



Scheduling

Zeitplanen periodischer Prozesse



Zeitplanung periodischer Prozesse

- Annahmen für präemptives Scheduling
 - Alle Prozesse treten periodisch mit einer Frequenz f_i auf.
 - Die Frist eines Prozesses entspricht dem nächsten Startpunkt.
 - Sind die maximalen Ausführungszeiten e_i bekannt, so kann leicht errechnet werden, ob ein ausführbarer Plan existiert.
 - Die für einen Prozesswechsel benötigten Zeiten sind vernachlässigbar.
 - Alle Prozesse sind unabhängig.
- Eine sehr gute Zusammenfassung zu dem Thema Zeitplanung periodischer Prozesse liefert Giorgio C. Buttazzo in seinem Paper „Rate Monotonic vs. EDF: Judgement Day“ (<http://www.cas.mcmaster.ca/~downd/rtsj05-rmedf.pdf>).



Einplanbarkeit

- Eine notwendige Bedingung zur Einplanbarkeit ist die Last:
 - Last eines einzelnen Prozesses: $\rho_i = e_i \cdot f_i$
 - Gesamte Auslastung bei n Prozessen:

$$\rho = \sum_{i=0}^n \rho_i$$

- Bei m Prozessoren ist $\rho < m$ eine notwendige aber nicht ausreichende Bedingung.

Zeitplanen nach Fristen

- **Ausgangspunkt:** Wir betrachten Systeme mit einem Prozessor und Fristen der Prozesse, die relativ zum Bereitzeitpunkt deren Perioden entsprechen, also $d_i = 1/f_i$.
- **Aussage:** Die Einplanung nach Fristen ist optimal.
- **Beweisidee:** Vor dem Verletzen einer Frist ist die CPU nie unbeschäftigt \Rightarrow die maximale Auslastung liegt bei 100%.
- Leider wird aufgrund von diversen Vorurteilen EDF selten benutzt.
- Betriebssysteme unterstützen selten ein EDF-Scheduling \Rightarrow Die Implementierung eines EDF-Scheduler auf der Basis von einem prioritätsbasierten Scheduler ist nicht effizient zu implementieren (Ausnahme: zeitgesteuerte Systeme)

Zeitplanung nach Raten

- Rate Monotonic bezeichnet ein Scheduling-Verfahren mit festen Prioritäten $Prio(i)$, die sich proportional zu den Frequenzen verhalten.
⇒ Prozesse mit hohen Raten werden bevorzugt. Das Verfahren ist optimal, falls eine Lösung mit statischen Prioritäten existiert. Verfahren mit dynamischen Prioritäten können allerdings eventuell bessere Ergebnisse liefern.
- Liu und Layland haben 1973 in einer Worst-Case-Analyse gezeigt, dass Ratenplanung sicher erfolgreich ist, falls bei n Prozessen auf einem Prozessor gilt:

$$\rho \leq \rho_{\max} = n \cdot (2^{1/n} - 1)$$

$$\lim_{n \rightarrow \infty} \rho_{\max} = \ln 2 \approx 0,69$$

- Derzeit zumeist verwendetes Scheduling-Verfahren im Bereich von periodischen Prozessen.



Scheduling

Planen abhängiger Prozesse

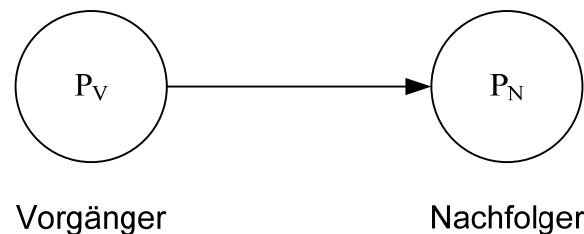


Allgemeines zum Scheduling in Echtzeitsystemen

- Grundsätzlich kann der Prozessor neu vergeben werden, falls:
 - ein Prozess endet,
 - ein Prozess in den blockierten Zustand (z.B. wegen Anforderung eines blockierten Betriebsmittels) wechselt,
 - eine neuer Prozess gestartet wird,
 - ein Prozess vom blockierten Zustand in den Wartezustand wechselt (z.B. durch die Freigabe eines angeforderten Betriebsmittels durch einen anderen Prozess)
 - oder nach dem Ablauf eines Zeitintervalls, siehe z.B. Round Robin.
- Hochpriorisierte Prozesse dürfen in Echtzeitsystemen nicht durch unwichtigere Prozesse behindert werden \Rightarrow Die Prioritätsreihenfolge muss bei allen Betriebsmitteln (CPU, Semaphore, Netzkommunikation, Puffer, Peripherie) eingehalten werden, d.h. Vordrängen in allen Warteschlangen.

Präzedenzsysteme

- Zur Vereinfachung werden zunächst Systeme betrachtet, bei denen die Bereitzeiten der Prozesse auch abhängig von der Beendigung anderer Prozesse sein können.
- Mit Hilfe von Präzedenzsystemen können solche Folgen von voneinander abhängigen Prozessen beschrieben werden.
- Zur Beschreibung werden typischerweise Graphen verwendet:



- Der Nachfolgerprozess kann also frühestens beim Erreichen der eigenen Bereitzeit **und** der Beendigung der Ausführung des Vorgängerprozesses ausgeführt werden.



