

# Betriebssysteme (BS)

## VL 9.4 – Fadenverwaltung – Linux

**Volkmar Sieh / Daniel Lohmann**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität  
Erlangen Nürnberg

WS 21 – 20. Dezember 2021

[https://www4.cs.fau.de/Lehre/WS21/V\\_BS](https://www4.cs.fau.de/Lehre/WS21/V_BS)



# Agenda

---

Einordnung

Betriebssystemfäden

Motivation

Kooperativer Fadenwechsel

Präemptiver Fadenwechsel

**Ablaufplanung**

Grundbegriffe und Klassifizierung

unter Windows

unter Linux

Zusammenfassung



# Linux Scheduler: Historie

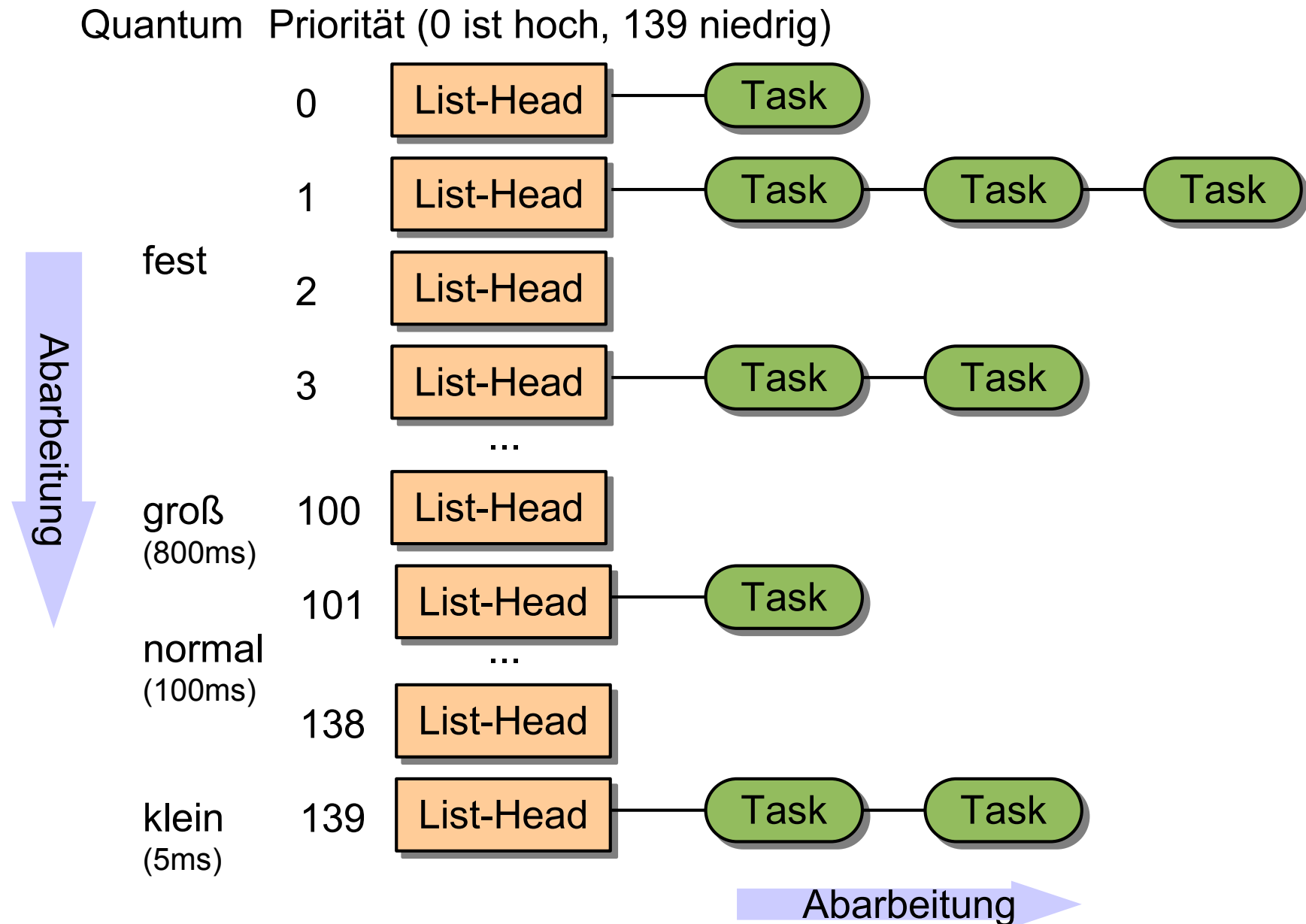
- Kernel < 2.2: Ring-Warteschlange mit Round Robin
  - kein SMP Support
  - keine Echtzeitprioritäten
- Kernel  $\geq 2.2$ : Erster SMP-Support
  - Einführung von *scheduling classes*
- Kernel  $\geq 2.4$ : O(n) Scheduler
  - Mehr Fairness durch Scheduler-Epochen
  - Miese Leistung auf SMP-Systemen
- Kernel  $\geq 2.5$ : O(1) Scheduler
  - Effiziente Epochenverwaltung durch *active* und *expired* Listen
  - Per-Core *runqueues* mit Lastausgleich
  - Viele Heuristiken zur Bevorzugung interaktiver Anwendungen
- Kernel  $\geq 2.6.23$ : *Completely Fair Scheduler (CFS)*
  - Schlanker Scheduler, der die Rechenzeit "ideal fair" verteilt
  - Hierarchisches Scheduling durch *scheduler groups*



