Scheduling

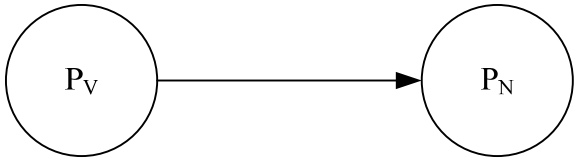
Planen abhängiger Prozesse

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 Betriebmittels durch einen anderen Prozess)
  - oder nach dem Ablauf eines Zeitintervals, 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:
  - $\forall\rangle$: $e'_i = e_i$
  - $$\forall i : d'_i = \begin{cases} d_i, falls N_i = \emptyset \\ min(d_i, min(d'_q - e'_q | q \in N_i) \end{cases}$$

    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.

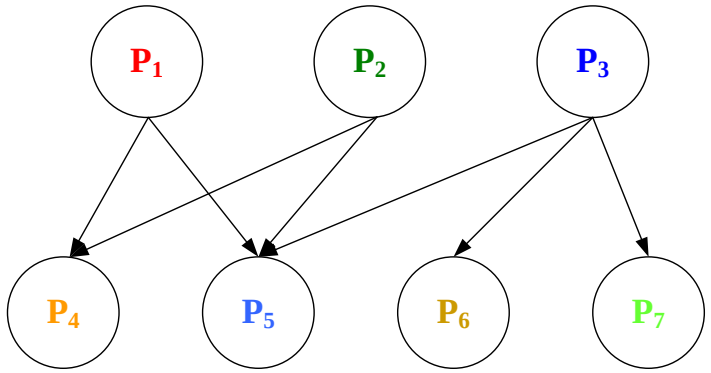
  $\rightarrow$ Ein Präzedenzsystem ist nur dann planbar, falls das zugehörige normalisierte Präzedenzsystem planbar ist.

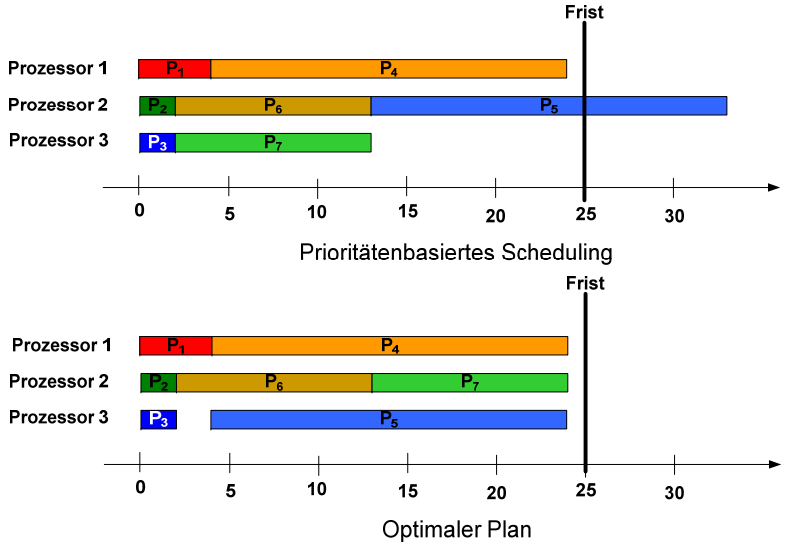
Anomalien bei nicht präemptiven Scheduling