Probleme bei Präzedenzsystemen

- Bei der Planung mit Präzedenzsystemen muss auch berücksichtigt werden, dass die Folgeprozesse noch rechtzeitig beendet werden können.
- Beispiel:
 $P_V: r_V=0; e_V=1; d_V=3;$
 $P_N: r_N=0; e_N=3; d_N=5;$
- Falls die Frist von P_V voll ausgenutzt wird, kann der Prozess P_N nicht mehr rechtzeitig beendet werden.
⇒ Die Fristen müssen entsprechend den Prozessabhängigkeiten neu berechnet werden (Normalisierung von Präzedenzsystemen).

Normalisierung von Präzedenzsystemen

- Anstelle des ursprünglichen Präzedenzsystems PS wird ein normalisiertes Präzedenzsystem PS' mit folgenden Eigenschaften:

- $\forall i: e'_i = e_i$

- $\forall i : d'_i = \begin{cases} d_i, & \text{falls } N_i = \emptyset \\ \min(d_i, \min(d'_q - e'_q | q \in N_i)) \end{cases}$

wobei N_i die Menge der Nachfolger im Präzedenzgraph bezeichnet und d'_i rekursiv beginnend bei Prozessen ohne Nachfolger berechnet wird.

- Falls die Bereitzeiten von externen Ereignissen abhängig sind, gilt $r'_i = r_i$. Sind die Bereitzeiten dagegen abhängig von der Beendigung der Prozesse, so ergeben sie sich aus dem konkreten Scheduling.

eingeführt.

⇒ Ein Präzedenzsystem ist nur dann planbar, falls das zugehörige normalisierte Präzedenzsystem planbar ist.

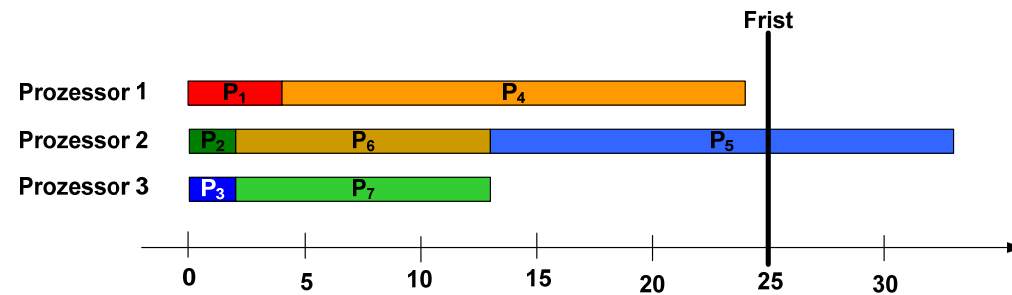
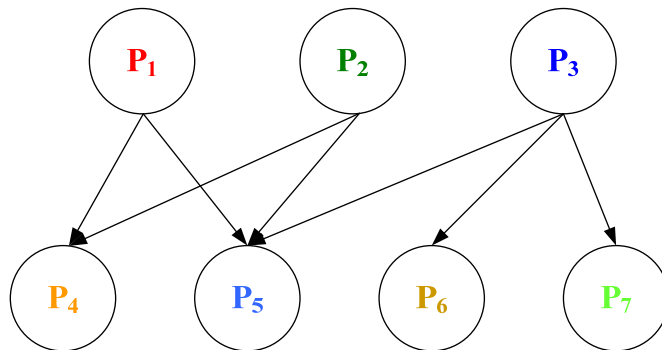


Anomalien bei nicht präemptiven Scheduling

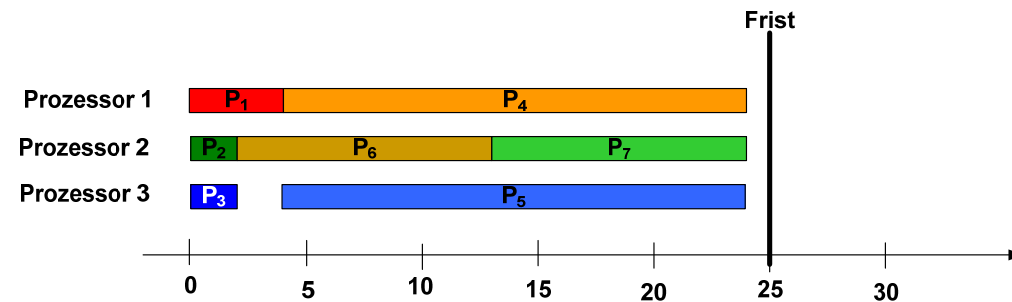
- Wird zum Scheduling von Präzedenzsystemen ein nicht präemptives prioritätenbasiertes Verfahren (z.B. EDF, LST) verwendet, so können Anomalien auftreten:
 - Durch Hinzufügen eines Prozessors kann sich die gesamte Ausführungszeit verlängern.
 - Durch freiwilliges Warten kann die gesamte Ausführungszeit verkürzt werden.