- sind die **Linux Kernel-Abstraktion** für ...
  - **UNIX Prozesse:** ein Kontrollfaden in einem Adressraum
  - **Linux Threads:** spezieller Prozess, der sich seinen virtuellen Adressraum mit mindestens einem anderen *Thread* teilt
- sind die vom Scheduler betrachteten Aktivitäten
  - ein Programm mit vielen Threads bekommt unter Linux mehr Rechenzeit als **ein** klassischer Prozess
  - gleiches gilt allerdings auch für ein Programm mit einem Prozess und vielen Kindprozessen

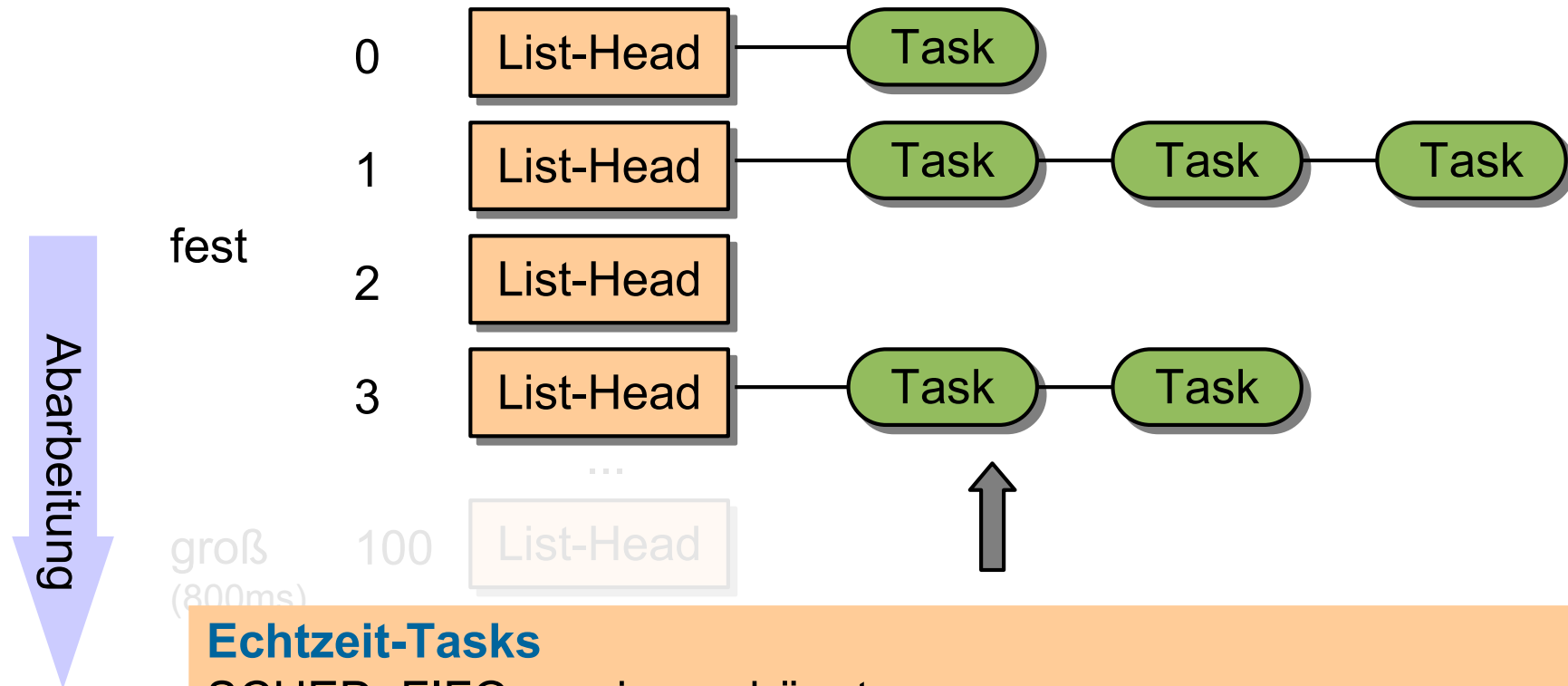


# Multi-Level Queues



# Multi-Level Queues

Quantum    Priorität (0 ist hoch, 139 niedrig)



## Echtzeit-Tasks

SCHED\_FIFO    nie verdrängt

SCHED\_RR    verdrängt, wenn feste Zeitscheibe abgelaufen

Echtzeit-Tasks verdrängen jeden anderen normalen Task.

Durch die einfache Strategie kann das Verhalten in einer weichen Echtzeitumgebung recht gut vorhergesagt werden.



# O(1) Scheduler: Multi-Level Queues

Quantum    Priorität (0 ist hoch, 139 niedrig)