- 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

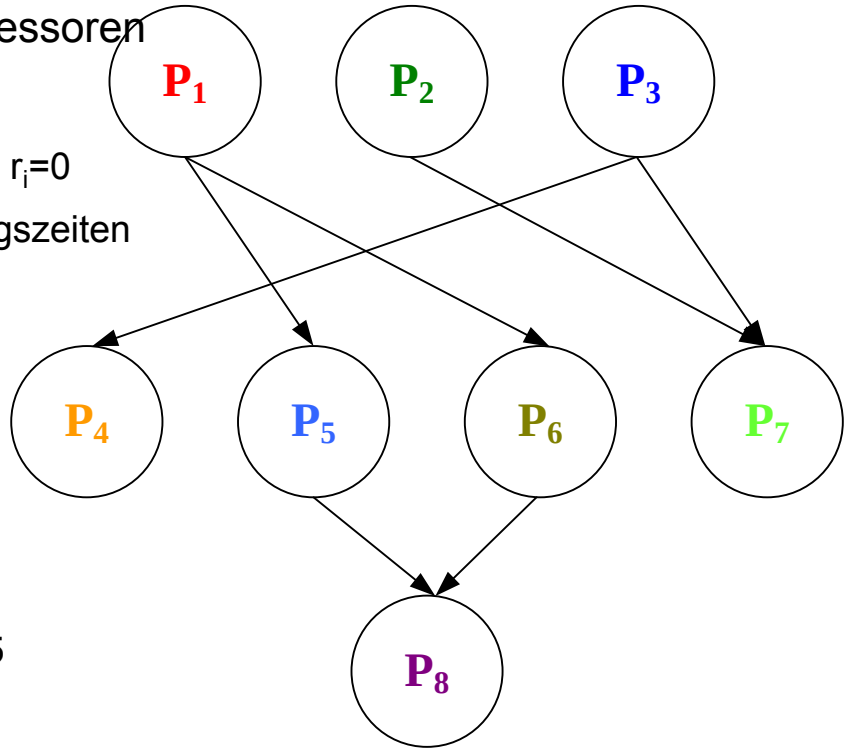
- Beispiel: 3 Prozessoren, 7 Prozesse (r_i=0, e_1=4; e_2=2; e_3=2; e_4=20; e_5=20; e_6=11; e_7=11, d_i=25), Präzedenzgraph:



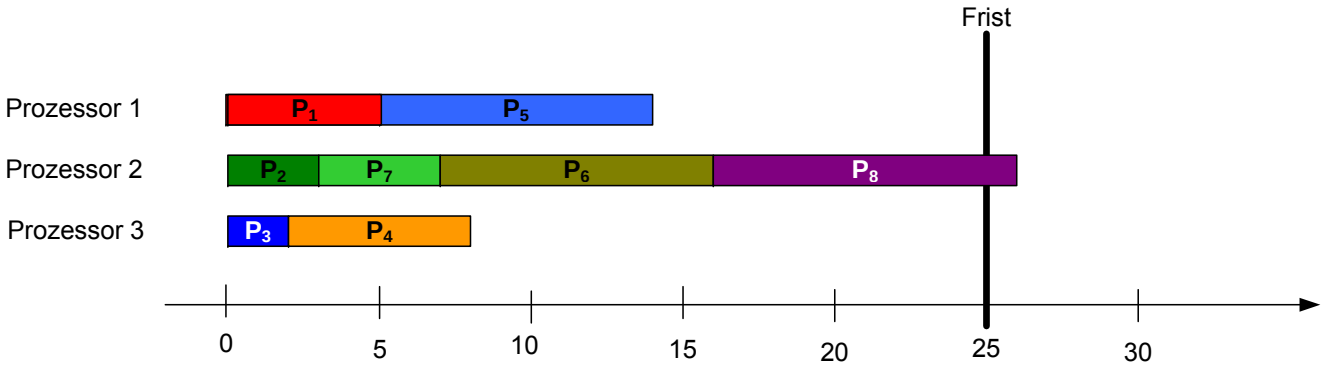


Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II

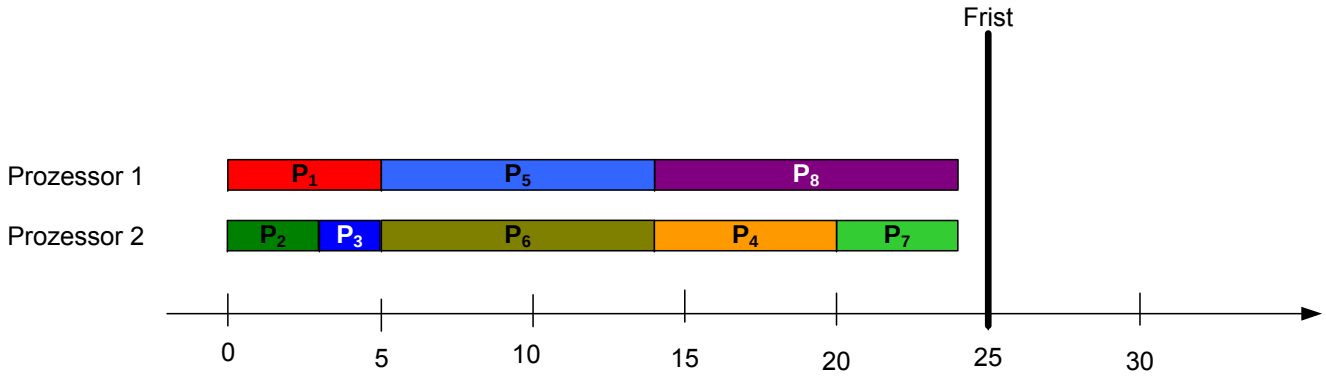
- Beispiel:
  - 2 bzw. 3 Prozessoren
  - 8 Prozesse:
    - Startzeiten $r_i=0$
    - Ausführungszeiten
      $e_1=5$;
      $e_2=3$;
      $e_3=2$;
      $e_4=6$;
      $e_5=9$;
      $e_6=9$;
      $e_7=4$,
      $e_8=10$
    - Frist: $d_i=25$
- Präzedenzgraph:



Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II



Prioritätenbasiertes Scheduling (LST) auf 3 Prozessoren



Prioritätenbasiertes Scheduling (LST) auf 2 Prozessoren

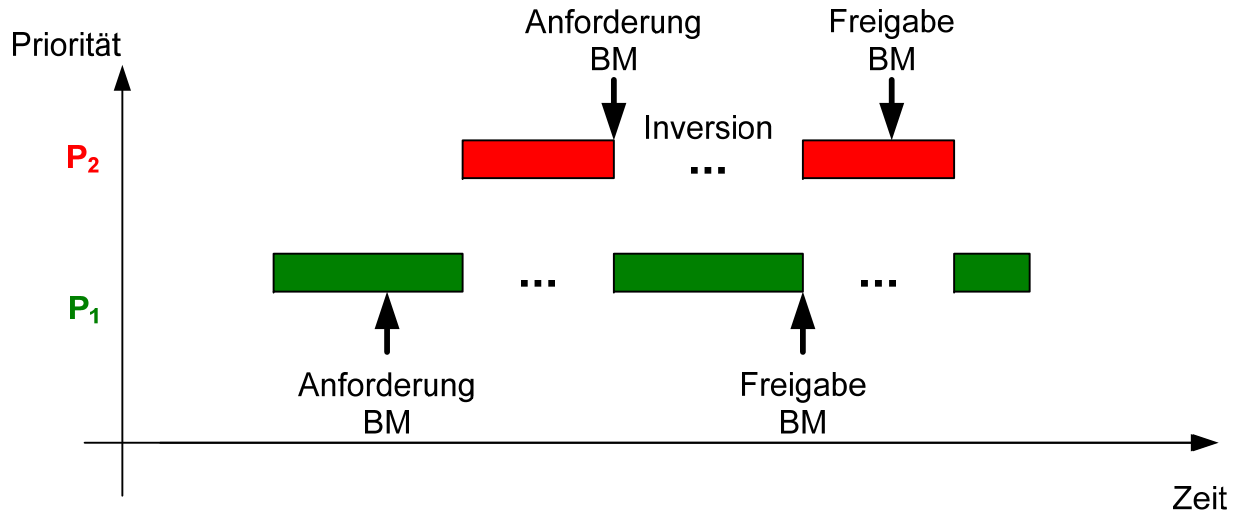
Scheduling

Problem: Prioritätsinversion

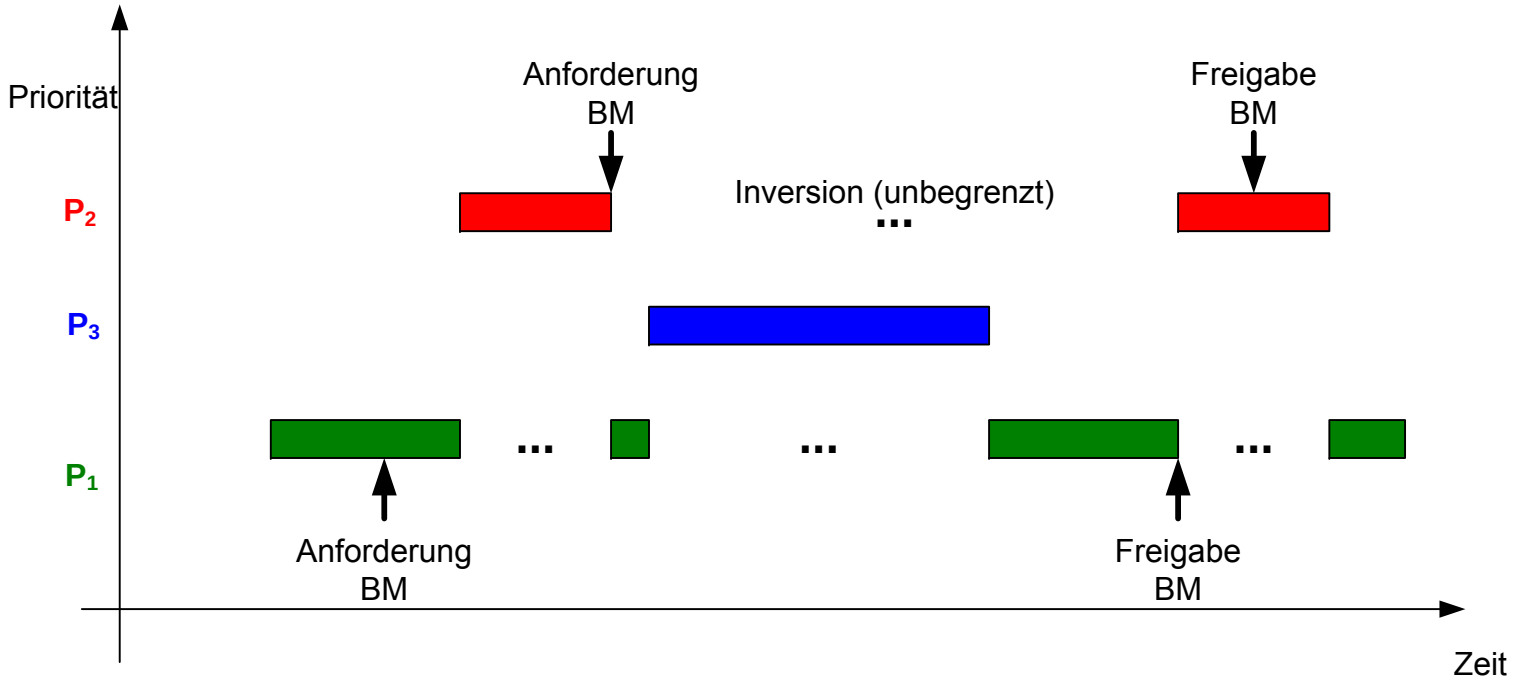
Motivation des Problems

- 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 $\Rightarrow$ das Verhalten des anderen Prozesses ist nicht bekannt.
  - Bisher haben wir noch keinen Mechanismus zum Umgang mit blockierten Betriebsmitteln kennengelernt, falls hochpriore Prozesse diese Betriebsmittel anfordern.

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 Semaphor 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.

