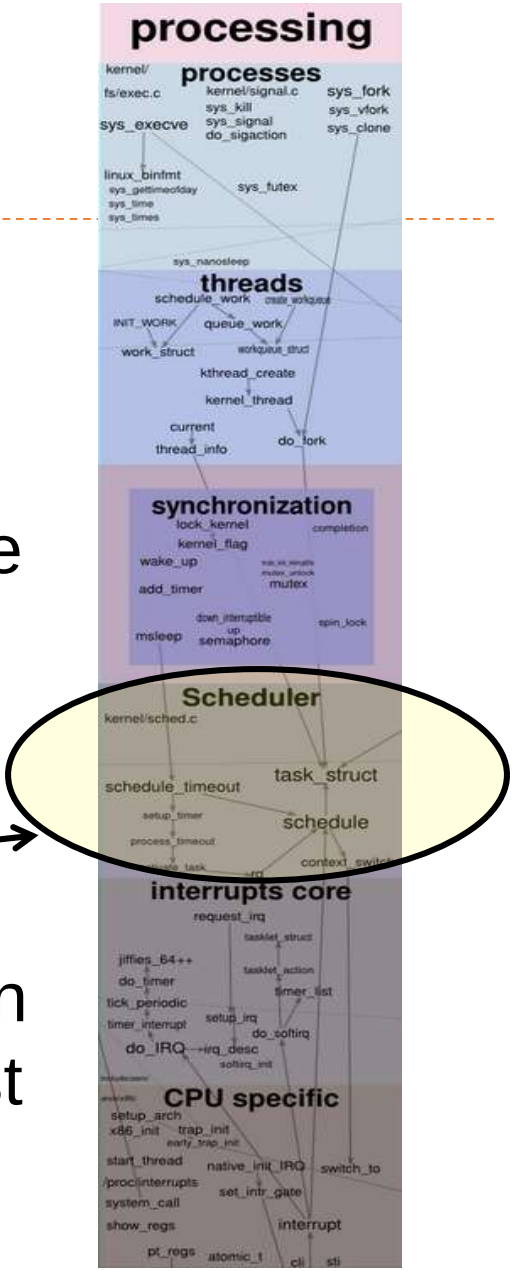
Betriebssysteme und Netzwerke Vorlesung 16

Artur Andrzejak

Scheduling von Prozessen - Grundlagen

Was ist Scheduling

- **Scheduling** (**S**) = Ablaufsteuerung, Zuteilung
- Auf einem multiprogrammierbaren System konkurrieren mehrere Prozesse oder Threads zur selben Zeit um CPU
- Der Teil des BS, der entscheidet, welcher Prozess (und wann) rechnen darf, heißt **Scheduler**
- Die Strategie, nach der er den nächsten zu rechnenden Prozess bestimmt, heißt **Scheduling-Strategie**
  - Implementiert von einem Scheduler



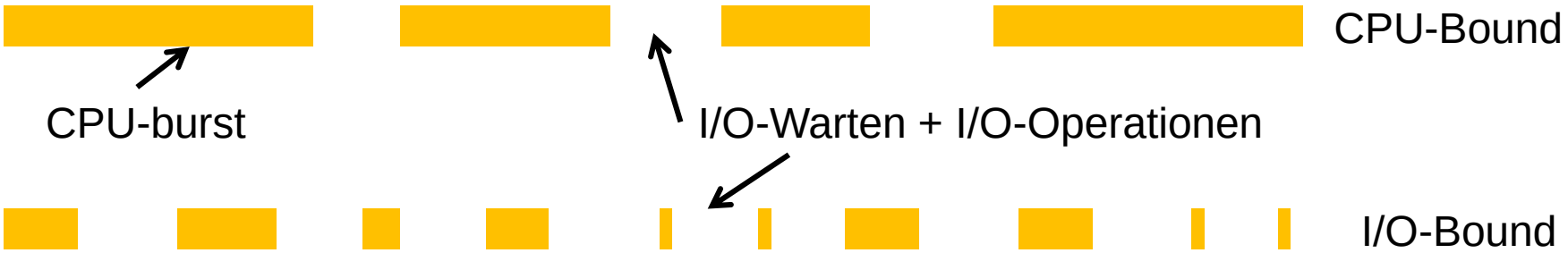
linux kernel map
(Teil: processing)

Prozessverhalten

- Die Effizienz der S.-Verfahren hängt u.a. davon ab, ob das Verhalten der Prozesse berücksichtigt wurde
- Ein Prozess wechselt zwischen einer Berechnungsphase (**CPU burst**) und I/O-Phase des I/O-Wartens plus I/O-Operationen (**I/O burst**)
  - Diese Phasen wechseln sich ab
- Die Länge der Berechnungsphase ist bei Prozessen unterschiedlich
  - Aber kurze B-phasen überwiegen stark – Länge hat meist Exponentialverteilung oder verwandte Verteilung
- Die Länge der I/O-Phase ist oft gleich - warum?

Prozessverhalten /2

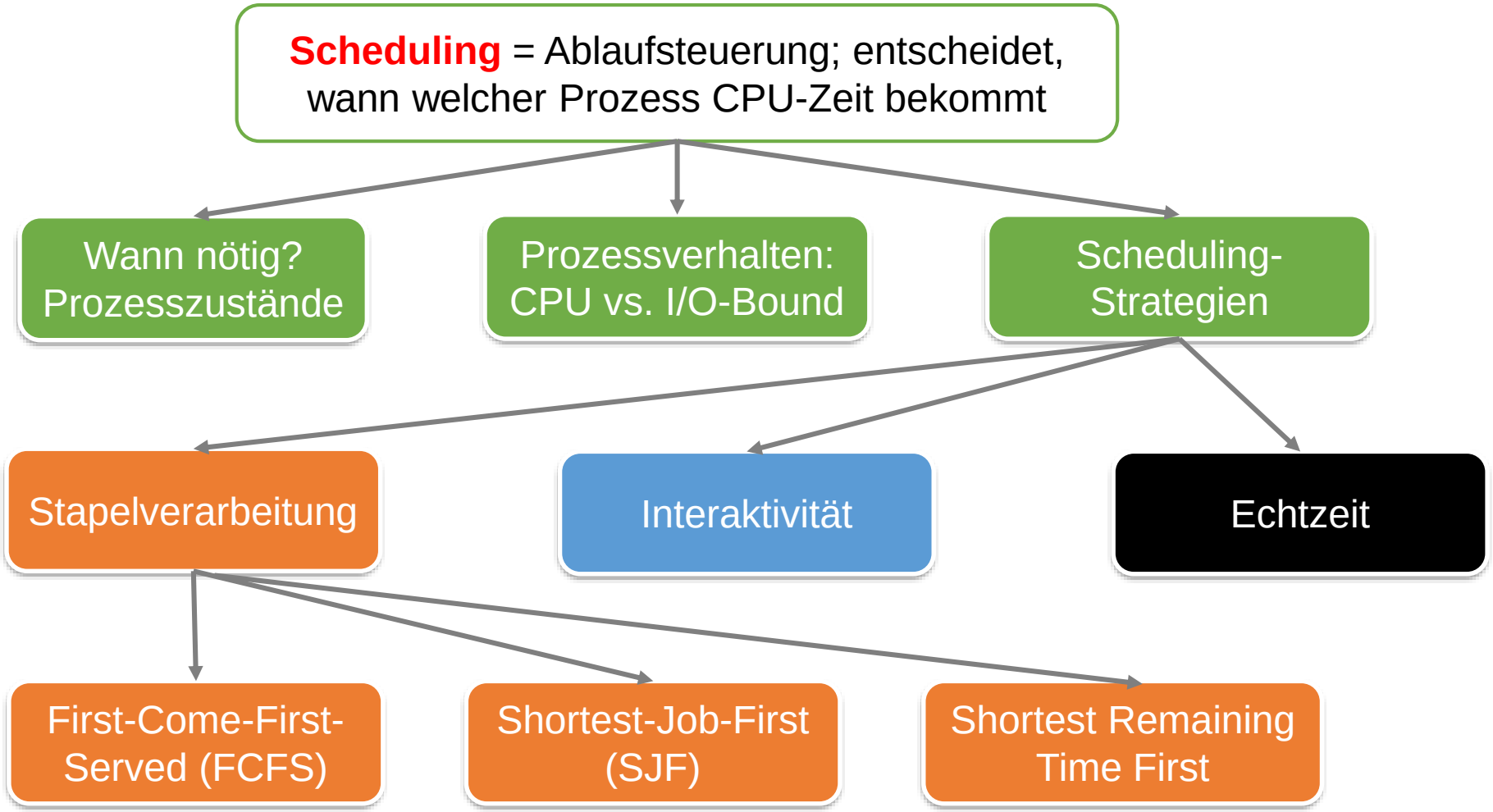
- Abhängig vom Verhältnis der Längen der beiden Phasen werden Prozesse bezeichnet als …
  - **Rechenintensiv** (**compute-bound**) bzw. **CPU-intensiv** (**CPU-bound**) – sie haben lange CPU-Burst-Phasen
  - … Oder **I/O-intensiv** (**I/O-bound**)
    - Diese müssen i.A. nicht länger auf I/O-warten, sondern haben einfach kürzere CPU-Burst-Phasen



