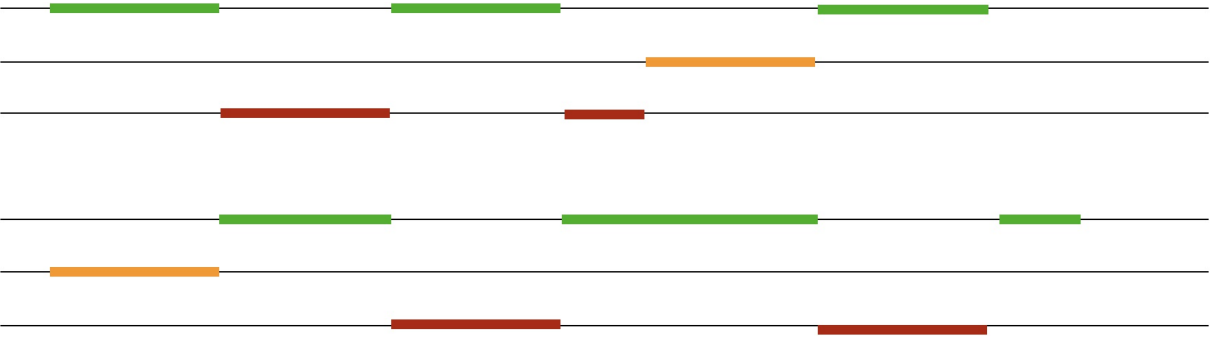


# Prozess-Scheduling II

Tutorium 04

Betriebssysteme SS2024

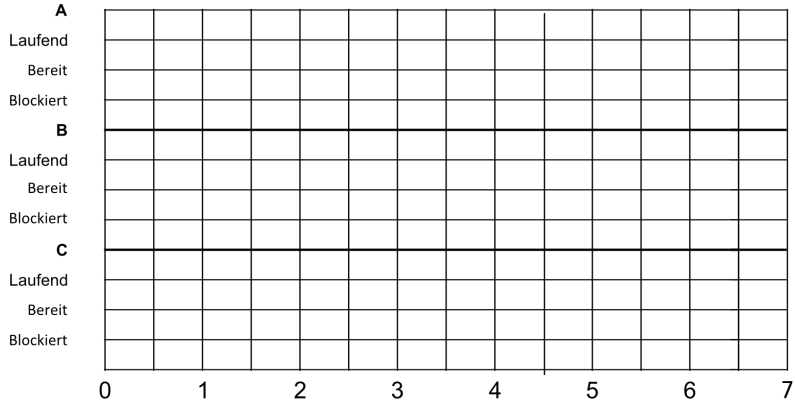
# Inhaltsverzeichnis

# Highest Priority First

- Alle Prozesse im System haben eine assozierte Priorität
- Zuteilung bereiter Prozesse basierend auf der Priorität
- Prioritäten können statisch sein oder dynamisch zugewiesen werden
- Bei dynamischen Prioritäten entspricht HPF gerade SJF
- Mögliche auftretende Probleme:
  - Aushungerung von Prozessen mit niedrigerer Priorität
  - Prioritätsumkehr bei Zugriff auf exklusive Ressourcen

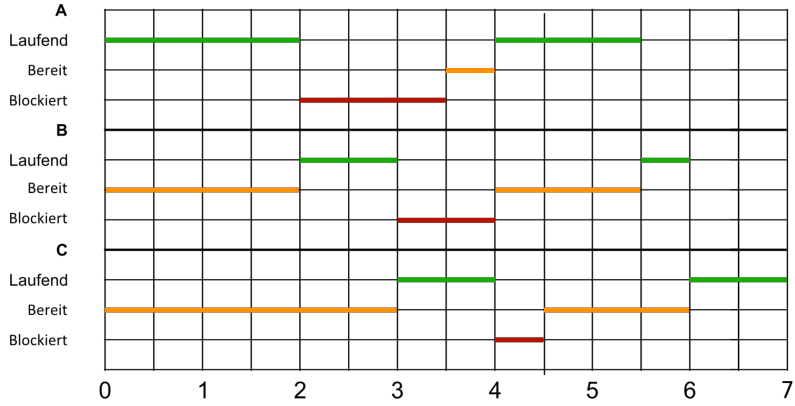
# Highest Priority First - Beispiel

- Prozess A: Start bei $t_A$=0s, läuft 2s, blockiert 1,5s, läuft 1,5s
- Prozess B: Start bei $t_B$=0s, läuft 1s, blockiert 1s, läuft 0,5s
- Prozess C: Start bei $t_C$=0s, läuft 1s, blockiert 0,5s, läuft 1s
- Prioritätsliste: P(A) $>$ P(B) $>$ P(C)