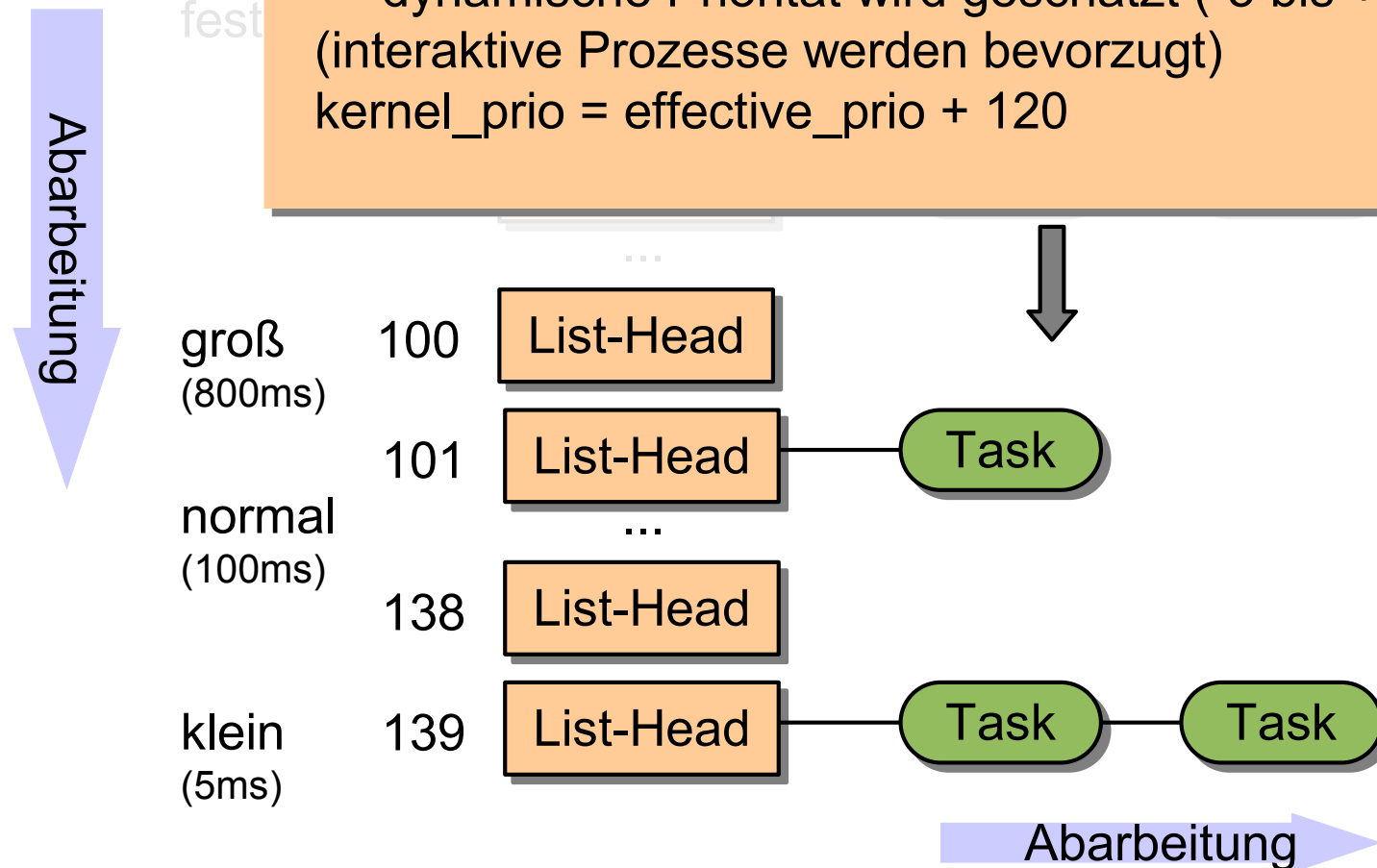
## Gewöhnliche Tasks

$\text{effective\_prio} = \text{static\_prio} + \text{dynamic\_prio}$