Kategorien von Scheduling-Strategien

- Wahl der S.-Strategie ist abhängig von der Art der Systemnutzung (und dem Verhalten der Prozesse)
- Drei Typen von Umgebungen
  - **Stapelverarbeitung**
    - Unternehmen: Buchhaltung, Datenanalysen
    - Wissenschaftliches Rechnen, z.B. Simulationen
  - **Interaktivität**
    - Unsere Laptops / Desktops
  - **Echtzeit**
    - Maschinen- und Anlagensteuerung
    - Mobile Geräte wie Navigation, Telefon
    - Eingebettete Systeme, z.B. in Autos, FritzBox, Waschmaschine

Übersicht Scheduling



Scheduling in Stapelverarbeitungssystemen

First-Come-First-Served (FCFS)

- Einfachste Lösung: Der als erster anfragende Prozess wird als erster bis zum Ende ausgeführt
  - Implementierung über eine First-In-First-Out Schlange
- Nachteile?
  - Lange **durchschnittliche Wartezeit**
    - Berechnungszeiten der Prozesse: P1 = 24 ms, P2 = P3 = 3 ms

∅ = (0+24+27)/3 =17 ms

∅ = (6+0+3)/3 =3 ms

  - **Convoy effect**: Viele I/O-intensive Prozesse werden durch einen einzigen CPU-intensiven Prozess verzögert

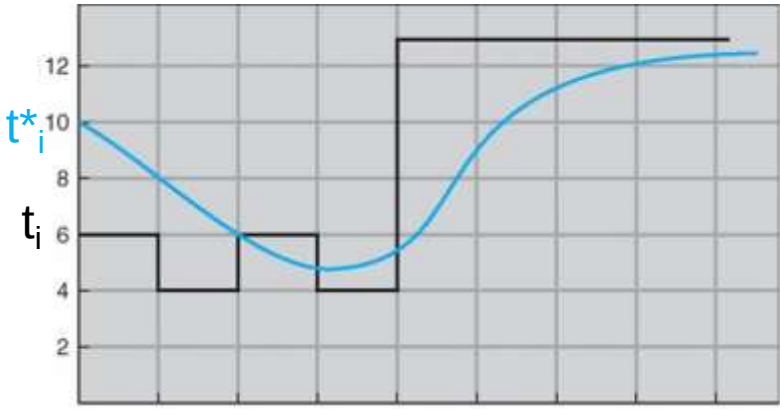
Shortest-Job-First (SJF)

- Annahme: Wir kennen die Länge des nächsten CPU-Bursts eines jeden Prozesses
- Der Scheduler wählt den Prozess mit der kleinsten (angenommenen) Berechnungszeit
- Beispiel (Berechnungslängen in Klammern):
  - P1 (6 (ms)), P2 (8), P3 (7), P4 (3)
  - Schedule (Belegungsplan)? Durchschnittliche Wartezeit?

∅ = (3+16+9+0)/4 = 7 ms

Shortest-Job-First /2

- Das größte Problem von SJF ist die mangelnde Kenntnis der Länge der CPU-Burst-Zeit (BZL)
- Man kann aber die BZL aus vorherigen Verhalten abschätzen - viele Verfahren sind bekannt (Link)
- Z.B. **exponentielle gleitende Durchschnitte** (**exponential moving average**, **EMA**)
  - t_i = Länge des i-ten CPU-Bursts
  - t^*_i = EMA_i = Abschätzung von t_i
  - Dann $t^*_{i+1} = \alpha t_i + (1-\alpha) t^*_i$
  - α ist ein Parameter, $0 \leq \alpha \leq 1$
  - Was passiert, wenn α kleiner bzw. größer wird?