Beispiel: Verkürzung durch freiwilliges Warten

- Beispiel: 3 Prozessoren, 7 Prozesse ($r_i=0$, $e_1=4$; $e_2=2$; $e_3=2$; $e_4=20$; $e_5=20$; $e_6=11$; $e_7=11$, $d_i=25$), Präzedenzgraph:



Prioritätenbasiertes Scheduling

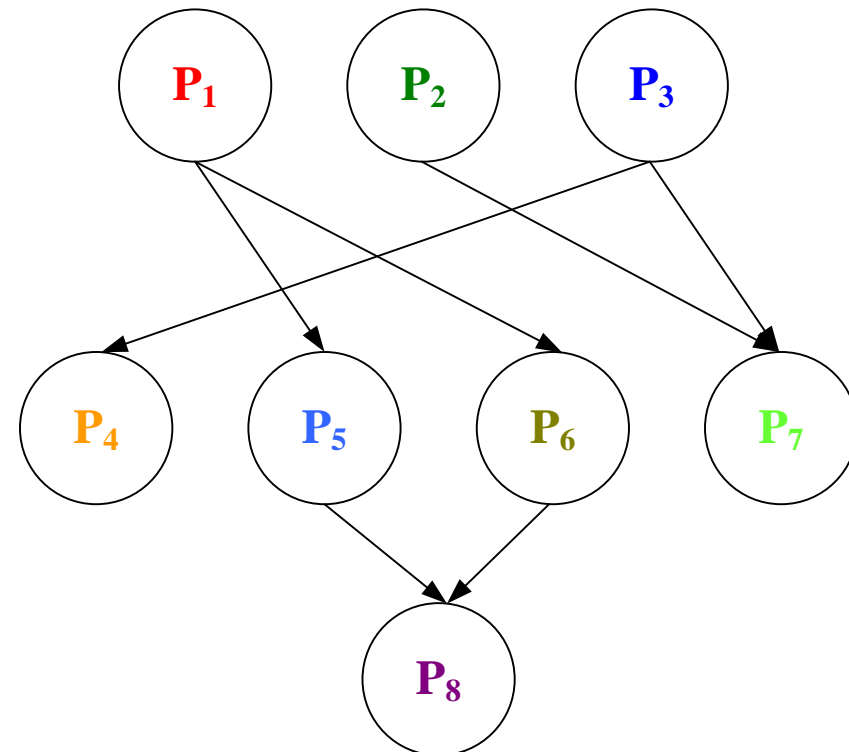


Optimaler Plan

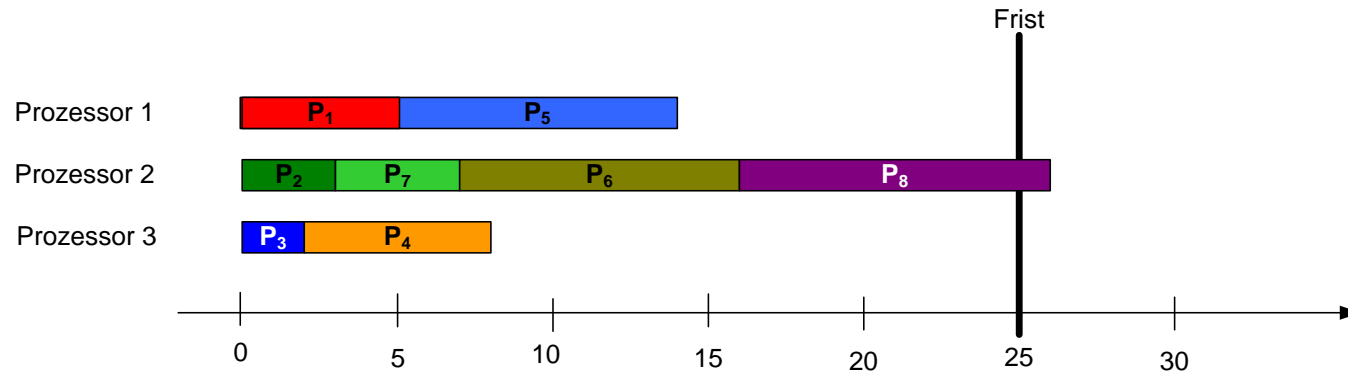
Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II

- Beispiel:
 - 2 bzw. 3 Prozessoren
 - 8 Prozesse:
 - Startzeiten $r_i=0$
 - Ausführungszeiten
 - $e_1=5$;
 - $e_2=3$;
 - $e_3=2$;
 - $e_4=6$;
 - $e_5=9$;
 - $e_6=9$;
 - $e_7=4$;
 - $e_8=10$
 - Frist: $d_i=25$

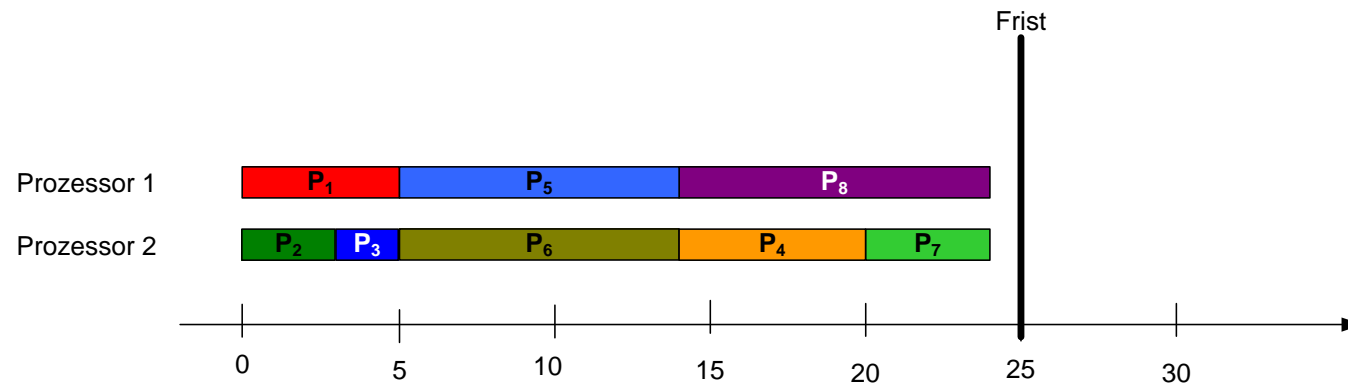
- Präzedenzgraph:



Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II



Prioritätenbasiertes Scheduling (LST) auf 3 Prozessoren



Prioritätenbasiertes Scheduling (LST) auf 2 Prozessoren



Scheduling

Problem: Prioritätsinversion



Motivation des Problems

- Selbst auf einem Einprozessoren-System mit präemptiven Scheduling gibt es Probleme bei voneinander abhängigen Prozessen.
- Abhängigkeiten können diverse Gründe haben:
 - Prozesse benötigen Ergebnisse eines anderen Prozesses
 - Betriebsmittel werden geteilt
 - Es existieren kritische Bereiche, die durch Semaphoren oder Monitoren geschützt sind.
- Gerade aus den letzten zwei Punkten entstehen einige Probleme:
 - Die Prozesse werden unter Umständen unabhängig voneinander implementiert
⇒ das Verhalten des anderen Prozesses ist nicht bekannt.
 - Bisher haben wir noch keinen Mechanismus zum Umgang mit blockierten Betriebsmitteln kennengelernt, falls hochpriorie Prozesse diese Betriebsmittel anfordern.

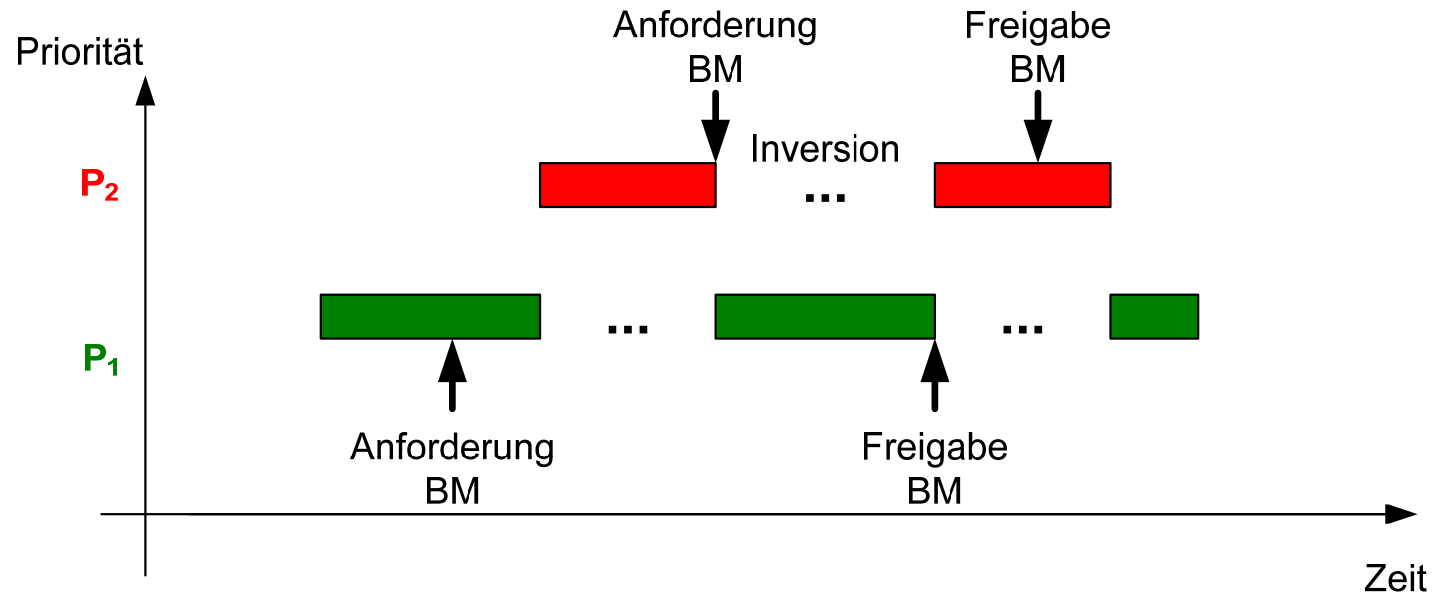


Prioritätsinversion

- **Definition:** Das Problem der **Prioritätsinversion** bezeichnet Situationen, in denen ein Prozess mit niedriger Priorität einen höherpriorisierten Prozess blockiert.
- Dabei unterscheidet man zwei Arten der Prioritätsinversion:
 - **begrenzte** (bounded) Prioritätsinversion: die Inversion ist durch die Dauer des kritischen Bereichs beschränkt.
 - **unbegrenzte** (unbounded) Prioritätsinversion: durch weitere Prozesse kann der hochpriorisierte Prozess auf unbestimmte Dauer blockiert werden.
- Während das Problem der begrenzten Prioritätsinversion aufgrund der begrenzten Zeitdauer akzeptiert werden kann (muss), ist die unbegrenzte Prioritätsinversion in Echtzeitsystemen unbedingt zu vermeiden.