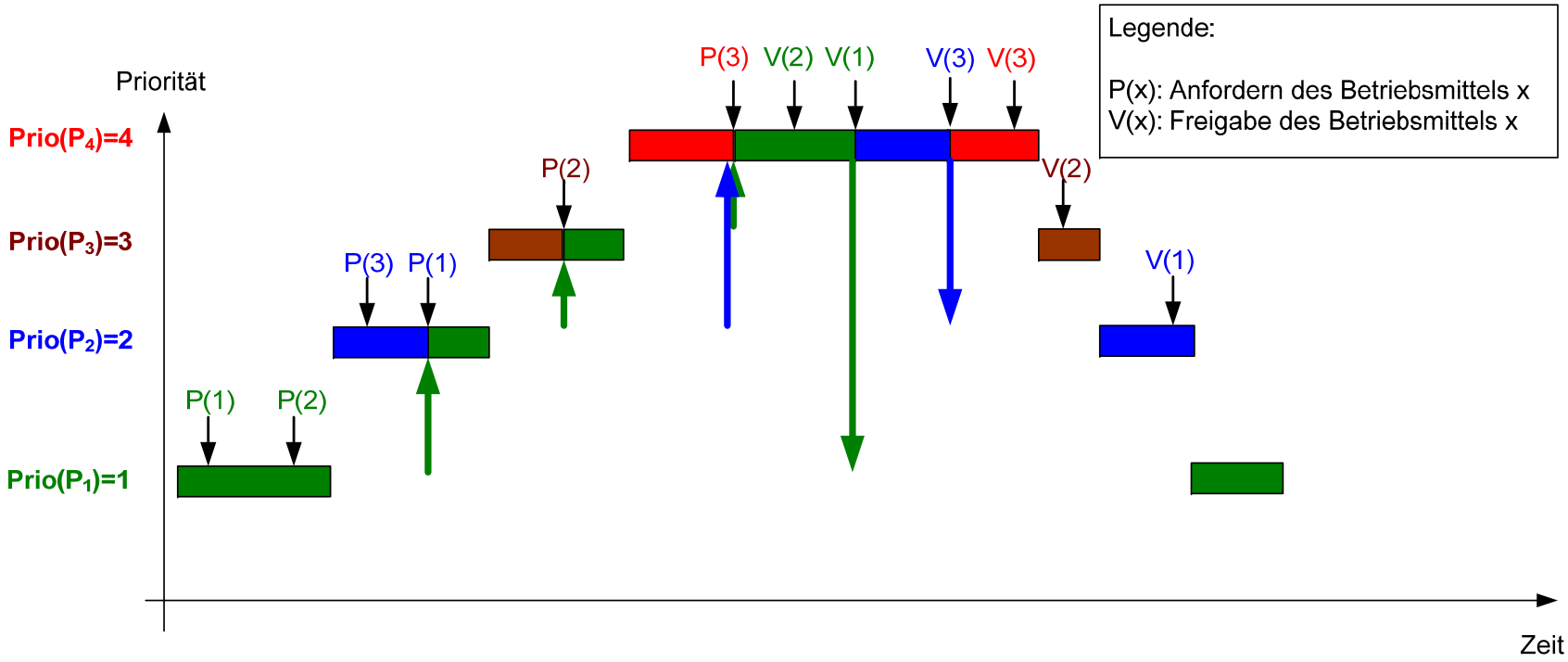
  $\rightarrow$ Unbegrenzte Prioritätsinversion wird verhindert.

  $\rightarrow$ Die Dauer der Blockade wird durch die Dauer des kritischen Abschnittes beschränkt.

  $\rightarrow$ Blockierungen werden hintereinander gereiht (Blockierungsketten).

  $\rightarrow$ Verklemmungen durch Programmierfehler werden nicht verhindert.

