Hochschule Bochum
Bochum University
of Applied Sciences

Vertiefung Systemtechnik

Prof. Dr. Peter Gerwinski

23. Oktober 2012

Literaturempfehlung

Prof. Dr. Joachim Wietzke, FH Darmstadt,
Prof. Dr. Manh Tien Tran, FH Kaiserslautern:

Automotive Embedded Systeme

Springer-Verlag, Berlin, Heidelberg 2005

Lizenz: proprietär

(gesetzt mit LaTeX, ca. 10 €)

2 Theorie der Echtzeitsysteme

2.2 Ressourcen

Ressourcen reservieren

- *Semaphor*
  gemeinsame Variable mehrerer Prozesse
  zur Regelung des Zugriffs auf eine Ressource

  Ressource belegt $\longrightarrow$ Kontextwechsel

- *Mutex*
  Mechanismus, damit immer nur ein Prozeß gleichzeitig
  auf eine Ressource zugreifen kann

- *Spinlock* (*busy waiting*)
  leichtgewichtige Alternative zu Kontextwechsel

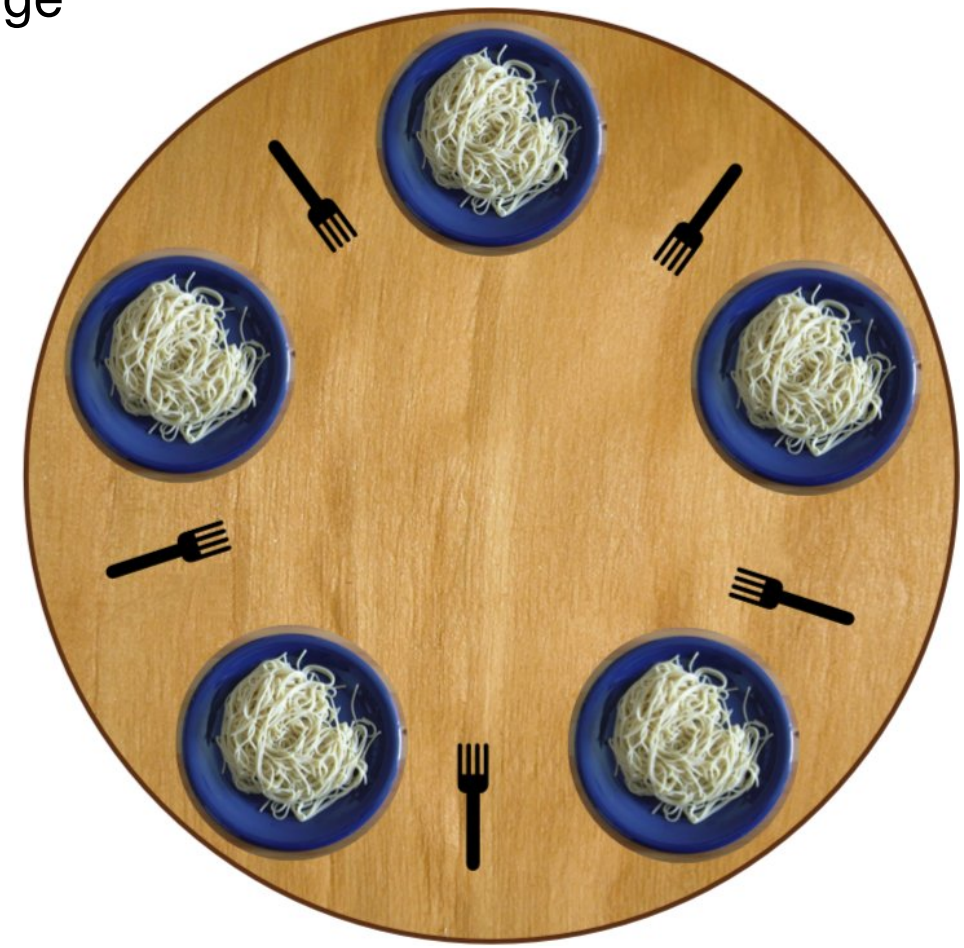
- *Kritischer Abschnitt – critical section*
  Programmabschnitt zwischen Reservierung
  und Freigabe einer Ressource
  $\longrightarrow$ sollte immer so kurz wie möglich sein

2 Theorie der Echtzeitsysteme

2.2 Ressourcen

Verklemmungen: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge (z. B. *busy waiting*)



Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel $\longrightarrow$ *Verklemmung*
schweigen $\longrightarrow$ *Deadlock*
philosophieren weiter $\longrightarrow$ *Livelock*

2 Theorie der Echtzeitsysteme

2.2 Ressourcen

Verklemmungen: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge
  (z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- Exklusivität $\longrightarrow$ Spooling
- *hold and wait* $\longrightarrow$ simultane Zuteilung
- Entzug nicht möglich $\longrightarrow$ Prozesse suspendieren, beenden, *Rollback*
- zirkuläre Blockade $\longrightarrow$ Reihenfolge abhängig von Ressourcen

2 Theorie der Echtzeitsysteme

2.3 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert;
  Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Prozeß mit positivem Zähler gibt,
  bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

⟶ *dynamische Prioritätenvergabe*:
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität
  Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