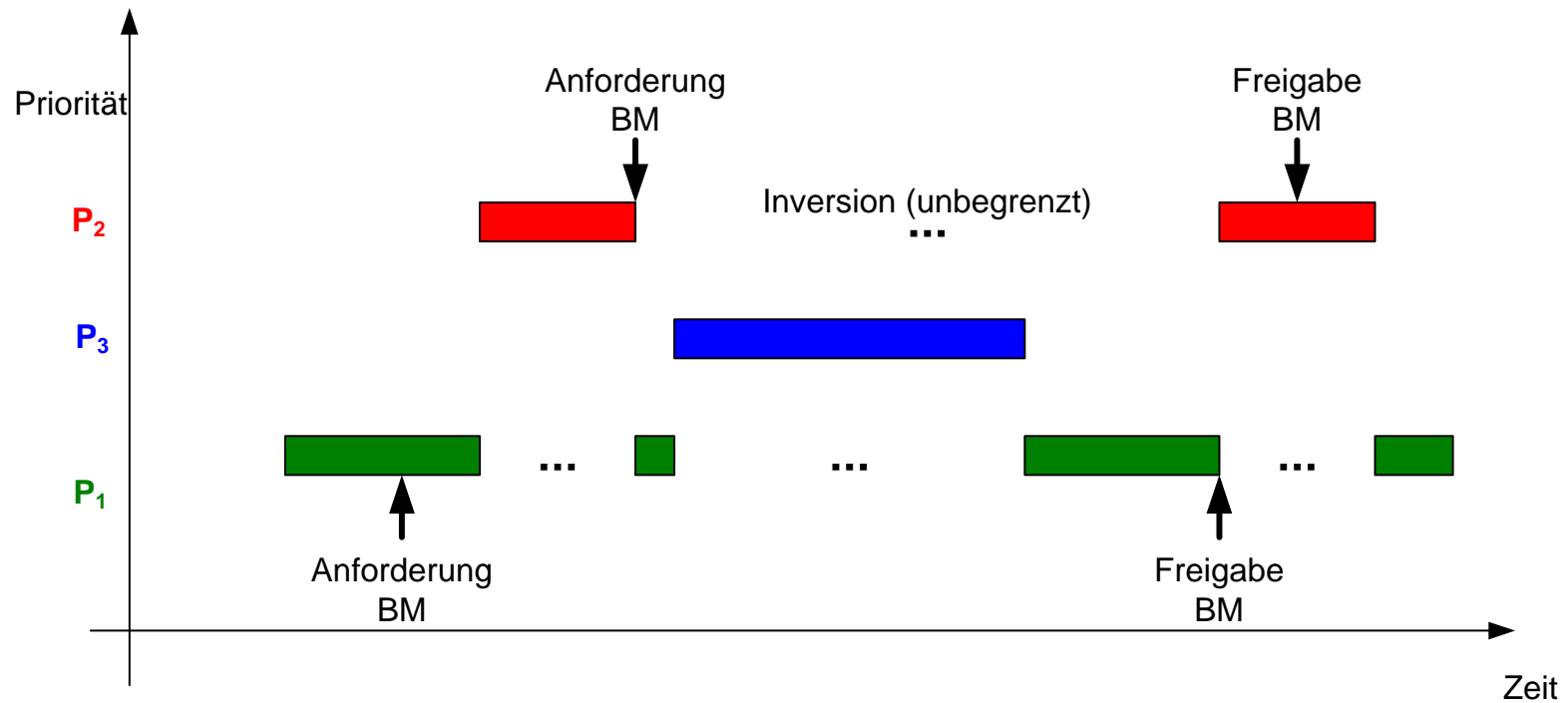


Begrenzte Inversion





Unbegrenzte Inversion



Reales Beispiel: Mars Pathfinder

- **System:** Der Mars Pathfinder hatte zur Speicherung der Daten einen Informationsbus (vergleichbar mit Shared Memory). Der Informationsbus war durch einen binären Semaphore geschützt. Ein Bus Management Prozess verwaltete den Bus mit hoher Priorität. Ein weiterer Prozess war für die Sammlung von geologischen Daten eingeplant. Dieser Task lief mit einer niedrigen Priorität. Zusätzlich gab es noch einen Kommunikationsprozess mittlerer Priorität.
- **Symptome:** Das System führte in unregelmäßigen Abständen einen Neustart durch. Daten gingen dadurch verloren.
- **Ursache:** Der binäre Semaphore war nicht mit dem Merkmal zur Unterstützung von Prioritätsvererbung (siehe später) erzeugt worden. Dadurch kam es zur Prioritätsinversion. Ein Watchdog (Timer) erkannte eine unzulässige Verzögerung des Bus Management Prozesses und führte aufgrund eines gravierenden Fehlers einen Neustart durch.