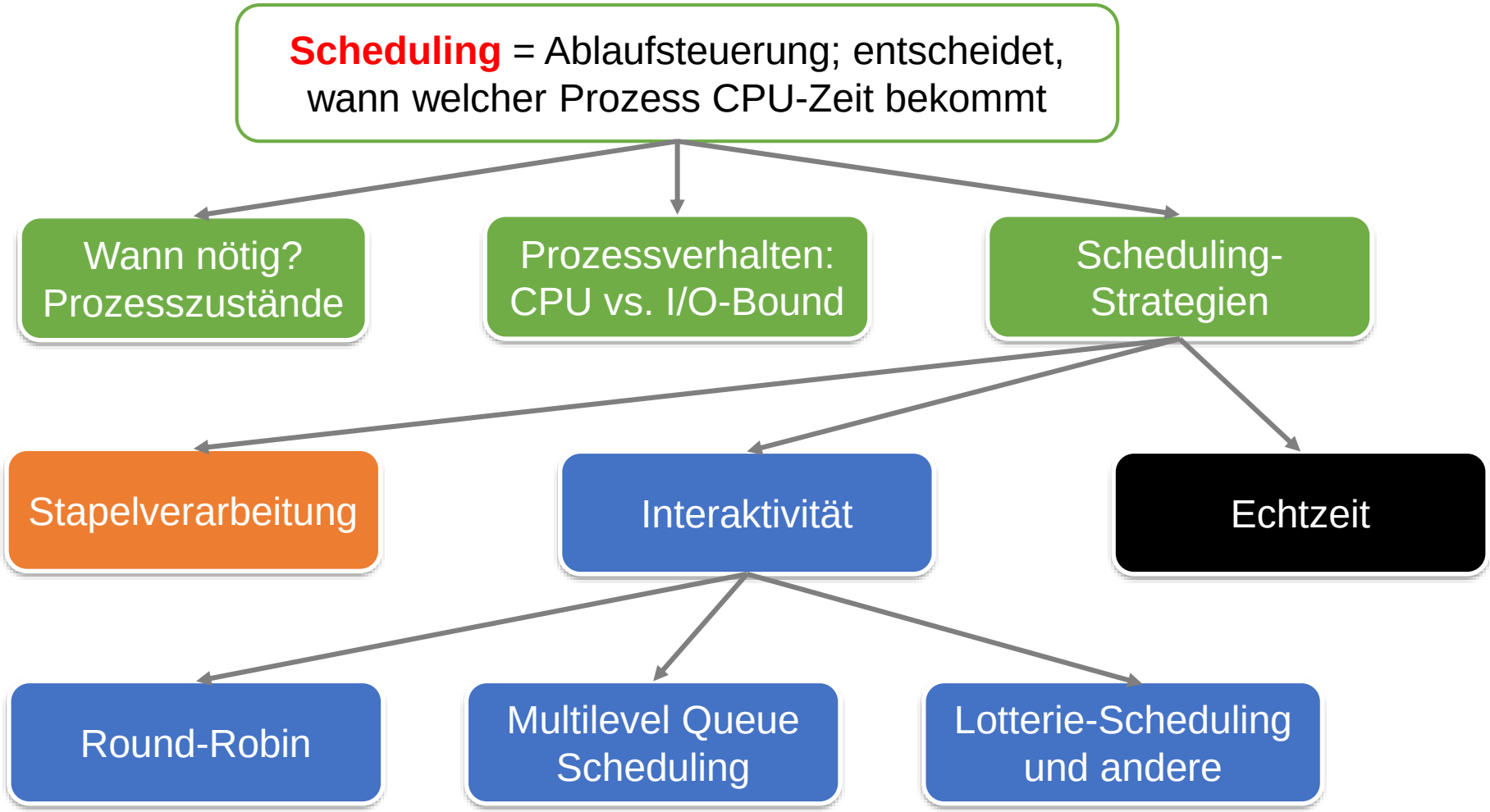


Shortest Remaining Time First

- Eine Variation von SJF, bei der die **Unterbrechbarkeit** angenommen wird
- Sei P ein (neuer) Prozess im Zustand *Ready*, und seine erwartete CPU-Burst-Zeit sei $\mathbf{t_b}$
- Die Zeit $\mathbf{t_b}$ wird verglichen mit:
  - Der (erwarteten) CPU-Burst-Zeit von jedem anderen Prozess im Zustand Ready und ..
  - (NEU) der restlichen CPU-Burst-Zeit $\mathbf{t_r}$ des laufenden Prozesses
- Ist $\mathbf{t_b}$ die kleinere Zeit als $\mathbf{t_r}$, wird der aktuelle Prozess unterbrochen und P ausgeführt
- Nachteile?
- Prozesse müssen unterbrechbar sein

Scheduling in Interaktiven Systemen

Übersicht Scheduling



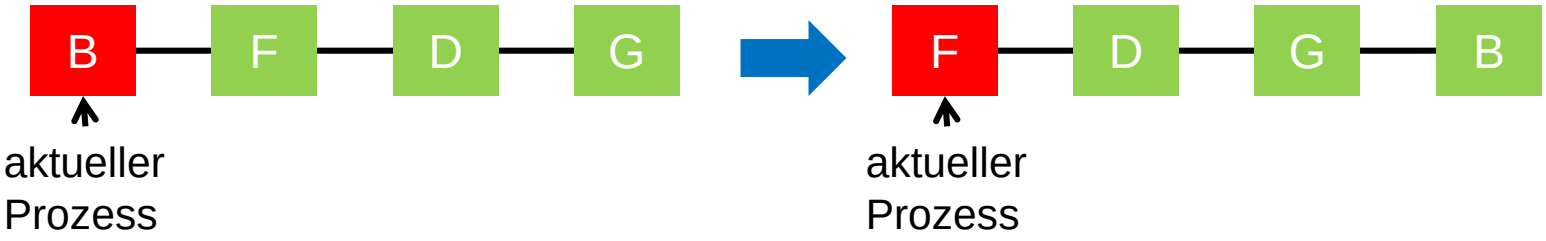
Round-Robin-Scheduling

1. Jedem Prozess wird eine maximale Zeitspanne zugewiesen, in dem er am Stück ausführen darf - sein **Quantum**
2. Prozess, der gerade ausgeführt hat, kommt ans Ende einer Warteschlange (FIFO)
3. Wenn ein Prozess am Ende seines Quantums immer noch läuft, wird die CPU unterbrochen und an den nächsten Prozess in der Schlange abgegeben
   - Falls CPU-Burst kürzer als Quantum ist, wird die Kontrolle über CPU an den nächsten Prozess abgeben

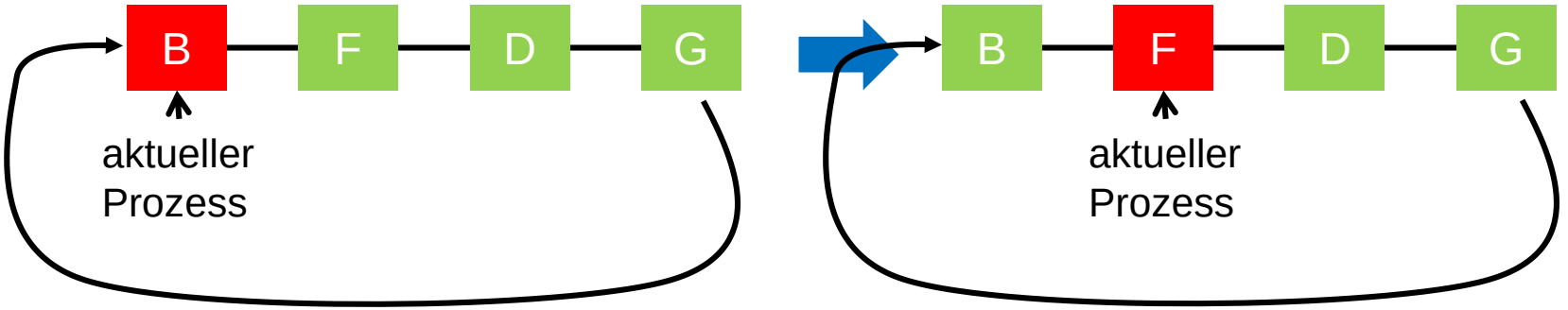
- Die Unterbrechbarkeit ist notwendig

RR: Implementierung der Prozess-Verwaltung

- Eine Liste; Endet der aktuelle Prozess, wird sein PCB vom Anfang an das Ende der Liste verschoben



- Einfacher: PCB werden in eine zyklische Liste angeordnet; nur der Zeiger zum akt. Prozess wandert