Beispiel: Prioritätsvererbung



Prioritätsobergrenzen (priority ceiling)

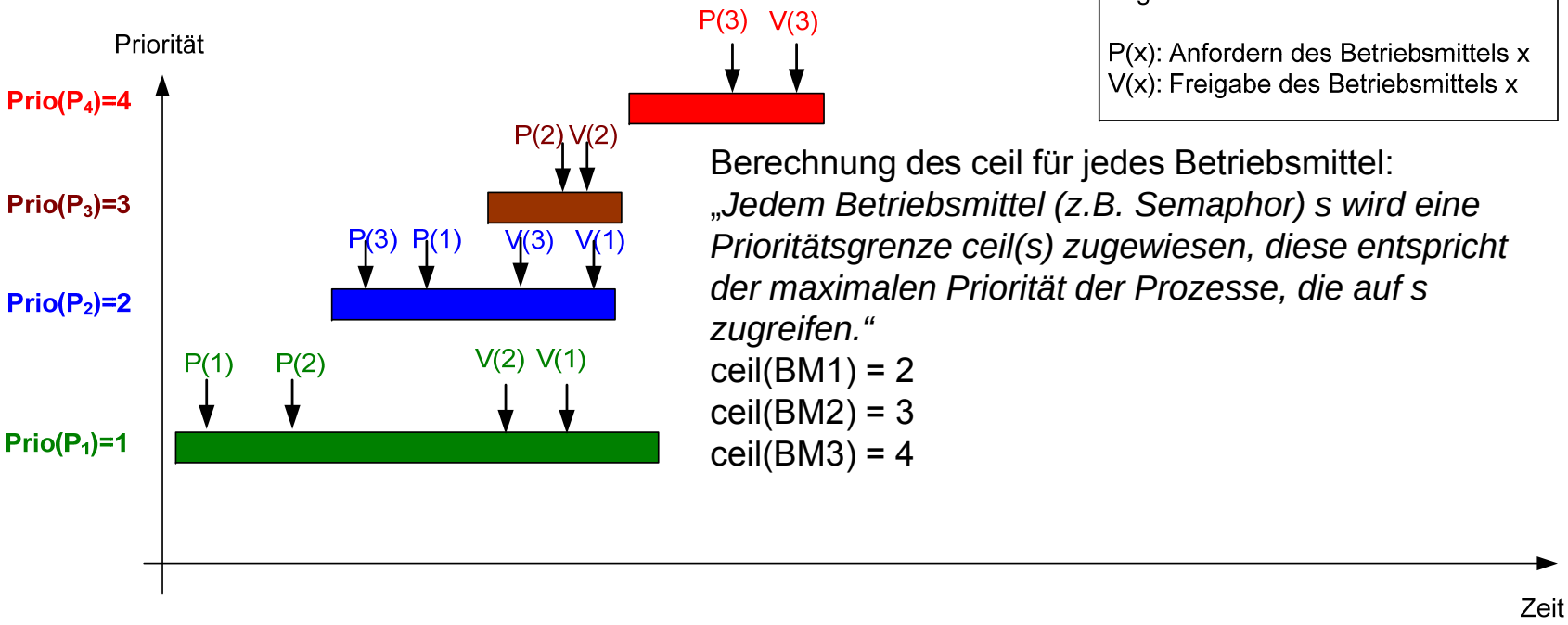
- Jedem Betriebsmittel (z.B. Semaphor) s wird eine Prioritätsgrenze ceil(s) zugewiesen, diese entspricht der maximalen Priorität der Prozesse, die auf s zugreifen.
  - Ein Prozess p darf ein BM nur blockieren, wenn er von keinem anderen Prozess, der andere BM besitzt, verzögert werden kann.
  - Die aktuelle Prioritätsgrenze für Prozess p ist
    aktceil (p)= max{ ceil(s) | s ∈ locked} mit locked = Menge aller von **anderen Prozessen** blockierten BM
  - Prozess p darf Betriebsmittel s benutzen, wenn für seine aktuelle Priorität aktprio gilt: aktprio(p) > aktceil(p)
  - Andernfalls gibt es genau einen Prozess, der blockierte Betriebsmittel mit der höchsten Prioritätsgrenze besitzt. Die Priorität dieses Prozesses wird auf aktprio gesetzt.
    - Beweisidee, dass es nur einen Prozess gibt: klar, da Prozesse nur dann auf Betriebsmittel zugreifen dürfen, wenn aktprio(p) > aktceil(p)
    - Beweisidee, dass die Priorität dieses Prozesses vorher kleiner ist als aktprio: die Priorität von Prozess p muss größer sein als vom Prozess, der das entsprechende Betriebsmittel hat, da sonst p gar nicht laufen würde
  - Gibt ein Prozess ein Betriebsmittel wieder frei, muss aktprio wieder neu berechnet werden

  → 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 ceil(s) zugewiesen.