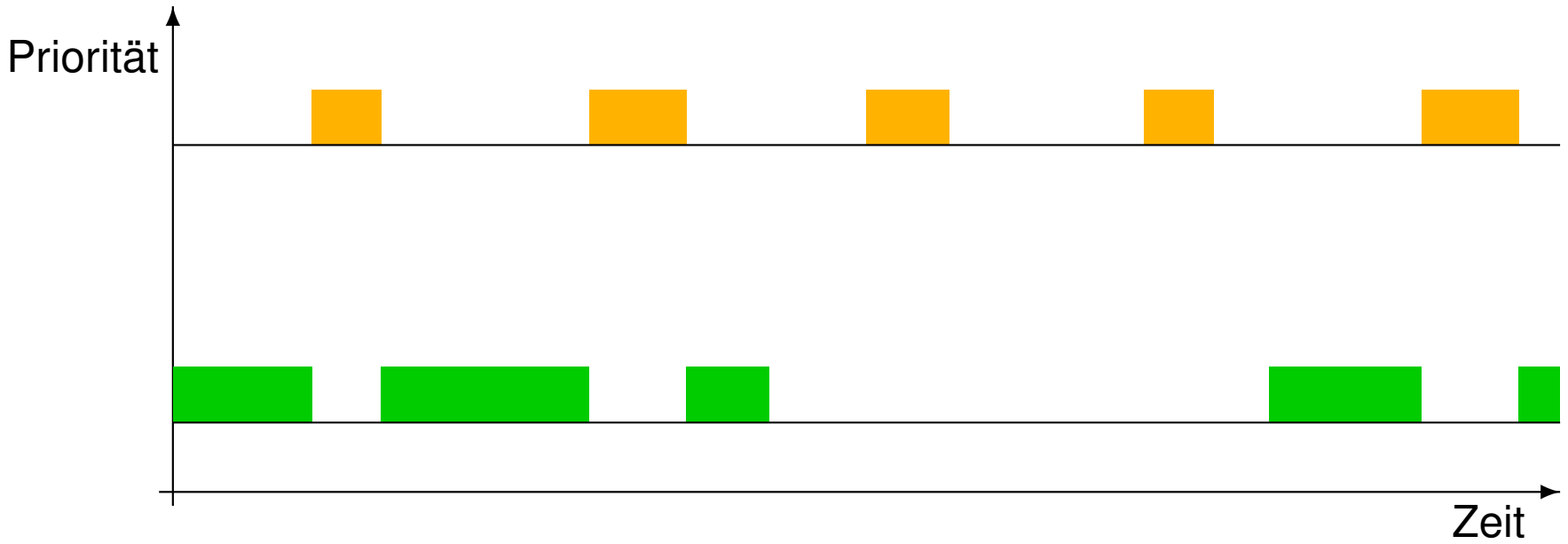
⟶ *statische Prioritätenvergabe*

2 Theorie der Echtzeitsysteme

2.3 Prioritäten

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität
  Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