Round-Robin-Scheduling

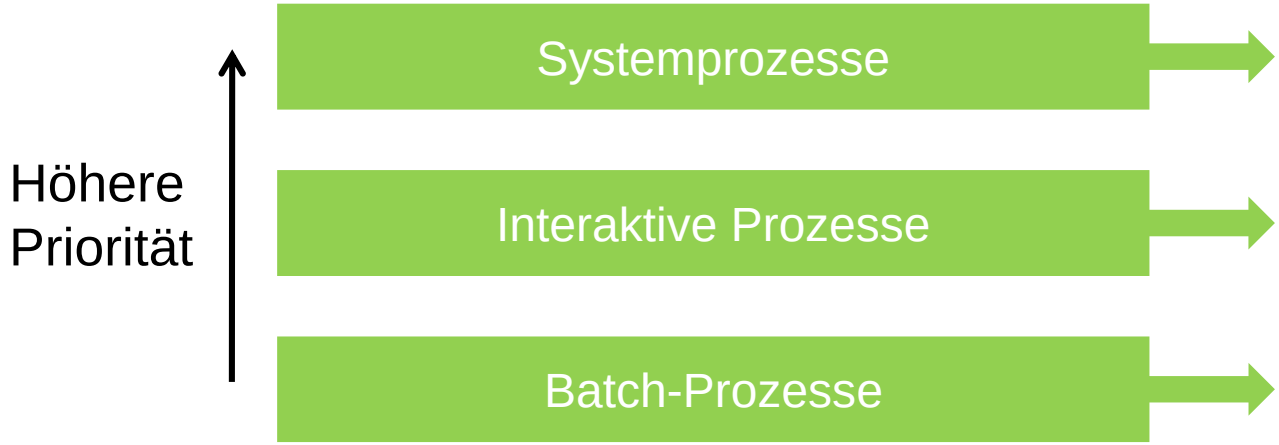
- Das einzige, was wir verändern können, ist die Länge des Quantums
- Welche Vorteile / Nachteile hat ein kurzes Quantum?
  - Vorteile: Hohe Reaktionszeit
  - Nachteile: Der Aufwand des Prozess-Wechsels (**context switch**) nimmt hohen Anteil an der CPU Nutzung an
- Moderne BS haben ein Quantum von 10 bis 100 ms
  - Die Zeit des context switch ist weniger als 10 Mikrosek.
  - => Die CPU-Effizienz ist ausreichend hoch
- Wenn das Quantum sehr lang wird, zu welcher bekannt Scheduling-Strategie degeneriert R-R?

First-come-first-served

Mehrere Warteschlangen

- Bei **multilevel queue scheduling** können Prozesse in mehrere Gruppen unterteilt werden, z.B.
  - Gruppe A: interaktive Prozesse
  - Gruppe B: Hintergrund (batch) Prozesse
- Pro Gruppe: eigene Warteschlange und Scheduler
  - Z.B. Gruppe A: Round-Robin; Gruppe B: FCFS
- Was fehlt hier, damit das funktioniert?
- Wir brauchen Scheduling zwischen den Schlagen
  - Meist mit Prioritäten
  - Z.B. nach dem Prinzip: eine Schlange niedriger Priorität kann nur ausführen, wenn alle Schlangen mit höherer Priorität leer sind (d.h. Prozesse blockiert / fertig)

Mehrere Warteschlangen /2



- Alternative Schedulingstrategie zwischen den Schlangen (hier: ohne Prioritäten):
  - Systemprozesse bekommen 10% der CPU-Zeit
  - Interaktive Prozesse bekommen 70% der CPU-Zeit
  - Batch-Prozesse bekommen 20% der CPU-Zeit
- Problem hier?

Mehrere Warteschlangen - Adaptiv

- Ein mehr flexibler Ansatz ist **multilevel feedback queue scheduling**
  - Wie die letzte Strategie, aber hier können die Prozesse zwischen den Schlangen wechseln
  - Z.B. Wenn ein Prozess zu viel CPU braucht, kommt er in eine Schlange mit niedriger Priorität
- Bsp.: 3 Schlangen, 2 oberste mit R-R, letzte FCFS
  - Nr. 1: hat Quantum von 8 ms; wenn ein Prozess in dieser Zeit fertig wird, bleibt er hier, sonst „rutscht“ er in Nr. 2
  - Nr. 2: hat Quantum von 16 ms; wird er hier nicht fertig, wird in Schlange Nr. 3 (mit FCFS) verschoben
- Dieser Ansatz ist sehr allgemein und deckt durch die **Parametrisierung** viele Szenarien ab

Lotterie-Scheduling

- Jeder Prozess bekommt **Lotterielose** für verschiede Systemressourcen wie z.B. CPU
  - Wenn eine Zuteilung passieren sollte, wird zufällig ein Los gezogen, und der Besitzer (Prozess) darf sie nutzten
  - Bei CPU-Scheduling: Lotterie wird 50 Mal / Sek. abgehalten, jeder Gewinner darf 20 ms rechnen
- Interessante Eigenschaften
  - Kooperierende Prozesse können ihre Lose tauschen
    - Z.B. Client-Prozess stellt eine Anfrage an einen Server, und schickt im seine Lose, um die Abarbeitung zu beschleunigen
  - Man kann leicht „Fein-Tuning“ des Zeitzuteilung betreiben, z.B. Rechenzeit im Verhältnis 10:20:25 usw.

Weitere Aspekte des Schedulings

Scheduling in Echtzeitsystemen (EZS)

- In **Echtzeitsystemen** spielt die Zeit die Hauptrolle
  - Eine verspätete Reaktion (z.B. Prozessausführung) ist ggf. genauso schlecht wie gar keine Antwort
  - **Harte EZS**: es gibt absolute Deadlines, z.B. beim Auto
  - **Weiche EZS**: es ist unerwünscht, aber tolerierbar, eine Deadline nicht einzuhalten
- Implementierung: das Programm wird in mehrere Prozesse unterteilt, mit jeweils vorhersagbarem Verhalten
  - Dadurch ist ggf. möglich, **statisches Scheduling** zu betreiben, d.h. vor dem Start des Programms die Scheduling-Entscheidungen zu planen

Evaluierung von Scheduling-Strategien

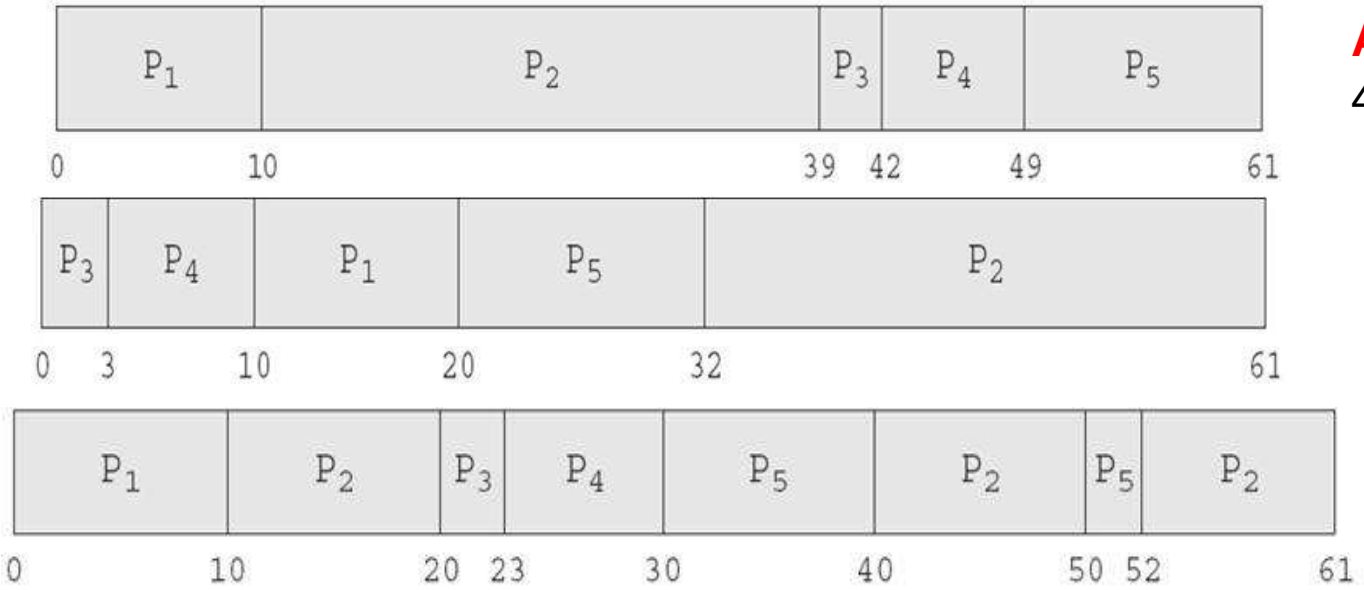
- Es gibt viele Strategien und viele Verwendungs-szenarien - wie finden wir den „besten“ Algorithmus?
- Als erstes brauchen wir ein strenges **Evaluierungskriterium** - **quantitativ auswertbar**
  - Z.B. „maximiere die CPU-Ausnutzung unter der Bedingung, dass die maximale Antwortzeit 1 Sekunde ist“
  - Z.B. „maximiere Durchsatz (# abgearbeiteter Jobs / Sekunde), so dass die durchschnittliche Durchlaufzeit linear proportional zu der Ausführungszeit aller Jobs ist“
- Die Evaluierung kann mehrere Ansätze benutzen
  - **Deterministisches Modellieren** (**Deterministic Modeling**)
  - **Warteschlangenmodelle** (**Queueing Models**)
  - **Simulationen**