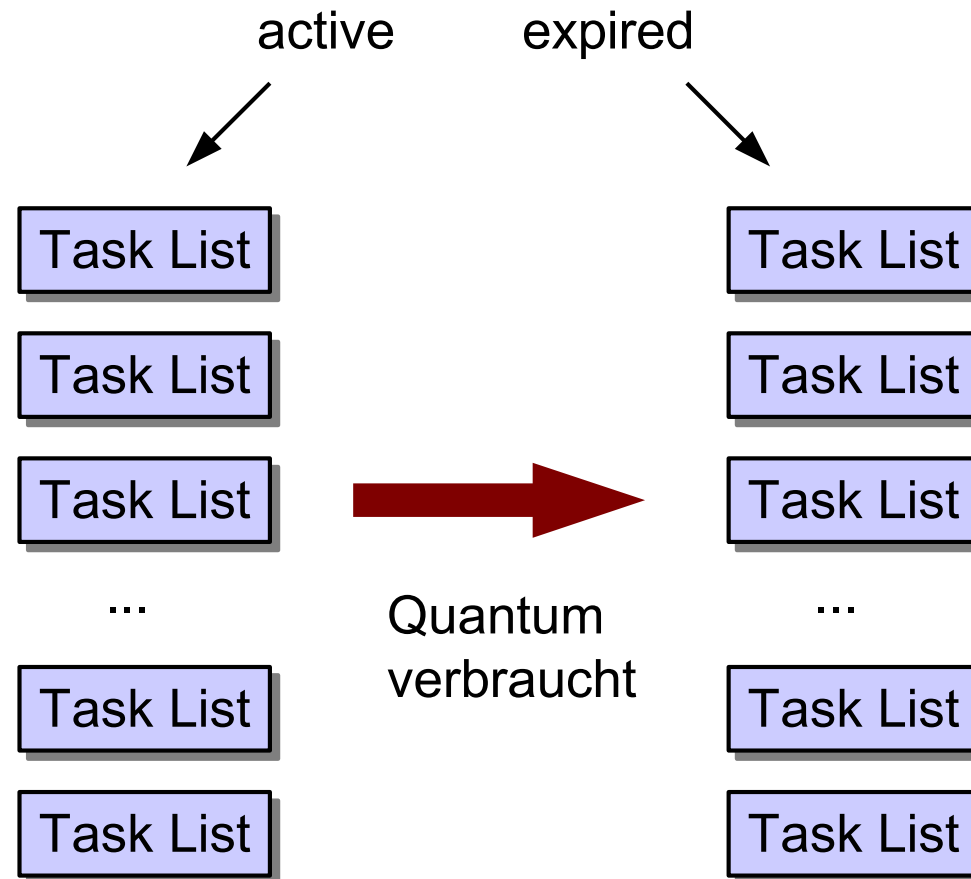
- statische Priorität entspricht dem nice value (-19 bis +20)
- dynamische Priorität wird geschätzt (-5 bis +5)