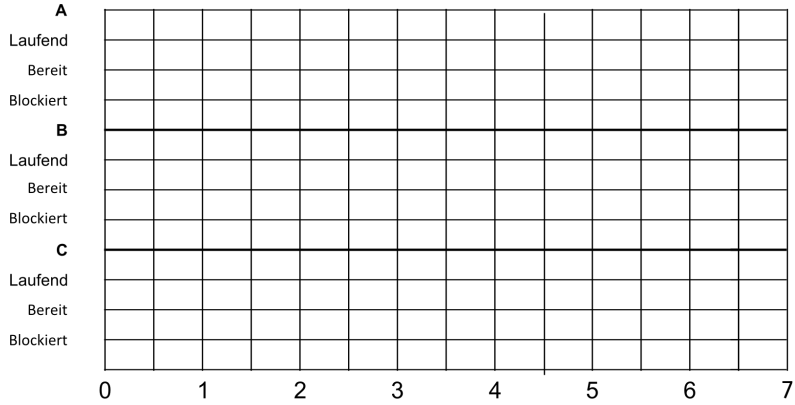
# Highest Priority First - Beispiel

- Prozess A: Start bei $t_A$=0s, läuft 2s, blockiert 1,5s, läuft 1,5s
- Prozess B: Start bei $t_B$=0s, läuft 1s, blockiert 1s, läuft 0,5s
- Prozess C: Start bei $t_C$=0s, läuft 1s, blockiert 0,5s, läuft 1s
- Prioritätsliste: P(A) $>$ P(B) $>$ P(C)



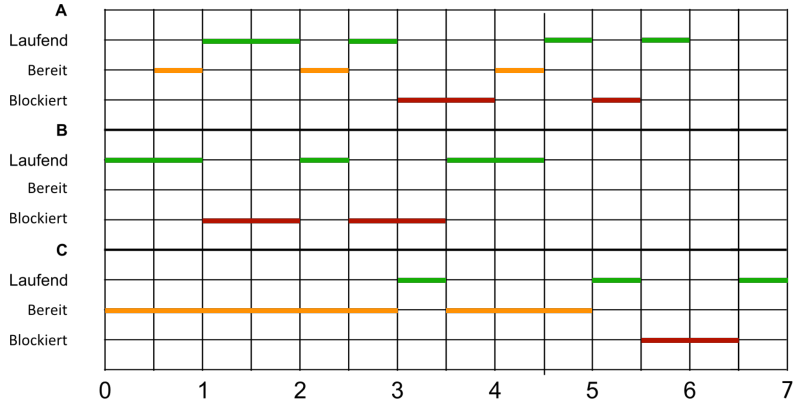
# Präemptives Highest Priority First - Beispiel

- Prozess A: Start bei $t_A$=0,5s. Läuft 1,5s, blockiert 1s, läuft 0,5s, blockiert 0,5s, läuft 0,5s
- Prozess B: Start bei $t_B$=0s. Läuft 1s, blockiert 1s, läuft 0,5s, blockiert 1s, läuft 1s
- Prozess C: Start bei $t_C$=0s. Läuft 1s, blockiert 1s, läuft 0,5s
- Prioritätsliste: P(B) $>$ P(A) $>$ P(C)



# Präemptives Highest Priority First - Beispiel

- Prozess A: Start bei $t_A$=0,5s. Läuft 1,5s, blockiert 1s, läuft 0,5s, blockiert 0,5s, läuft 0,5s
- Prozess B: Start bei $t_B$=0s. Läuft 1s, blockiert 1s, läuft 0,5s, blockiert 1s, läuft 1s
- Prozess C: Start bei $t_C$=0s. Läuft 1s, blockiert 1s, läuft 0,5s
- Prioritätsliste: P(B) $>$ P(A) $>$ P(C)