Deterministisches Modellieren

- Man vergleicht den Wert des quantitativen Kriteriums zu jeder Scheduling-S. für (einige) Beispielszenarien
  - Szenario:

Prozess	P1	P2	P3	P4	P5
Burst-Zeit	10	29	3	7	12

  - Kriterium: durchschnittliche Wartezeit



Alg. A: (0+10+39+42+49)/5 = **28** ms

Alg. B: (10+32+0+3+20)/5 = **13** ms

Alg. C: (0+32+20+23+40)/5 = **23** ms

Welche Algorithmen sind das - A, B, C? A: FCFS, B: SJF, C: RR

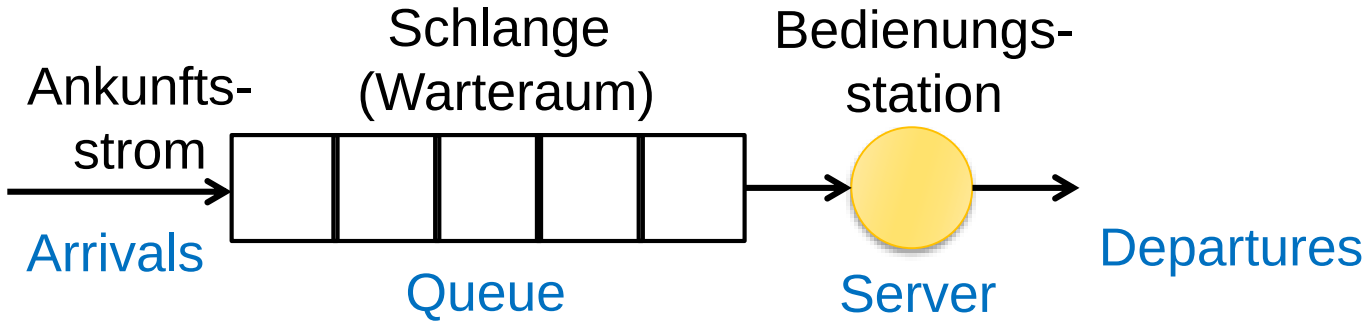
Überblick Warteschlangentheorie

Motivation Warteschlangentheorie

- Video „Why the other line is likely to move faster”
  - https://www.youtube.com/watch?v=F5Ri_HhzIl0, [17a]

Warteschlangentheorie

- Die **Warteschlangentheorie** (oder **Bedienungstheorie**) ist ein Teilgebiet der Wahrscheinlichkeitstheorie
  - Analyse von Systemen, in denen Aufträge von **Bedienungsstationen** bearbeitet werden



- Die „Aufträge“/“Kunden“ treffen einzeln und zu zufälligen Zeitpunkten ein
- Sie werden bedient, sofern mind. eine Bedienungsstation frei ist; sonst reihen sie sich in die Schlange ein

Warteschlangentheorie /2

- Einige Variationen:
  - Mehrere Schlangen und/oder Stationen
  - Einige Kunden verlassen das System, bevor sie bedient worden sind
  - Einige Kunden treten nicht ein, weil ihnen die Warteschlange zu lang erscheint
  - Reihenfolge der Abarbeitung: FIFO, LIFO, Priorität, …
  - …
- Was möchte man typischerweise wissen?
  - Die Anzahl der Kunden im System
  - Ihre (durchschnitliche) Verweilzeiten im System
    - Genauer: Wahrschein-lichkeitsverteilung der Verweilzeiten

Ankunftszeiten /1

- Seien $T_1 < T_2 < \ldots < T_n$ die (zufälligen) Ankunftszeiten der Kunden
- Die Zufallsvariablen $t_k = T_k - T_{k-1}$ ($T_0 = 0$) heissen **Zwischenankunftszeiten** (Zazs)
- Es wird angenommen, dass Zazs unabhängig und identisch verteilt sind
- Häufige Verteilungsfunktionen für Zazs
  - Exponentialverteilung **M** (exp): Dichte $f(x) = ge^{-gx}$ ($x \geq 0$)
  - Erlangverteilung $\mathbf{E_n}$: Summe von n unabhängigen M‘s
  - Hyperexponentialverteilung **H**: Gewichtete Summe von M‘s
  - Deterministische Verteilung **D**: Deterministisch
  - Allgemeine Verteilung **G**: Alles andere

Ankunftszeiten /2

- Wichtige Parameter der Verteilungsfunktionen $F_t(x)$ für Zazs sind
  - **Erwartungswert E[t]** – „durchschnittliche“ Zaz
  - **Varianz D[t]**

- Der Kehrwert des Erwartungswertes der Zazs heißt **Ankunftsrate λ**: $\lambda = 1/E[t]$
  - Gibt an, wie viele Kunden im Durchschnitt pro Zeiteinheit in das System Eintreten

Bedienungszeiten

- Die Zeiten S_1, S_2, … der „Abarbeitung“ an der Station werden auch als identisch verteilte und unabhängige Zufallsvariablen modelliert
- Die Verteilungsfunktion der S_k sei $F_S(x)$ mit Erwartungswert E[S] und Varianz D[S]
- Der Kehrwert dieses Erwartungswerts E[S] heisst die **Bedienrate** μ: $\mu = 1/E[S]$
  - Eine Interpretation?
- Wie viele Kunden im Durchschnitt pro Zeiteinheit von der Bedienungsstation abgefertigt werden können

Charakterisierung der Systeme

- Die „Entwickler“ der Warteschlangentheorie (D.G. Kendall und B.W. Gnedenko) haben zur Charakterisierung der Systeme folgende Notation eingeführt: **A / B / c / m**, wobei
- **A** = Verteilungstyp der Zwischenankunftszeiten
- **B** = Verteilungstyp der Bedienungszeiten
- **c** = Anzahl der parallelen Bediener
- **m** = Kapazität des Warteraums