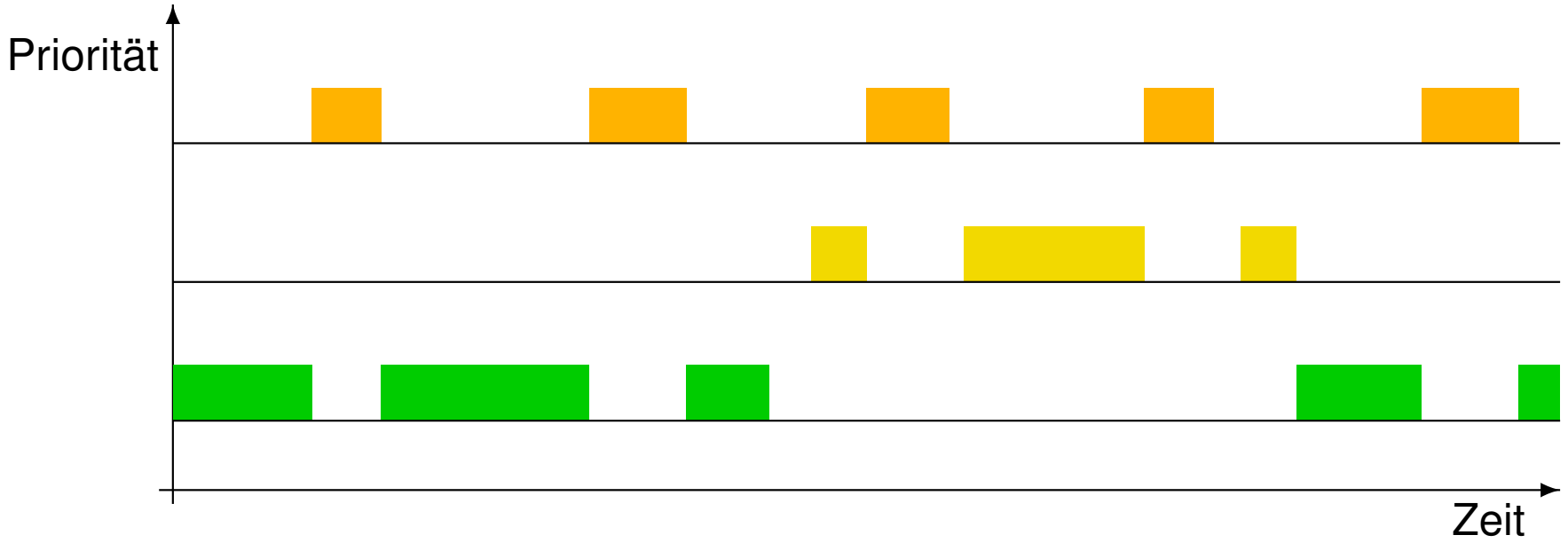


2 Theorie der Echtzeitsysteme

2.3 Prioritäten

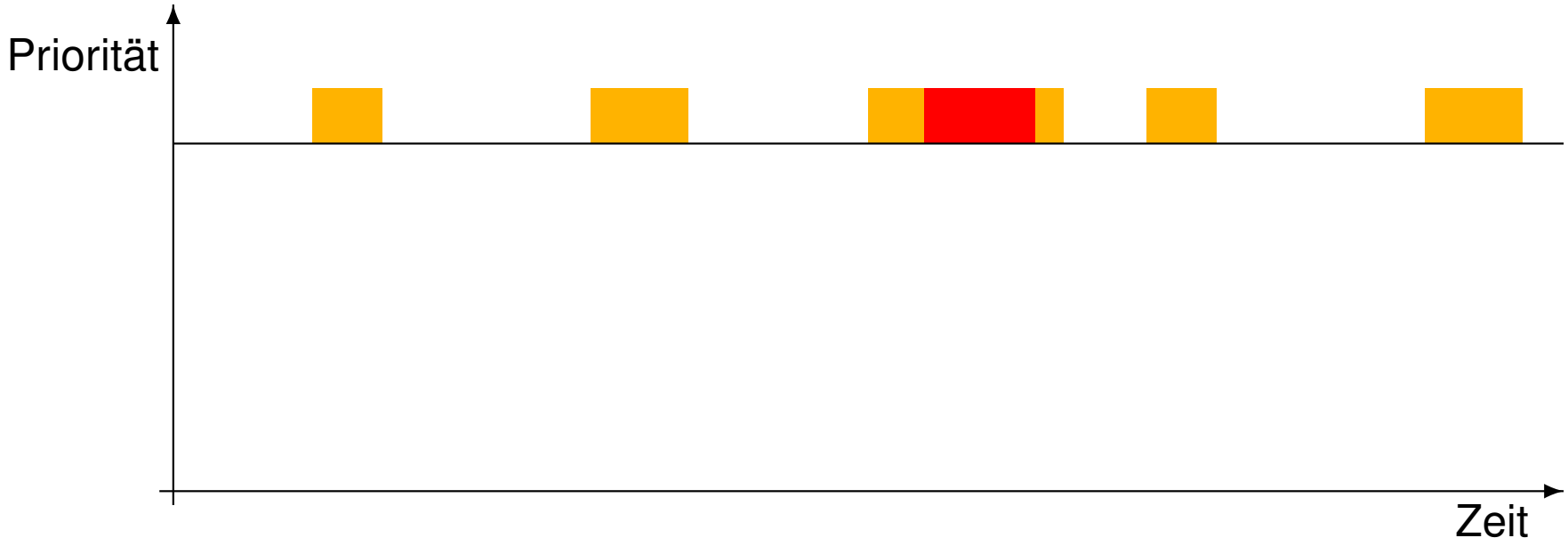
Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität
  Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.



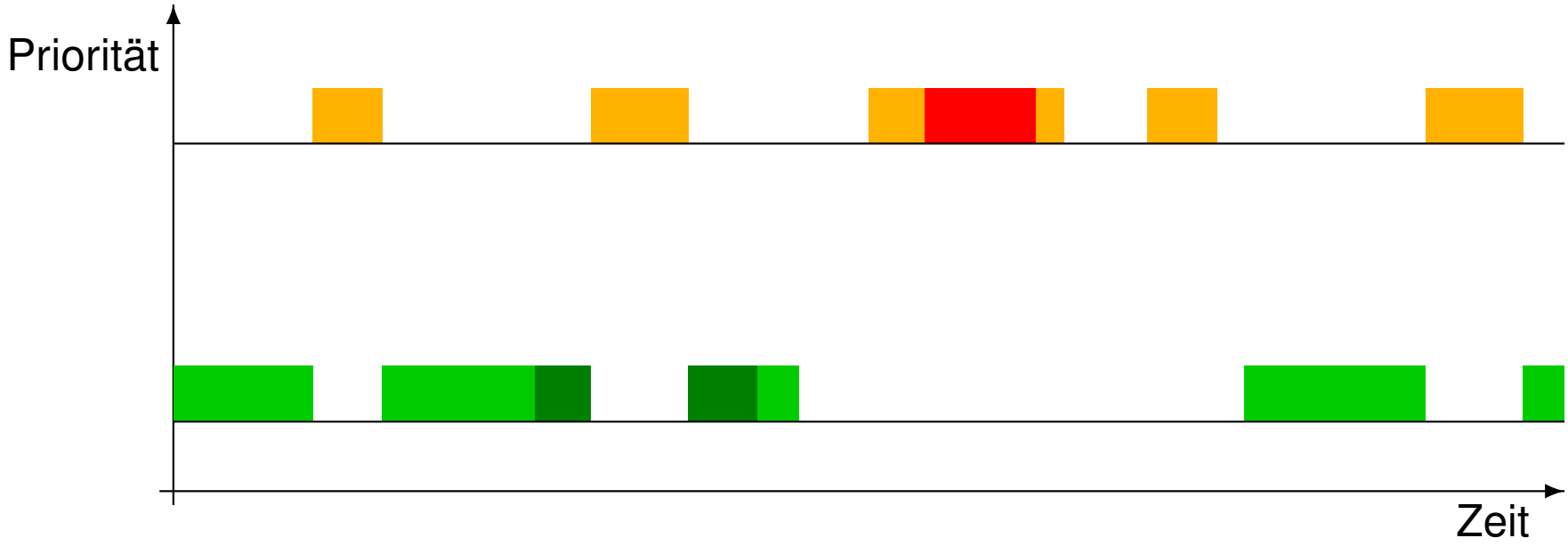
2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen



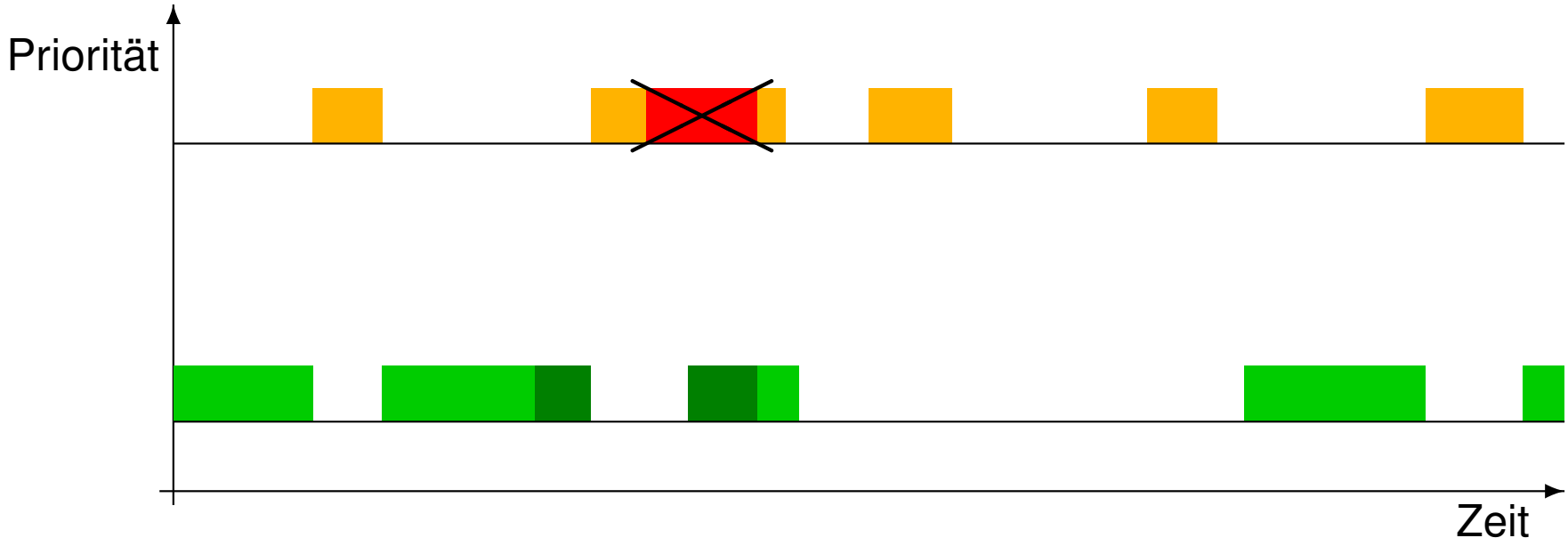
2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen



2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen



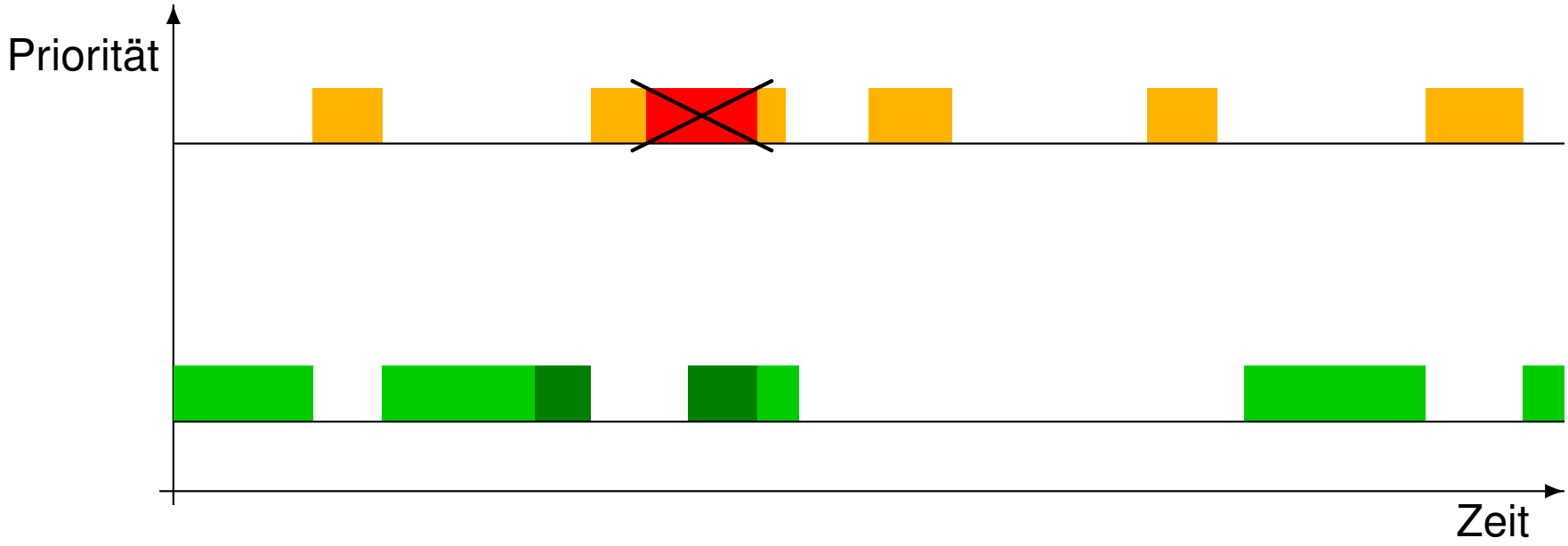
2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst,
daß er eine Ressource verspätet bekommt.