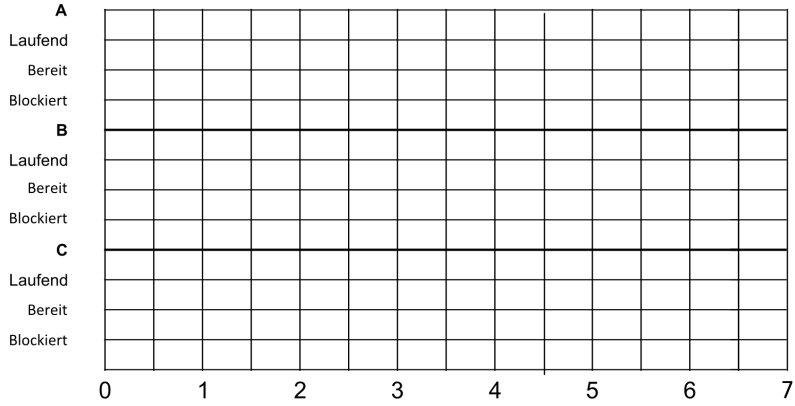


# Round Robin Strategie

- Die Strategie arbeitet mit Zeitlslots von vorgegebener Dauer
- Bereite Prozesse werden in eine Warteschlange eingereiht
- Prozesse in der Warteschlange erhalten der Reihe nach einen Zeitslot für die Ausführung
- Ein laufender Prozess wird nach Ablauf des Zeitslots verdrängt
- Bei Blockierung oder Terminierung eines laufenden Prozesses vor Ablauf der Zeitscheibe wird direkt der nächste wartende Prozess zugeteilt

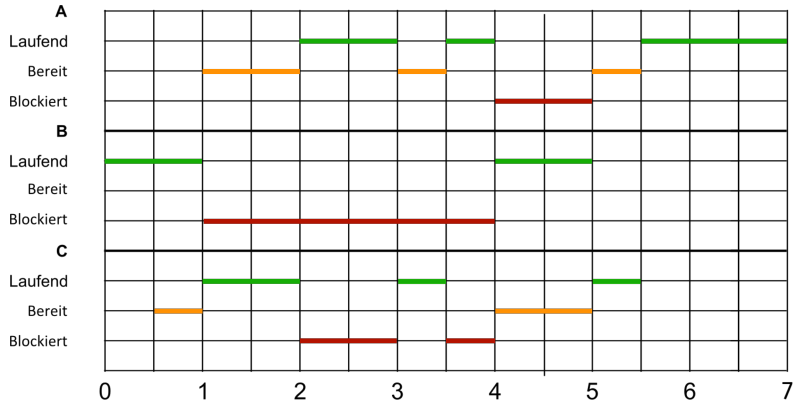
# Round Robin - Beispiel

- Prozess A: Start bei $t_A$=1s, läuft 1,5s , blockiert 1s, läuft 1,5s
- Prozess B: Start bei $t_B$=0s, läuft 1s, blockiert 3s, läuft 1s
- Prozess C: Start bei $t_C$=0,5s, läuft 1s, blockiert 1s, läuft 0,5s, blockiert 0,5s, läuft 0,5s
- Länge der Zeitscheibe 1s.



# Round Robin - Beispiel Lösung

- Prozess A: Start bei $t_A$=1s, läuft 1,5s , blockiert 1s, läuft 1,5s
- Prozess B: Start bei $t_B$=0s, läuft 1s, blockiert 3s, läuft 1s
- Prozess C: Start bei $t_C$=0,5s, läuft 1s, blockiert 1s, läuft 0,5s, blockiert 0,5s, läuft 0,5s
- Länge der Zeitscheibe 1s.



# Threads

- Threads sind Aktivitätsträger
- Ein Prozess kann mehrere Threads verwalten
- Threads innerhalb eines Prozesses haben Zugriff auf die selben Daten
- Jeder Thread hat aber einen eigenen Program Counter, eigene Register und einen eigenen Stack
- Zwei Arten von Threads möglich:
  - User-Level Threads
  - Kernel-Level Threads

# Kernel-Level Threads

- Werden vom Scheduler des Betriebssystems verwaltet
- Fairness wichtig bei Scheduling-Strategie
- Bilden die Basis für User-Level Threads
- Beispiele:
  - POSIX Threads (bei UNIX Systemen verbreitet)
  - TPL Threads (bei Windows Systemen verbreitet)
  - Java Threads sind User-Level Threads die aber meistens direkt auf Kernel-level Threads abgebildet werden

# User-Level Threads

- User-Level-Threads werden auf Kernel-Threads abgebildet
- Parallelität ist deshalb eingeschränkt
- Können sich gegenseitig beeinflussen (bspw. blockieren)
- Schnelleres Umschalten möglich, da kein Kontextwechsel notwendig ist
- Anwendung beliebiger Scheduling-Strategien möglich
- Beispiele:
  - Java Virtual Threads (seit Java SE 21)
  - Windows Fibers