(interaktive Prozesse werden bevorzugt)

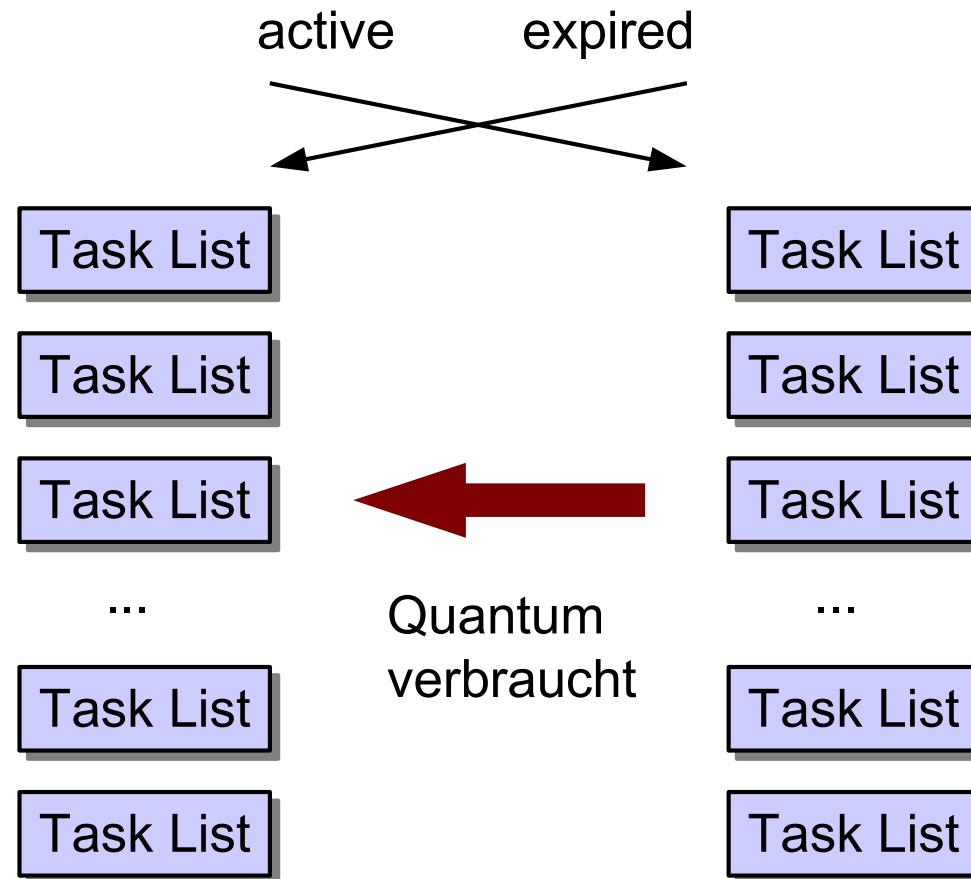
$\text{kernel\_prio} = \text{effective\_prio} + 120$