$\longrightarrow$ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



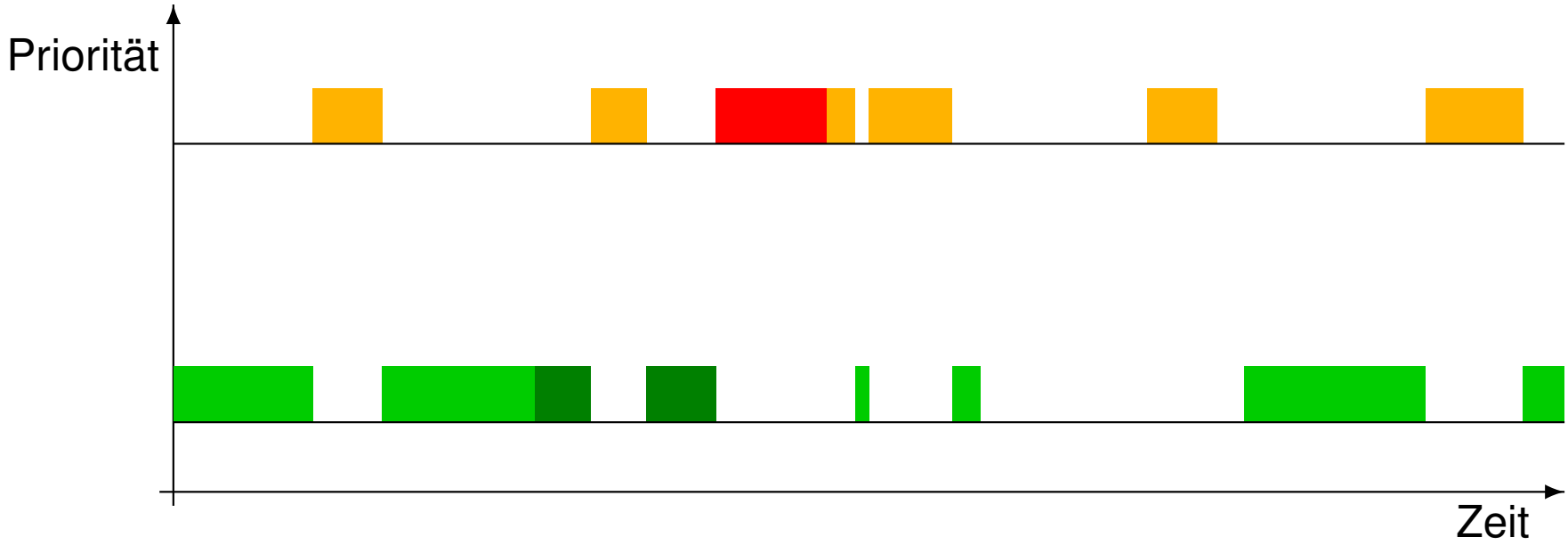
2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst,
daß er eine Ressource verspätet bekommt.