Ansätze zur Lösung der Prioritätsinversion

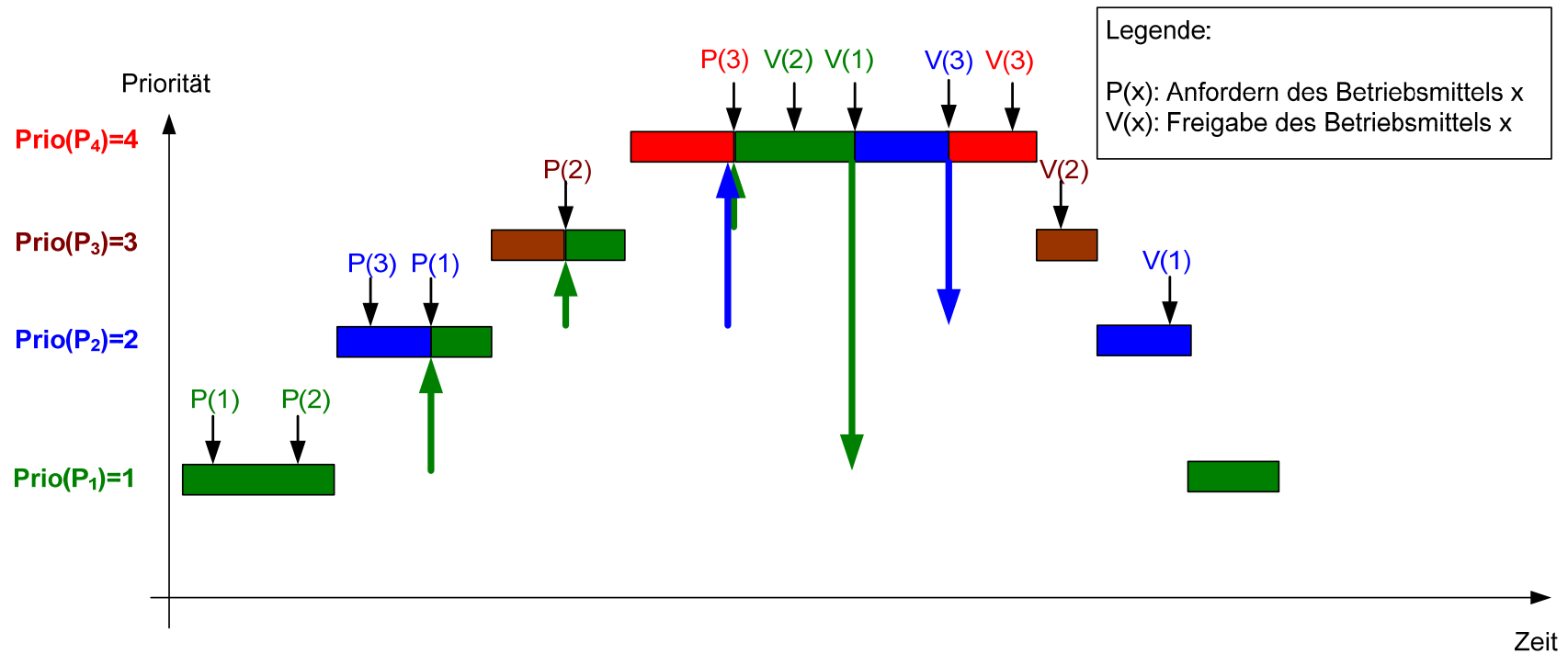
- Es existieren verschiedene Ansätze um das Problem der unbegrenzten Prioritätsinversion zu begrenzen:
 - Prioritätsvererbung (priority inheritance)
 - Prioritätsobergrenzen (priority ceiling)
 - Unmittelbare Prioritätsobergrenzen (immediate priority ceiling)
- Anforderungen an Lösungen:
 - leicht zu implementieren
 - Anwendungsunabhängige Implementierung
 - Eventuell Ausschluss von Verklemmungen



Prioritätsvererbung (priority inheritance)

- Sobald ein Prozess höherer Priorität ein Betriebsmittel anfordert, das ein Prozess mit niedrigerer Priorität besitzt, erbt der Prozess mit niedrigerer Priorität die höhere Priorität. Nachdem das Betriebsmittel freigegeben wurde, fällt die Priorität wieder auf die ursprüngliche Priorität zurück.
 - ⇒ Unbegrenzte Prioritätsinversion wird verhindert.
 - ⇒ Die Dauer der Blockade wird durch die Dauer des kritischen Abschnittes beschränkt.
 - ⇒ Blockierungen werden hintereinander gereiht (Blockierungsketten).
 - ⇒ Verklemmungen durch Programmierfehler werden nicht verhindert.