# O(1) Scheduler: Active und Expired



# O(1) Scheduler: Active und Expired



# Fazit Linux O(1) Scheduler

---

- ***„interactive, probabilistic, online, preemptive, multiprocessor CPU scheduling“***
- Bevorzugung interaktiver Prozesse
  - schnelle Reaktion auf Eingaben
  - gleichzeitig Fortschrittgarantie für CPU-lastige Prozesse
- O(1) bei allen Operationen des Scheduler
  - Einfügen, Entfernen, Scheduling-Entscheidung
- Mehrprozessorunterstützung
  - Mehrere Bereit-Listen: Parallele Scheduler Ausführung
  - Keine Idle-Phasen (war ein Problem beim alten Linux Scheduler)
  - CPU-Lastausgleich

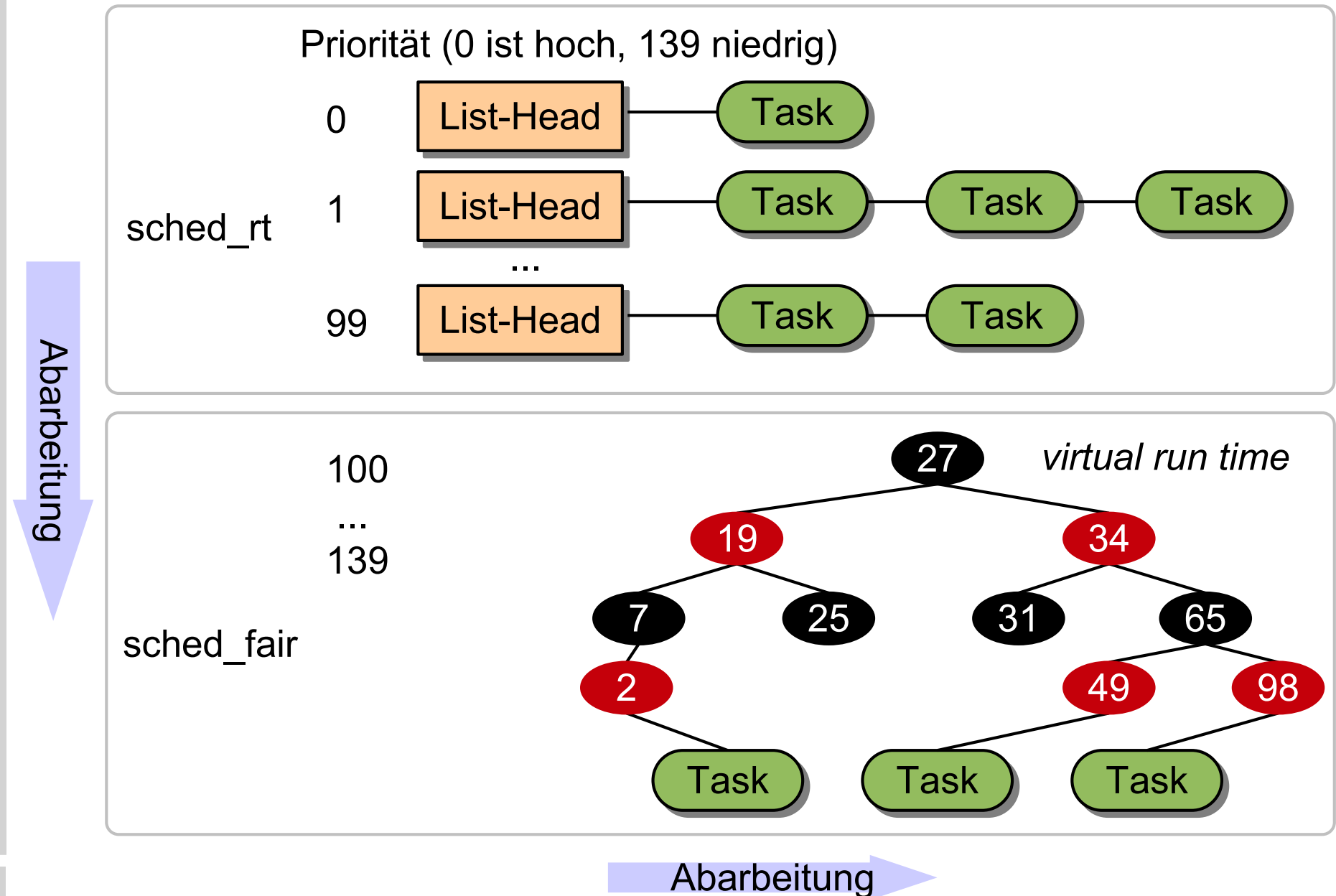


# Completely Fair Scheduler (CFS)

- Ansatz: Ablaufbereite Tasks bekommen die Rechenzeit gleichmäßig ("fair") zugeteilt
  - bei  $n$  Tasks jeweils  $1/n$ -tel der CPU-Leistung
  - hierarchische Zuteilung durch *scheduling groups*
- CFS läuft nur bei SCHED\_NORMAL
  - Echtzeittask (SCHED\_RR und SCHED\_FIFO) wie bisher
  - ansonsten: Task mit *geringster* CPU-Zeit hat höchste Priorität
- Scheduling-Kriterium ist die bislang zugeteilte CPU-Zeit
  - Ready-Liste als Rot-Schwarz-Baum, sortiert nach der Zeit
  - Komplexität  $O(\log N)$   
(in der Praxis trotzdem effizienter als alter  $O(1)$ -Scheduler)
  - Prioritäten (im Sinne von nice) werden durch "schnellere/langsamere" Uhren abgebildet