$\longrightarrow$ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



2 Theorie der Echtzeitsysteme

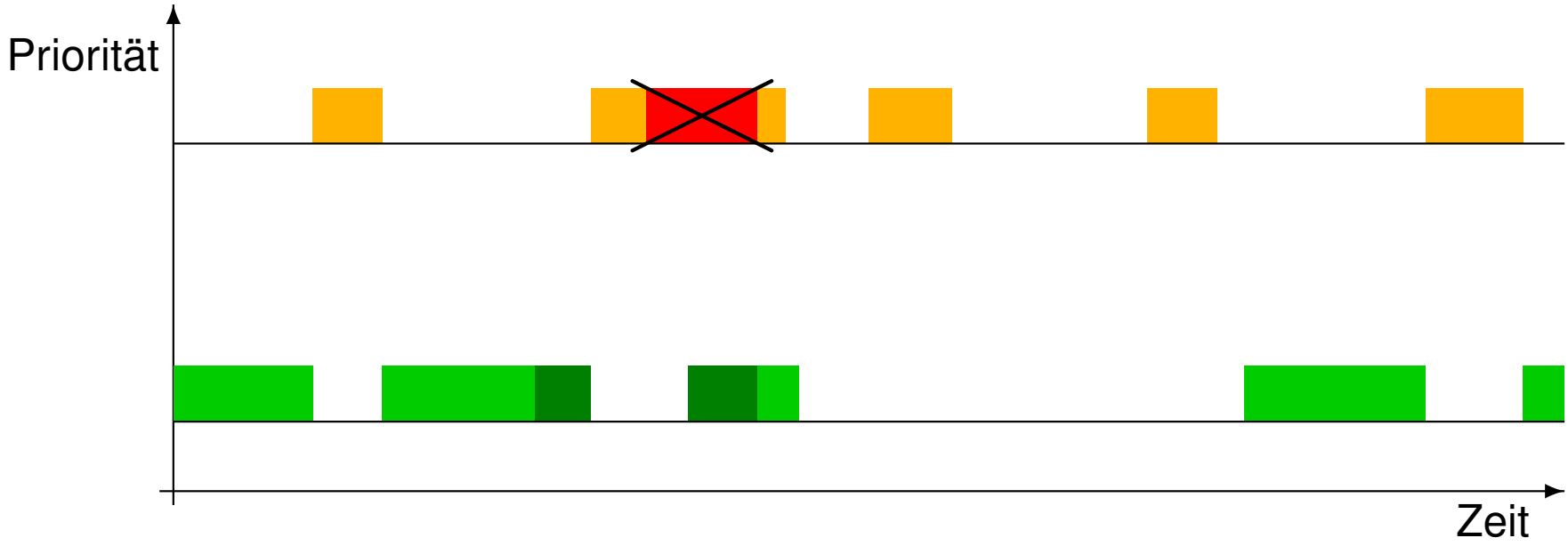
2.3 Prioritäten und Ressourcen

unbegrenzte Prioritätsinversion

2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

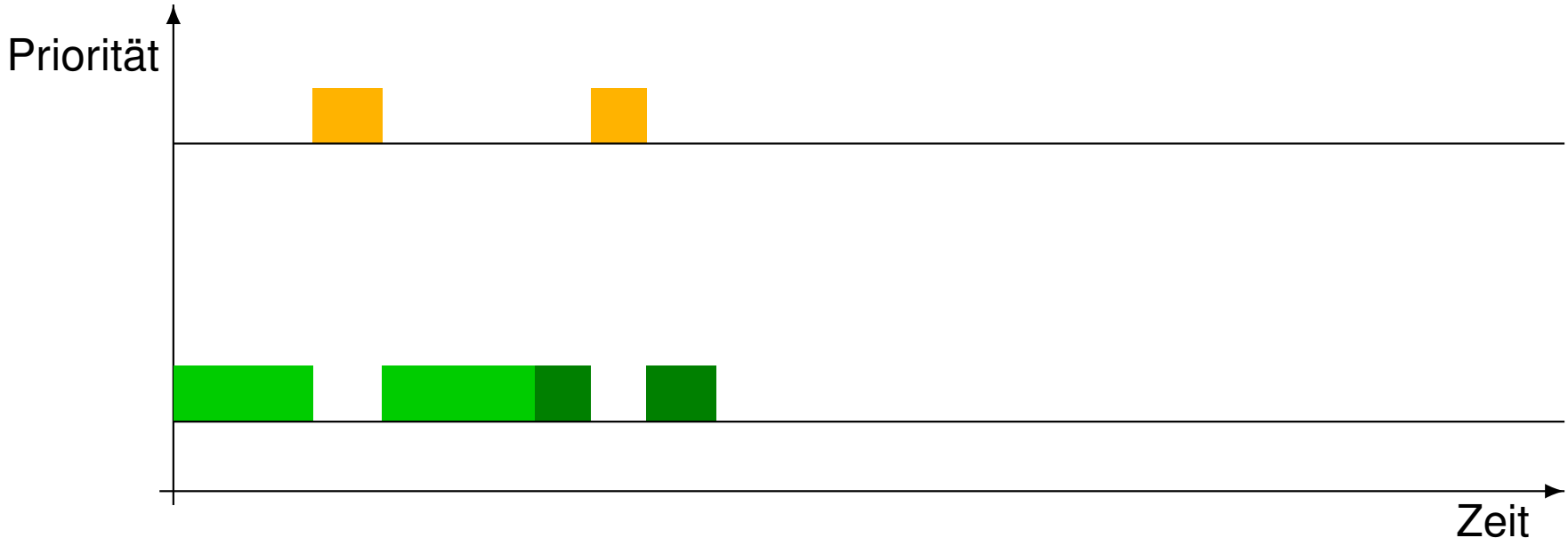
unbegrenzte Prioritätsinversion



2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

unbegrenzte Prioritätsinversion

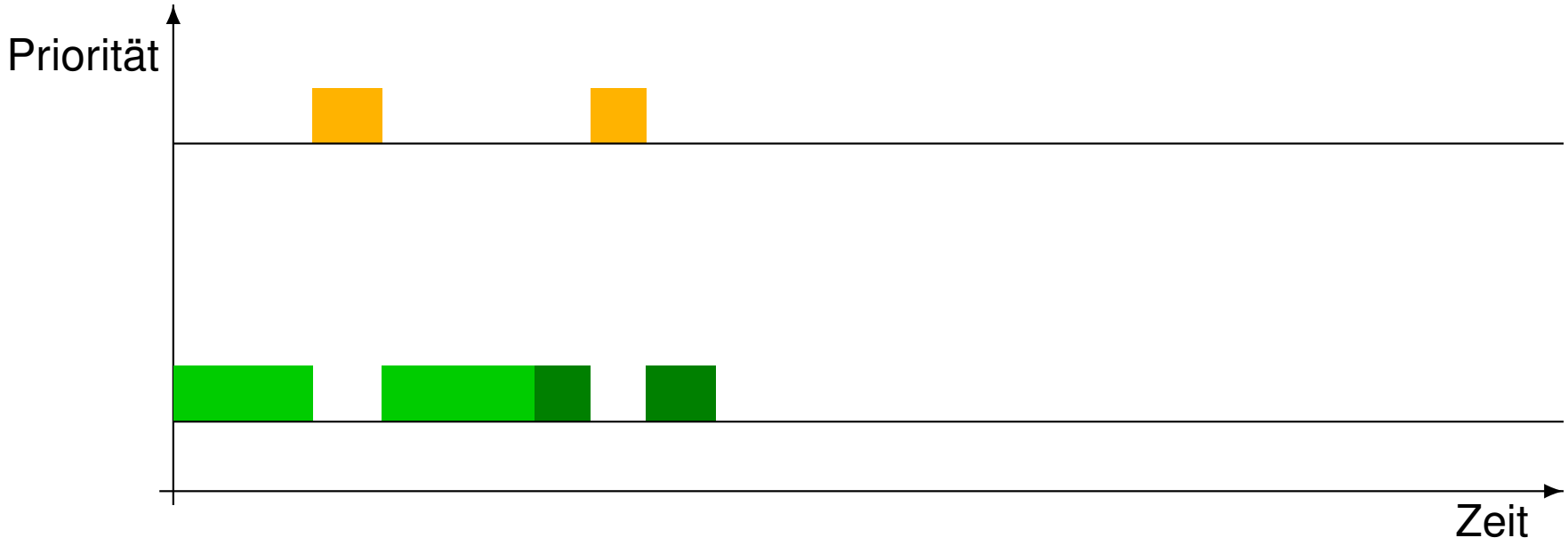


2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

$\longrightarrow$ *unbegrenzte Prioritätsinversion*

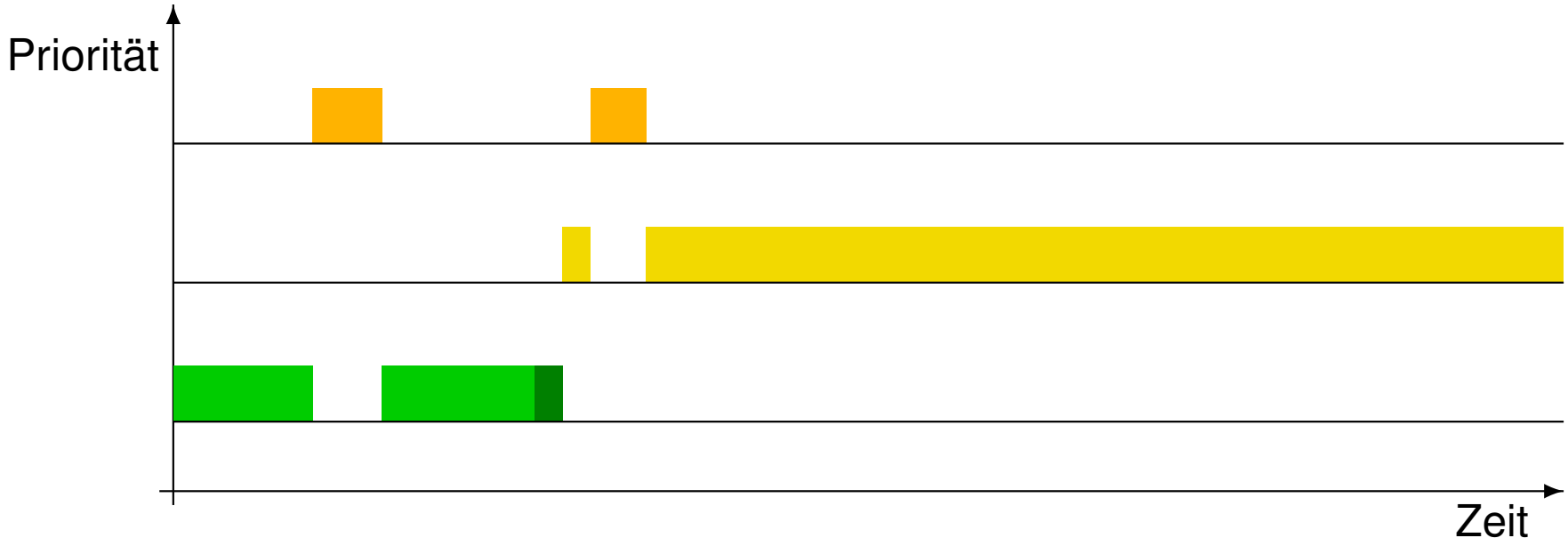


2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

$\longrightarrow$ *unbegrenzte Prioritätsinversion*



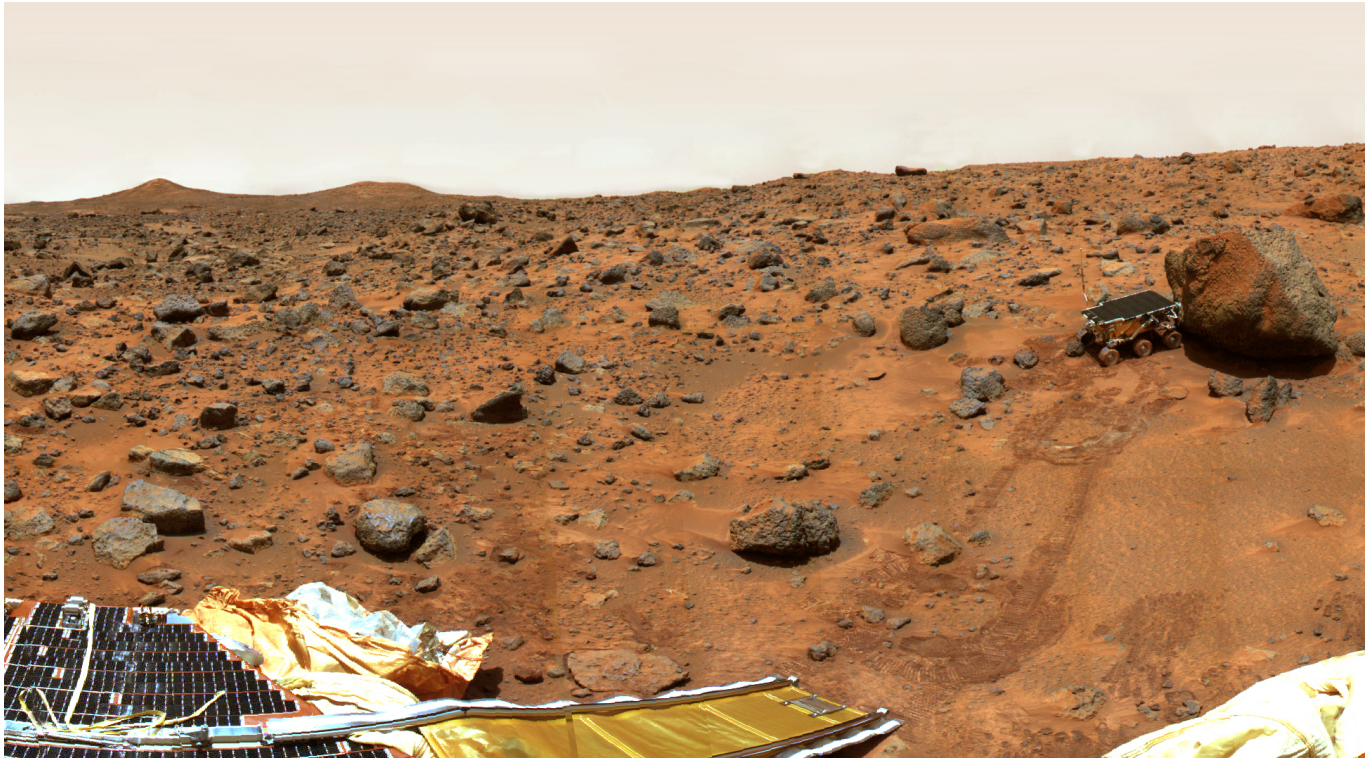
2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