Formel von Little

- Sei für ein System
  - N_t : Anzahl der Kunden im System zur Zeit t
  - V_k : die Verweilzeit eines Kunden k im System
  - λ : die Ankunftsrate der Kunden
- Dann sagt die **Formel von Little** (Link)

$$E[N] = \lambda * E[V]$$

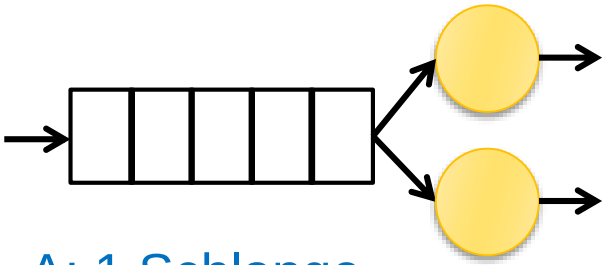
- D.h. in etwa: „Ankunftsrate λ = (durchschnittliche Kundenanzahl im System) / (durchschnittliche Verweilzeit im System)
- Man kann aus zwei der Variablen jeweils den dritten Parameter berechnen usw.

Zusammenfassung Konzepte

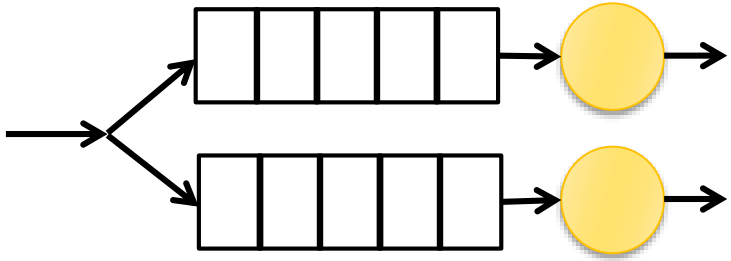
- Video „Concepts of Queueing Theory“
  - https://www.youtube.com/watch?v=QUT3pOTdHgA
  - Bis ca. 2:20 (min:sec)
- Video 2.1 zum Operations Management Tutorial Auslastung und Wartezeiten in Bediensystemen
  - https://www.youtube.com/watch?v=bDkDejhgnG0
  - Von 0:50 bis ca. 3:50 (min:sec)

Vergleich der Systementwürfe

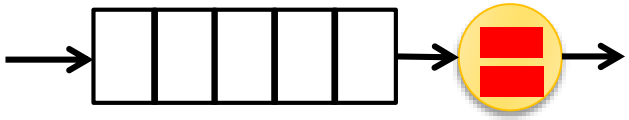
- Die Formeln lassen alltägliche Situationen oder alternative Entwürfe bei BS untersuchen:



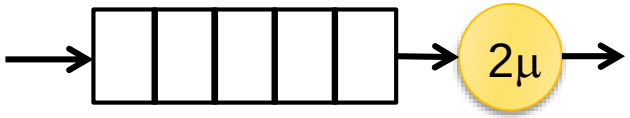
A: 1 Schlange,
2 Bediener (Flughafen)



B: 2 Schlangen, 2 Bediener (Kunden schon bei Ankunft zu 50% verteilt)



C: Bediener kann 2 Kunden gleichzeitig abarbeiten



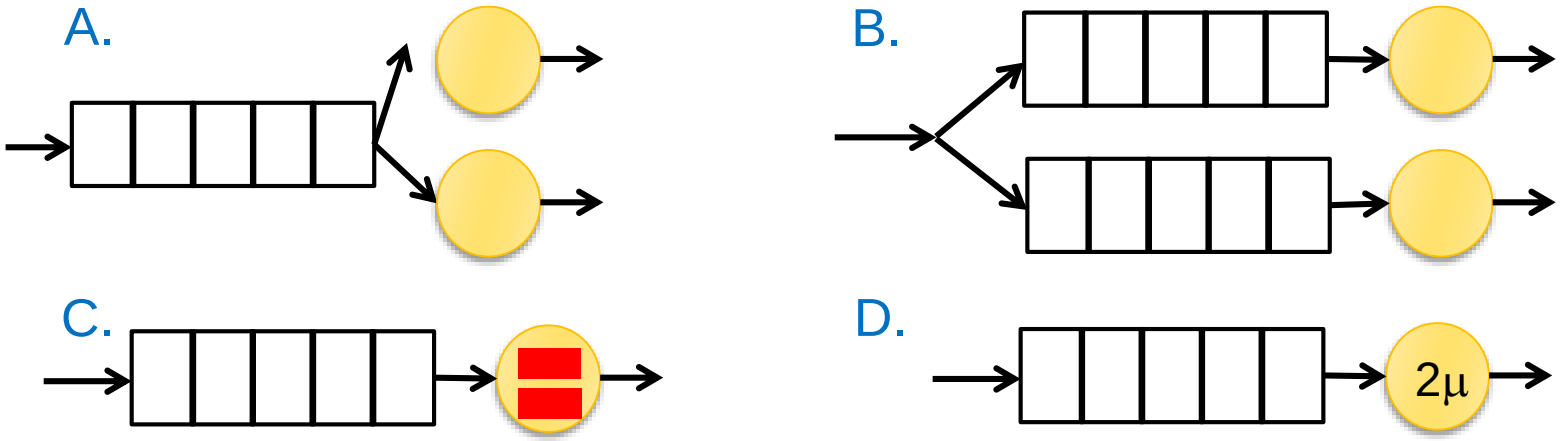
D: Bediener arbeitet 2x schneller (2μ)

Überall:
λ = 1.8
μ = 1.0

In welchem Fall ist E[N], die erwartete Anzahl der Kunden im System, am geringsten?

Aus: http://www.stochastik.tu-clausthal.de/Presse/Schulen/Warteschlangen

Vergleich der Systementwürfe



- A. Es stehen zwei Bediener zur Verfügung. Die ankommenden Kunden werden dem jeweils freiwerdenden Schalter zugewiesen.
- B. Auch hier stehen zwei parallele Bediener zur Verfügung. Allerdings werden die Kunden bereits bei ihrer Ankunft gleichmäßig (d. h. zu je 50%) auf die beiden Warteschlangen verteilt.
- C. Es gibt nur einen Bediener und eine Warteschlange. Der Bediener kann jedoch zwei Kunden gleichzeitig bedienen
- D. Es gibt nur einen Bediener und eine Warteschlange. Der Bediener arbeitet jedoch mit doppelter Geschwindigkeit.

E[N] = … A: 5.64, B: 9.90, C: 6.35, D: 4.49

Aus: http://www.stochastik.tu-clausthal.de/Presse/Schulen/Warteschlangen

Warteschlangenmodelle für das Scheduling

- Für viele Prozesse kann man die Verteilung der Längen von CPU-Bursts ermitteln (oft Exponentialv.)
  - Welche Verteilung ist das in der WS-Theorie?

