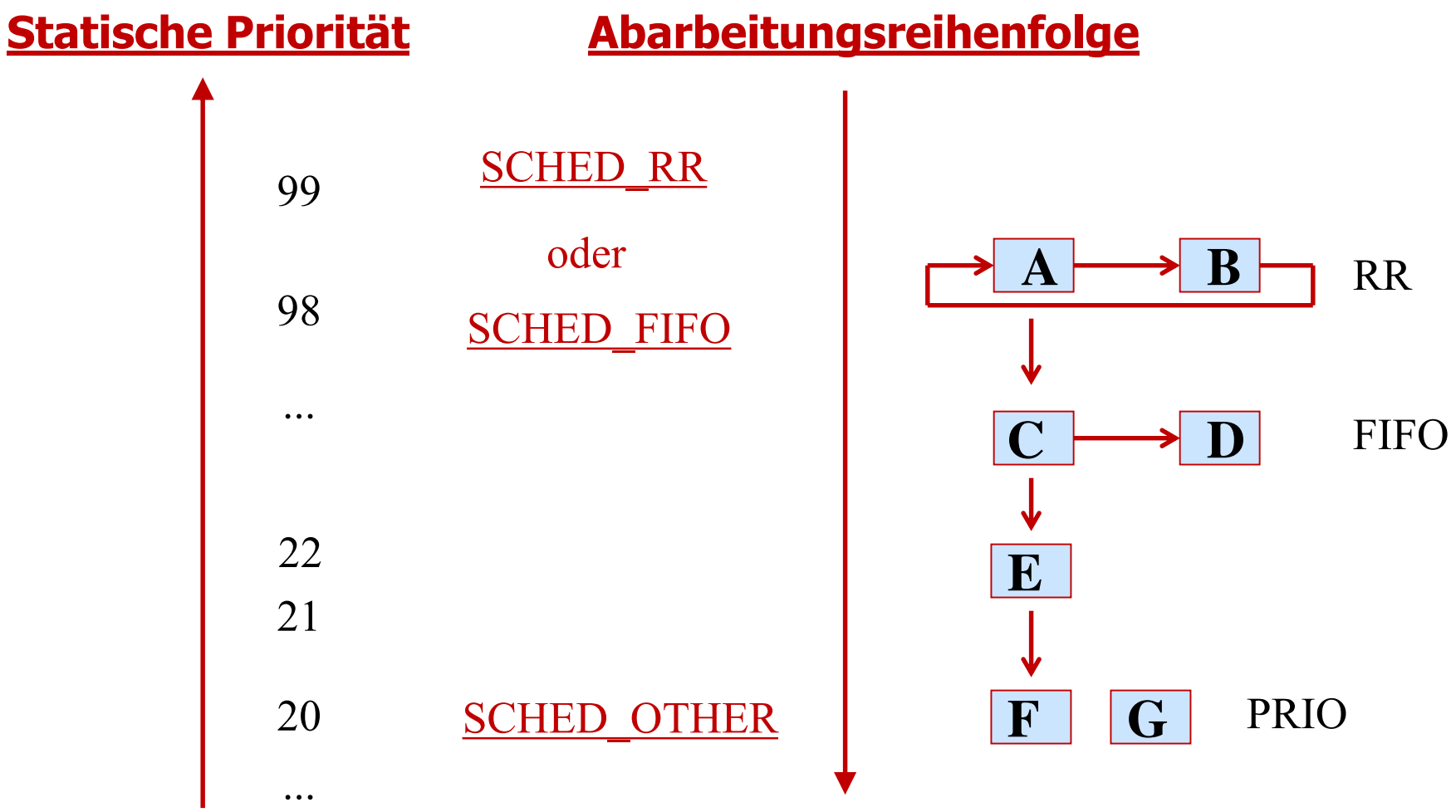
Auch das hinreichende RMS-Kriterium ist erfüllt:

$$(1/4 + 2/6 + 1/8) = 17/24 = 0{,}7083 < 3\,(\,2^{1/3}-1) \approx 78\,\%.$$

3.5 Fallbeispiel: UNIX-Scheduling

- UNIX/Linux-Scheduling basiert auf Multilevel-Feedback-Scheduling
  - Statische und dynamische Prioritäten zur Prozessauswahl
  - Verwaltung der lauffähigen Prozesse in einer Liste (runqueue)
- Scheduling-Verfahren (policies)
  - SCHED_OTHER: basiert auf PRIO-P, für „normale" Prozesse
    - Statische Priorität = 20, dynamische Priorität (nice): [-20 19]
    - Verdrängung durch Prozesse mit höherer statischer Priorität
    - Die nicht verbrauchte Zeitscheibe bleibt als „Gutachten" erhalten
  - SCHED_FIFO: FCFS, für „Echtzeitprozesse" (Real-Time FIFO)
    - Statische Priorität: 1-99, bei gleicher Priorität Wahl des ersten Prozesses
    - Verdrängung durch Prozesse mit höherer statischer Priorität
  - SCHED_RR: Round-Robin, für „Echtzeitprozesse" (Real-Time RR)
    - Statische Priorität: 1-99, bei gleicher statischer Priorität $\Rightarrow$ FIFO
    - Nach Ablauf der Zeitscheibe $\Rightarrow$ Einordnung ans Ende der runqueue
    - Verdrängung durch Prozesse mit höherer statischer Priorität

Multilevel-Scheduling von Prozessen



Funktionsweise des Schedulers

- Scheduling-Algorithmus in Linux
  - Entferne alle Prozesse aus der Wartschlange runqueue, deren Zustand nicht TASK_RUNNING ist, d.h. die nicht ablaufbereit sind
  - Bewerte jeden lauffähigen Prozess aus der runqueue und wähle den Prozess mit der höchsten Bewertung
  - Wenn alle Zeitkonten (Quantum) der lauffähigen Prozesse (blockierte werden nicht berücksichtigt) abgelaufen sind, dann berechne die Zeitkonten aller Prozesse neu
    - Quantum: Bestimmte Anzahl sog. Uhrticks, üblicherweise 20, wobei ein Uhrtick etwa 10ms beträgt
  - Bestimme den auszuführenden Prozess und rufe Dispatcher auf
- CPU wird entzogen, wenn
  - Quantum vollständig verbraucht (Quantum=0)
  - Thread blockiert wegen E/A
  - Thread höherer Priorität ist ablaufbereit