$\longrightarrow$ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



2 Theorie der Echtzeitsysteme

2.3 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

$\longrightarrow$ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*
  Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*
  Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

} nur möglich, wenn Mutexe im Spiel sind

- *Priority Aging*
  Die Priorität wächst mit der Wartezeit.

3 Echtzeitbetriebssysteme

3.1 Definition

Wikipedia:

> *Ein Echtzeitbetriebssystem (englisch real-time operating system, kurz RTOS genannt) ist ein Betriebssystem mit zusätzlichen Echtzeit-Funktionen für die unbedingte Einhaltung von Zeitbedingungen und die Vorhersagbarkeit des Prozessverhaltens (hartes Echtzeitverhalten).*

3 Echtzeitbetriebssysteme

3.1 Definition

Wikipedia:

> *Ein Echtzeitbetriebssystem (englisch real-time operating system, kurz RTOS genannt) ist ein Betriebssystem mit zusätzlichen Echtzeit-Funktionen für die unbedingte Einhaltung von Zeitbedingungen und die Vorhersagbarkeit des Prozessverhaltens (hartes Echtzeitverhalten).*

$$\text{Echtzeitbetriebssystem} = \text{Betriebssystem} + \begin{pmatrix} \text{Semaphoren} \\ \text{Mutexe} \\ \text{Spinlocks} \\ \text{Prioritätsvererbung} \\ \text{Prioritätsobergrenze} \\ \text{Prioritätsalterung} \end{pmatrix}$$

3 Echtzeitbetriebssysteme

3.2 Beispiele

Name	Lizenz	kompatibel zu	Besonderheiten
TinyOS	BSD	–	kooperativ, eigene Sprache
FreeRTOS	GPL	–	
MicroC/OS-II	prop.	–	
eCos	GPL	Unix	stark konfigurierbar
QNX	prop.	Unix	
VxWorks	prop.	Unix	
RTAI	GPL	Unix (Linux)	Linux als Hintergrundprozeß
RT_PREEMPT	GPL	Unix (Linux)	angepaßter Kernel
Windows CE	prop.	MS-Windows	