    Bedienungszeiten, da die CPU die Jobs abarbeitet
- Genauso lässt sich die Ankunftsrate neuer CPU-Anfragen durch eine Verteilung nähern
- Aus diesen beiden Verteilungen kann man für die einfachen Algorithmen den durchsch. Durchsatz, CPU-Ausnutzung, Wartezeit, … berechnen
- Leider sind diese Modelle idealisiert und beschreiben kaum komplexe, reale Systeme

Quellen / Mehr zur Warteschlangentheorie

- TU Clausthal, Institut für Angewandte Stochastik und Operations Research, Stochastische Modelle in den Ingenieurwissenschaften, Link
- K. Berger, P. Christodoulides, K. Grill, Warteschlangentheorie, Link
- Wikipedia – Warteschlangentheorie, Link

Zusammenfassung Scheduling

- Scheduling - Grundlagen
  - Kriterien, Typen von Systemen
- Scheduling-Algorithmen
  - Stapelverarbeitung
  - Scheduling für Interaktive Systeme
- Quellen Scheduling: todo

Danke.

Zusatzfolien: Scheduling

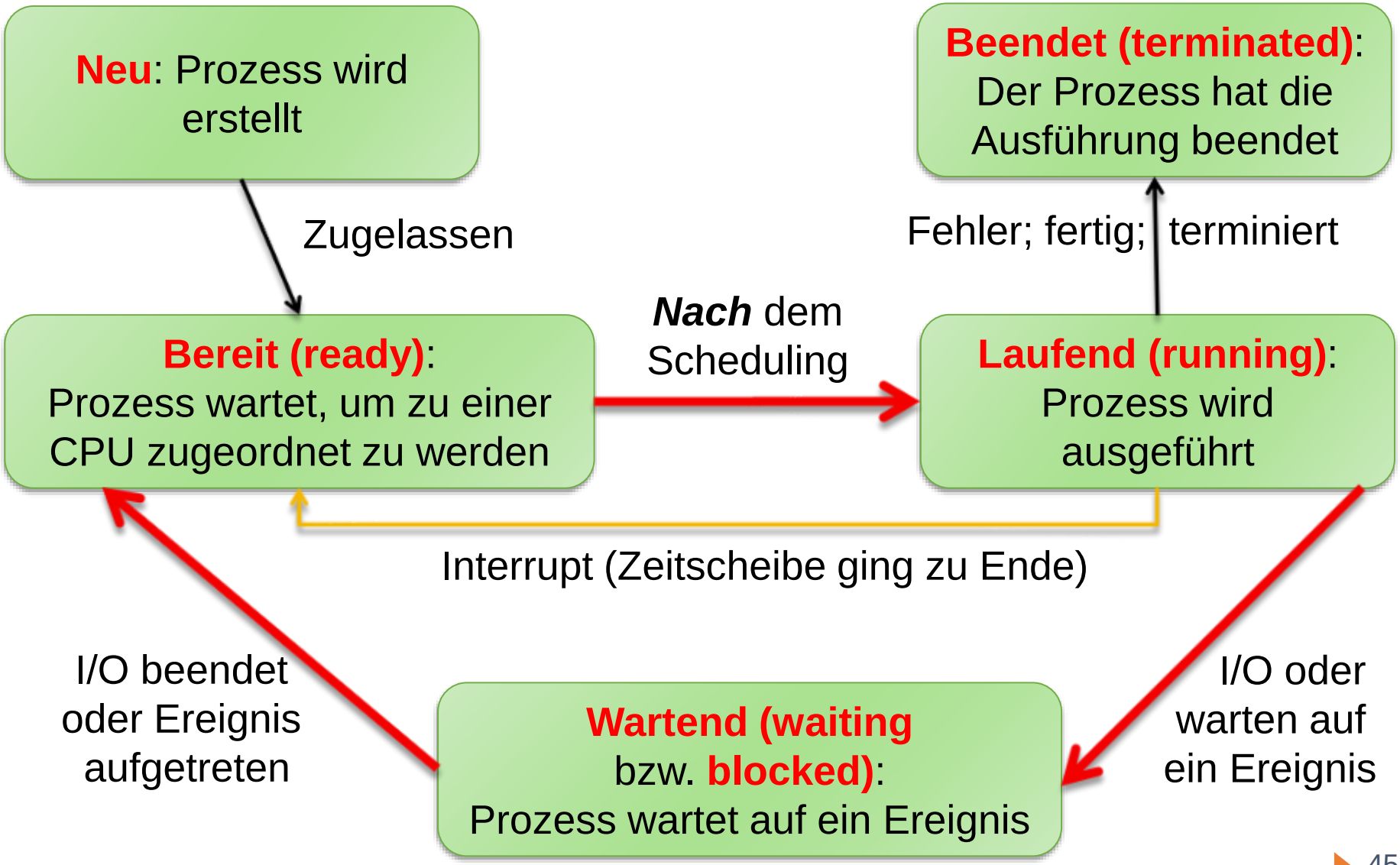
Thread-Scheduling

- Scheduling hängt stark davon ab, ob Threads auf **Anwenderebene** oder **Kernebene** benutzt werden
- Benutzerebene
  - Der Kern weiß nichts von Threads; er wählt nur einen Prozess P und übergibt ihm die Kontrolle für ein Quantum
  - Der Thread-Scheduler innerhalb von P entscheidet dann, welcher Thread laufen soll, z.B. P1
  - Da es keine Interrupts „in“ P gibt, kann dieser bis zum Ende des Quantums weiterlaufen usw., usw.
    - Das Aushungern anderer Threads in P ist möglich
    - Aber nicht das Aushungern anderer Prozesse!
  - Die Scheduling-S. innerhalb von P kann beliebig sein
    - Häufig: Round-Robin und Prioritätsscheduling

Thread-Scheduling /2

- Kern-Ebene Threads
  - Hier ist das BS der Existenz von Threads „bewusst“
  - BS verhindert das Aushungern dieser innerhalb eines Prozesses
- Problem hier ist **Leistung**
  - Thread-Wechsel auf Anwenderebene kostet nur einige Zyklen
  - Der Wechsel im Kern dauert einige Größenordnungen länger
  - Der Wechsel zwischen den Threads verschiedener Prozesse noch länger - warum?
    - Wechsel der Seitentabelle
    - Caches müssen ungültig gemacht werden