Beispiel: Prioritätsvererbung

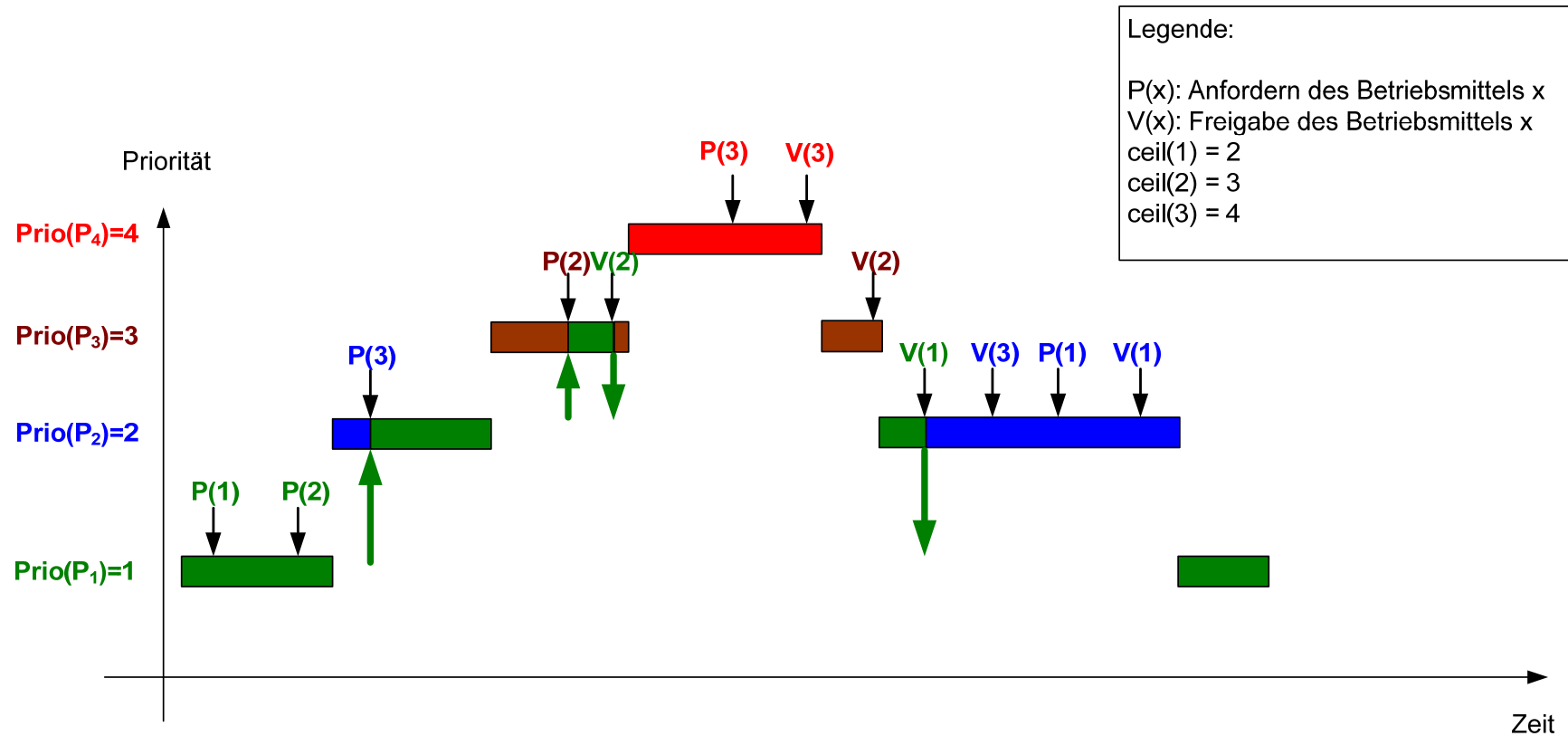




Prioritätsobergrenzen (priority ceiling)

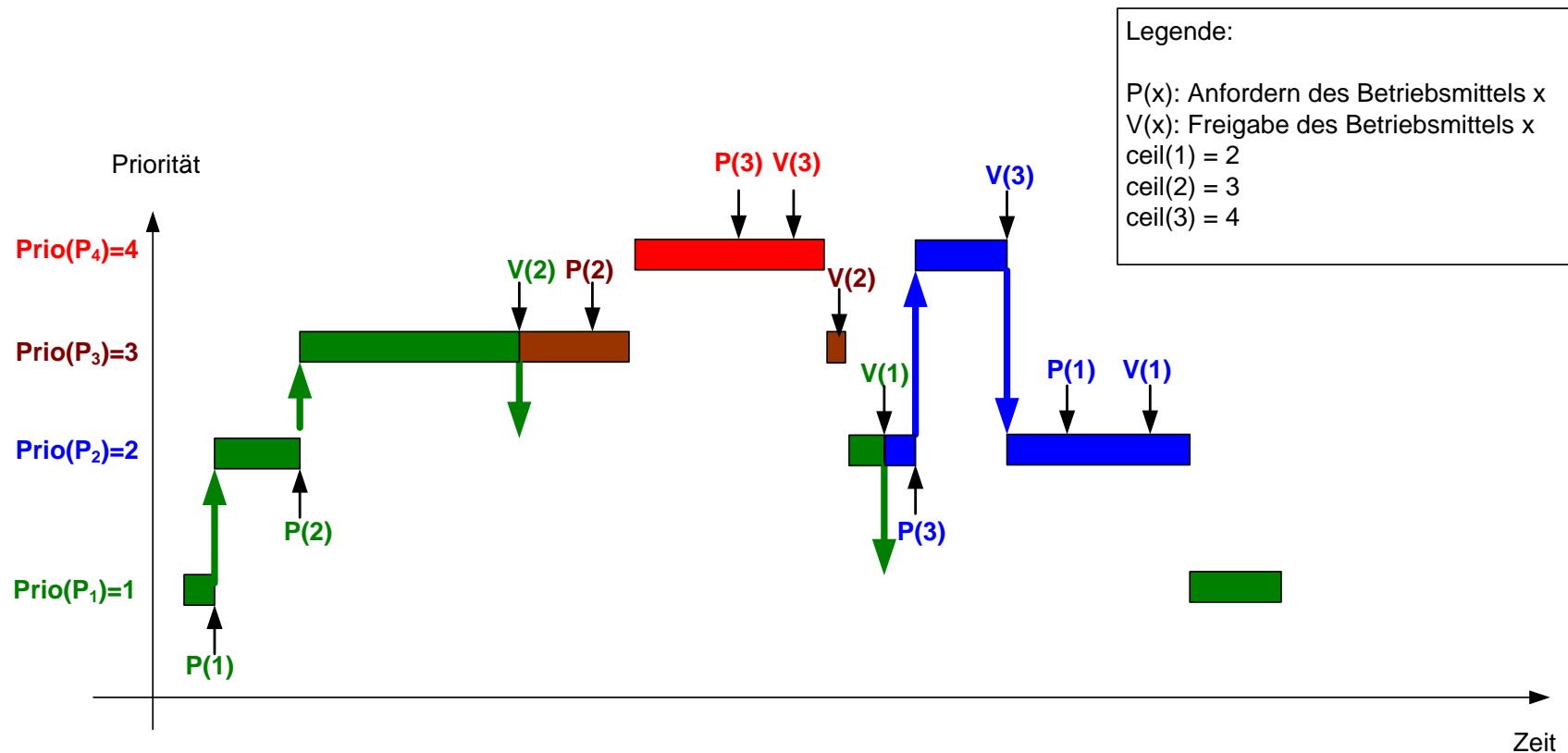
- Jedem Betriebsmittel (z.B. Semaphore) s wird eine Prioritätsgrenze $\text{ceil}(s)$ zugewiesen, diese entspricht der maximalen Priorität der Prozesse, die auf s zugreifen.
 - Ein Prozess p darf ein BM nur blockieren, wenn er von keinem anderen Prozess, der andere BM besitzt, verzögert werden kann.
 - Die aktuelle Prioritätsgrenze für Prozess p ist $\text{aktceil}(p) = \max\{\text{ceil}(s) \mid s \in \text{locked}\}$ mit locked = Menge aller von anderen Prozessen blockierten BM
 - Prozess p darf Betriebsmittel s benutzen, wenn für seine aktuelle Priorität aktprio gilt: $\text{aktprio}(p) > \text{aktceil}(p)$
 - Andernfalls gibt es genau einen Prozess, der s besitzt. Die Priorität dieses Prozesses wird auf $\text{aktprio}(p)$ gesetzt.
- ⇒ Blockierung nur für die Dauer eines kritischen Abschnitts
- ⇒ Verhindert Verklemmungen
- ⇒ schwieriger zu realisieren, zusätzlicher Prozesszustand
- Vereinfachtes Protokoll: **Immediate priority ceiling**: Prozesse, die ein Betriebsmittel s belegen, bekommen sofort die Priorität $\text{ceil}(s)$ zugewiesen.