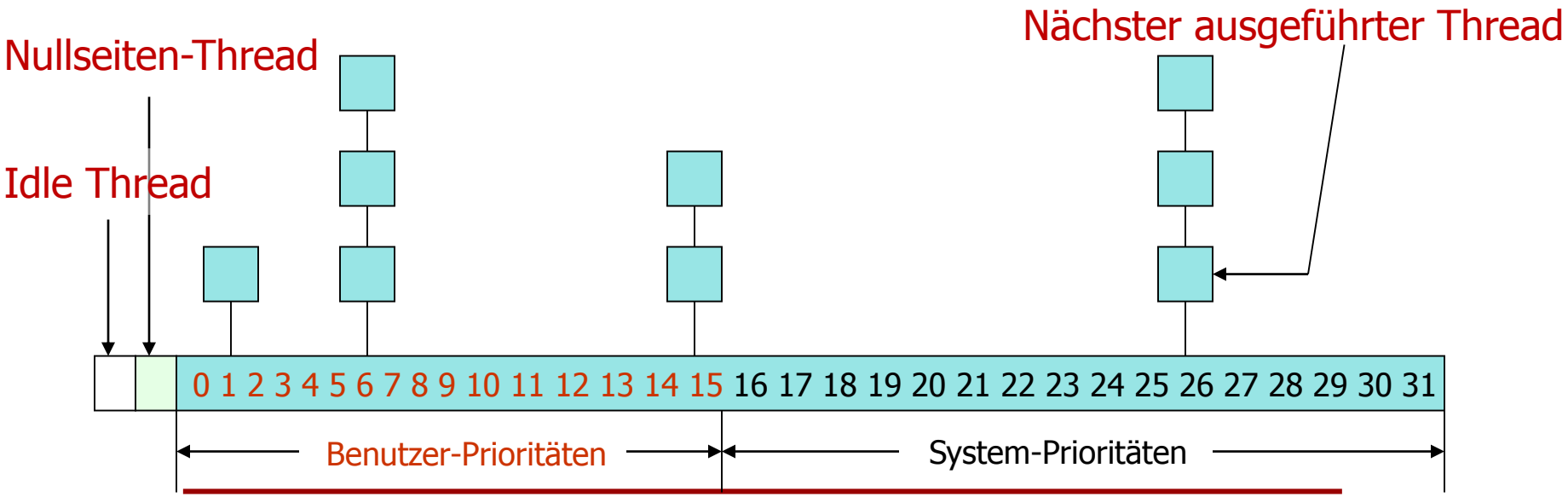
Prozessbewertung

- Bewertung eines Prozesses durch die Funktion goodness(...)
  1. Rückgabe –1: Freiwillige Abgabe der CPU
  2. Rückgabe 0: Quantum aufgebraucht
  3. Rückgabe [1-1000]: Normaler Prozess
  4. Rückgabe > 1000: Echtzeitprozess

  ⇒ Je größer der Rückgabewert, desto besser ist die Bewertung
- Berechnung der Funktion goodness() im Fall 3 und 4
  - Bei normalen Prozessen (Fall 3)
    - Allgemein: Güte = quantum + priorität
    - Gleicher Prozess hat noch Zeit übrig:
      Güte = quantum + priorität + 1
    - Gleicher Adressraum wie aktueller Prozess:
      Güte = quantum + priorität + 1
  - Bei Echtzeit-Prozessen (Fall 4): Güte = 1000 + Priorität
- Neuberechnung: Quantum_neu = Quantum_alt/2+Basispriorität

Windows: Arbeitsweise des Scheduling-Algorithmus

- Verwaltung der Prioritätsliste mit 32 Prioritäten
  - Jeder Eintrag enthält Liste mit allen wartenden Threads der gleichen Priorität
  - Durchlaufen der Liste von Priorität 31 bis Priorität 0
  - Bei nichtleerem Eintrag ⇒ führe die Prozesse nach Round-Robin aus
- Spezielle Threads
  - Null Thread: läuft im Hintergrund, überschreibt Speicherseiten mit Nullen
  - Idle Thread: Läuft, wenn keine anderen Threads inkl. Null Thread aktiv



Zeitscheibenlänge

- Zeitscheibenlängen
  - Standardeinstellung: 20ms, Einzelprozessor-Server: 120ms
  - Multiprozessor-Systemen: abhängig von Taktfrequenz
  - Einstellungen können um Faktor 2, 4 oder 6 erhöht werden
    - Kürzere Zeitscheiben bevorzugen interaktive Benutzer
    - Längere Zeitscheiben erfordern weniger Kontextwechsel und sind effizienter

- Verringerung der Priorität
  - Verbraucht ein Thread seine gesamte nächste Zeitscheibe, so wird seine Priorität um eine Einheit verringert, solange die aktuelle Priorität höher ist als seine Basispriorität (*Variante von Aging*)

Scheduling in Windows

- Kombination aus Prozess- und Threadpriorität ergibt 32 Werte [0, 31]
  - Systemprioritäten 16..31 ⇒ Zuweisung durch Administrator
  - Benutzerprioritäten 0..15 (Änderung mit API `SetThreadPriority`)

		Win32-Prozessklassen-Prioritäten					
		Echtzeit	Hoch	Über normal	Normal	Unter normal	Idle
Win32-Thread-Prioritäten	Zeitkritisch	31	15	15	15	15	15
	Höchste	26	15	12	10	8	6
	Über normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Unter normal	23	12	9	7	5	3
	Niedrigste	22	11	8	6	4	2
	Idle	16	1	1	1	1	1