Wann ist Scheduling nötig/möglich? /1

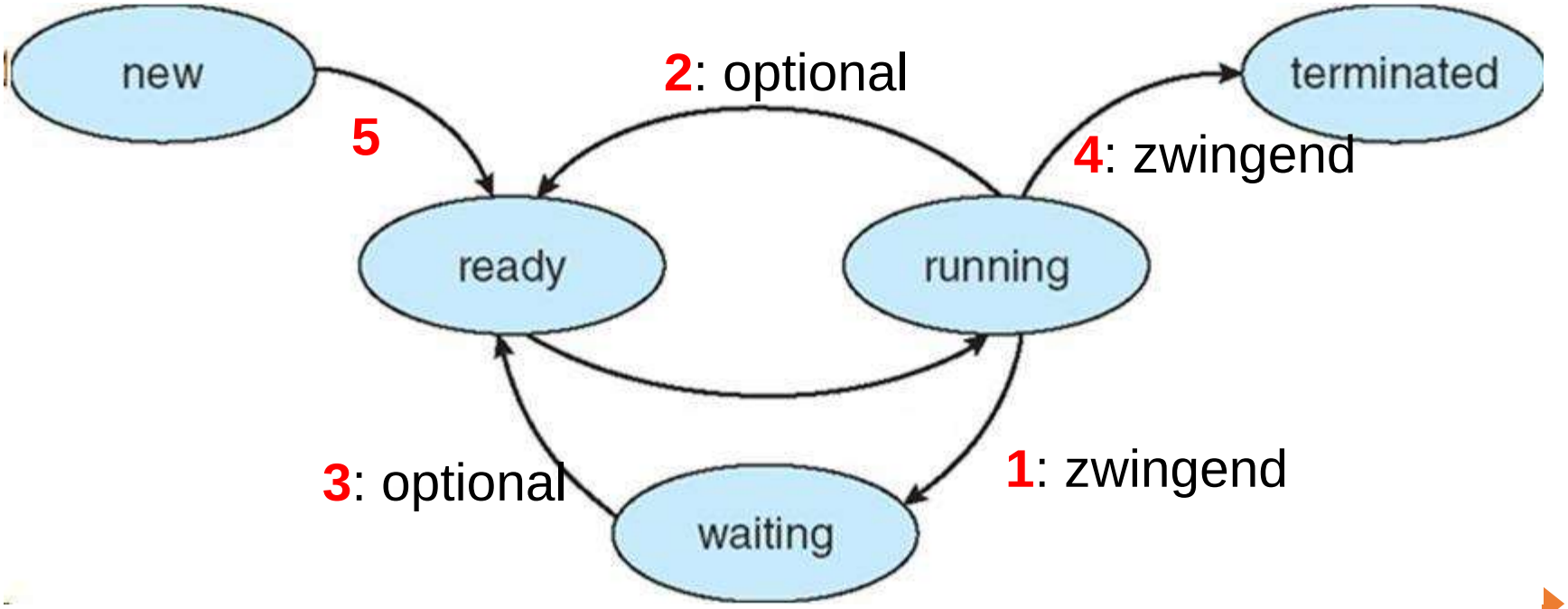


Wann ist Scheduling nötig/möglich? /2

- **1.** Wenn der Prozess von „running“ zu „waiting“ wechselt
  - Wegen einer I/O-Anfrage, eines Semaphors, eines Systemaufrufs (wait() nach fork()), eines Seitenfehlers …
- **2.** Wenn der Prozesszustand von „running“ zu „ready“ wechselt
  - z.B. falls ein I/O-Interrupt, Timer-Interrupt oder ein Trap auftritt
- **3.** Wenn ein (blockierter) Prozess von „waiting“ zu „ready“ wechselt
  - Meistens bei der Beendigung einer I/O-Phase
- **4.** Bei der Beendigung eines Prozesses
- **5.** Ggf. bei der Erzeugung eines neuen Prozesses

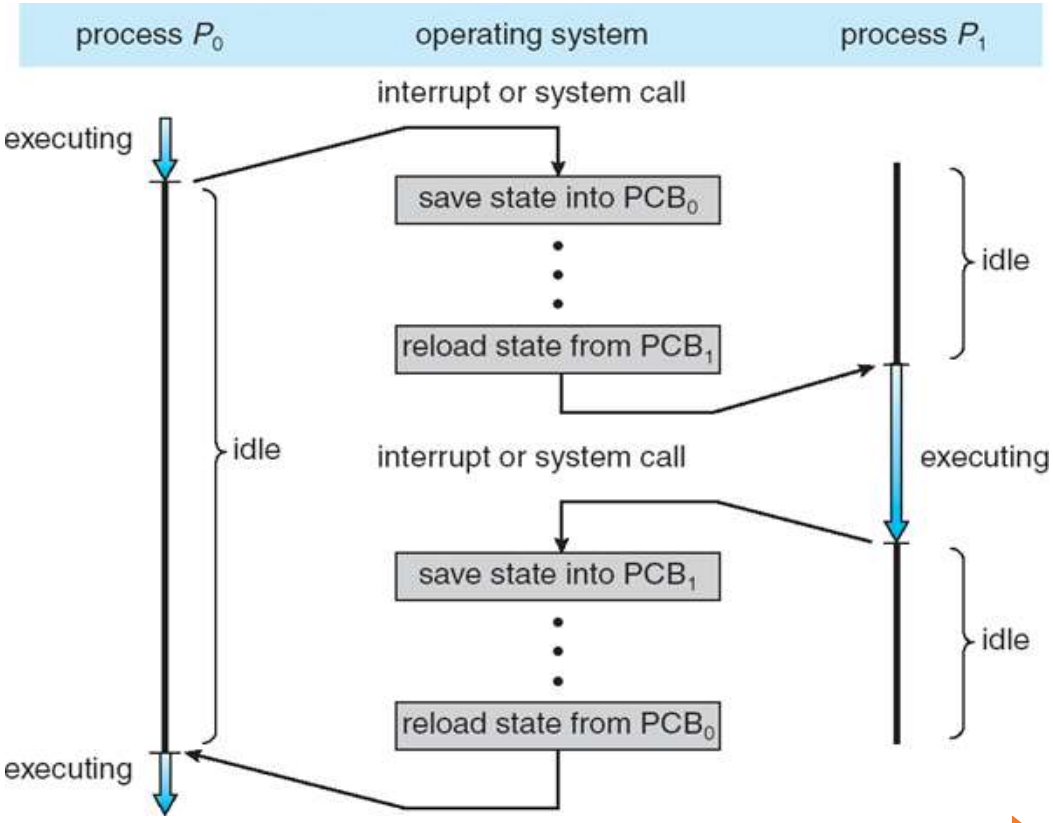
Unterbrechende (preemptive) S-Strategien

- Wenn ein BS in Fällen 2 und 3 ein Thread- bzw. Prozesswechsel erzwingen kann, heißt die Scheduling-Strategie **unterbrechend** (**preemptive**)
  - Windows ab Win 95, alle Linux Versionen



Scheduler und Dispatcher

- Wir haben auch den Begriff **Dispatcher** kennengelernt – was ist der Unterschied?
- Dispatcher ist der „ausführende Teil“ eines Kurzzeit-Schedulers
- Implementiert den Prozesswechsel



Ziele von Scheduling-Strategien - Alle Systeme

- **Fairness**
  - Jeder Prozess bekommt einen fairen Anteil der Rechenzeit
- **Policy Enforcement**
  - Vorgegebene Strategien werden durchgesetzt
- **Balance**
  - Alle Teile des Systems sind ausgelastet

Ziele - Stapelverarbeitungssysteme

- **Durchsatz** (**throughput**)
  - Die Anzahl der abgearbeiteten Jobs pro Sekunde maximieren
- **Durchlaufzeit** (**turnaround time**)
  - Die Zeit von Start bis zur Beendigung eines Jobs minimieren
- **CPU-Ausnutzung** (**CPU utilization**)
  - Die CPU soll möglichst stark ausgelastet werden
- **Wartezeit** (**waiting time**)
  - Summe der Zeiten, die ein P. trotzt „ready“-Zustands warten muss

Ziele - Interaktive Systeme

- **Antwortzeit**
  - Schnelle Antwort auf Anfragen
- **Proportionalität**
  - Erwartungen des Benutzers erfüllen
    - z.B. Wartezeit beim Dekomprimieren einer Datei kann länger als beim Schließen eines Fensters sein

Ziele - Echtzeitsysteme

- **Deadlines einhalten**
  - Wenn ein wichtiger Prozess nicht rechtzeitig Kontrolle übernimmt, kann eine Maschine / Auto beschädigt werden
- **Vorhersagbarkeit**
  - Z.B. Multimediasysteme: wenn ein Audioprozess zu unregelmäßig läuft, wird sich die Qualität drastisch verschlechtern