Beispiel: Prioritätsobergrenzen



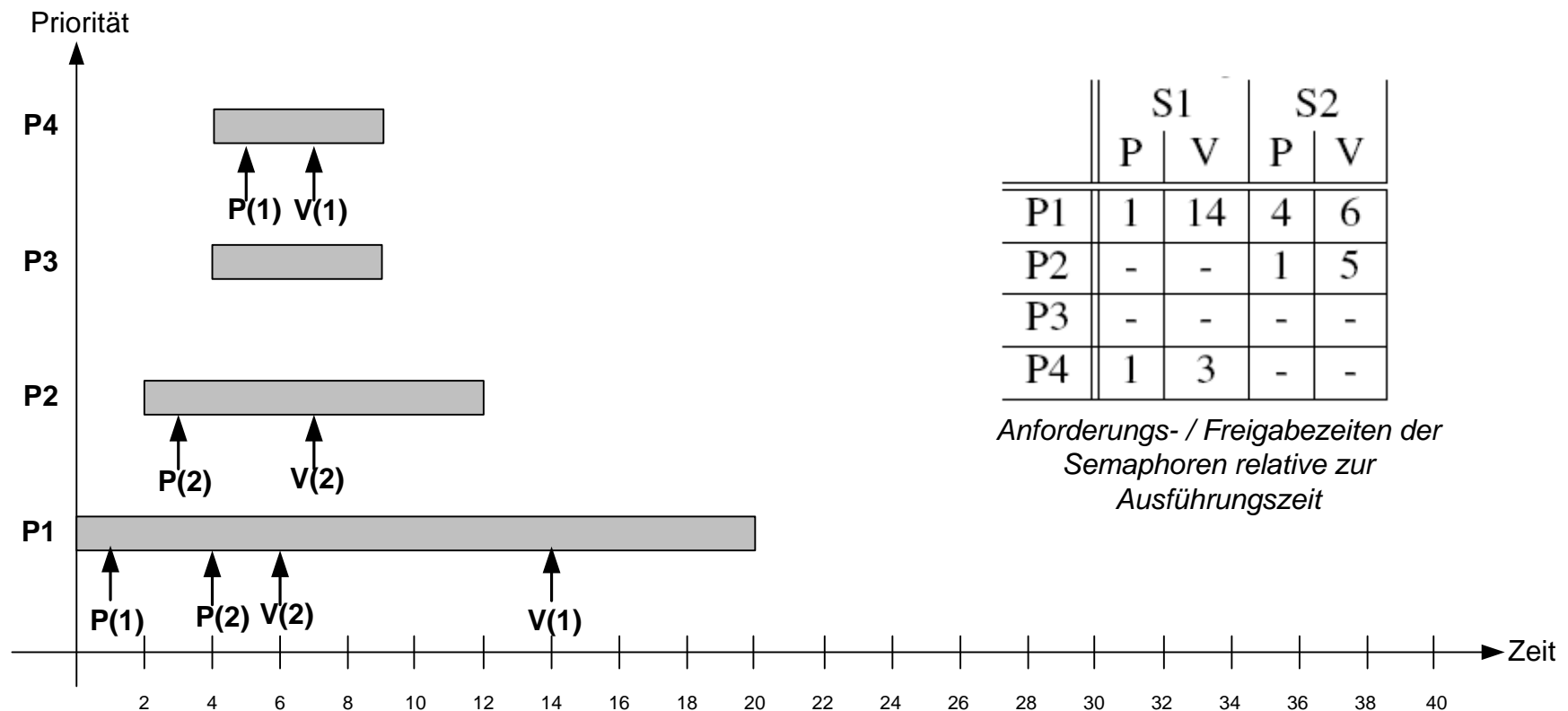


Beispiel: Immediate Priority Ceiling





Klausur WS 06/07 - Szenario



Fortsetzung – Möglicher Ausführungsplan für ein 1-Prozessor-System