# Completely Fair Scheduler (CFS)



# Fazit: Completely Fair Scheduler (CFS)

- **„interactive, probabilistic, online, preemptive, multiprocessor CPU scheduling“**
- **Keine** direkte Bevorzugung interaktiver Prozesse
  - Fortschrittsgarantie für alle Prozesse
  - Im Vergleich zum  $O(1)$ -Scheduler:  
Besserstellung von Hintergrundprozessen
- $O(\log n)$  bei den meisten Operationen des Scheduler
  - Einfügen, Scheduling-Entscheidung
- Hierarchie durch *scheduling groups*
  - z.B. Fairness zwischen Benutzern: Eine Gruppe pro Benutzer
  - innerhalb der Gruppe: Fairness zwischen allen Mitgliedern
  - Benutzer kann weitere Gruppen erstellen,  
um "seinen Anteil" aufzuteilen



# Agenda

---

Einordnung

Betriebssystemfäden

Motivation

Kooperativer Fadenwechsel

Präemptiver Fadenwechsel

Ablaufplanung

**Zusammenfassung**



# Zusammenfassung

---

- *Threads* sind Koroutinen des Betriebssystems
  - BS hat Entzugsmechanismen
  - Strategie der Ablaufplanung wird als *Scheduling* bezeichnet
- *Scheduling* hat großen Einfluss auf die Performanz des Gesamtsystems, es legt fest, ...
  - welche Prozesse warten und welche voranschreiten
  - welche Betriebsmittel wie ausgelastet sind
- Es gibt verschiedenste Varianten des Scheduling
  - nur wenige Unterschiede bei gängigen PC/Workstation-Betriebssystemen
  - eventuell aber starke Unterschiede in anderen Anwendungsdomänen