Beispiel: Prioritätsobergrenzen: Ausgangszustand



Legende:

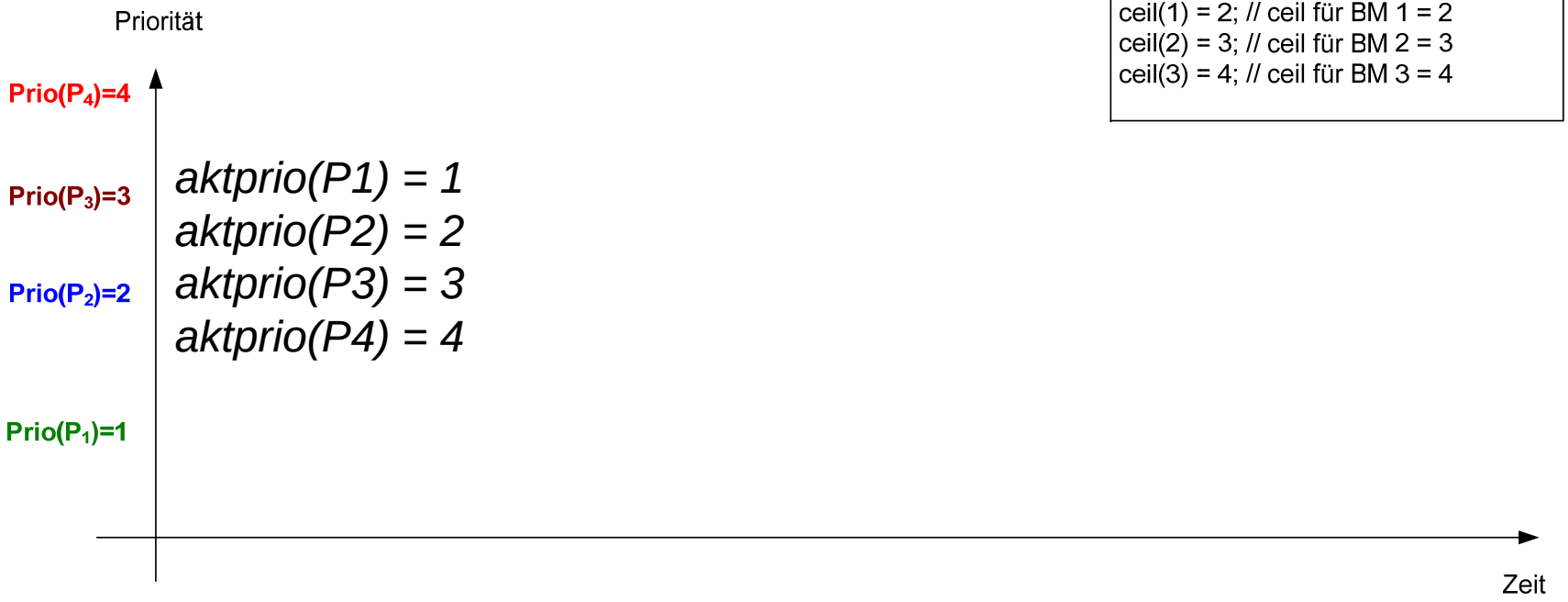
P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x

Berechnung des ceil für jedes Betriebsmittel:
„Jedem Betriebsmittel (z.B. Semaphor) s wird eine Prioritätsgrenze ceil(s) zugewiesen, diese entspricht der maximalen Priorität der Prozesse, die auf s zugreifen.“
ceil(BM1) = 2
ceil(BM2) = 3
ceil(BM3) = 4

Beispiel: Prioritätsobergrenzen

Legende:

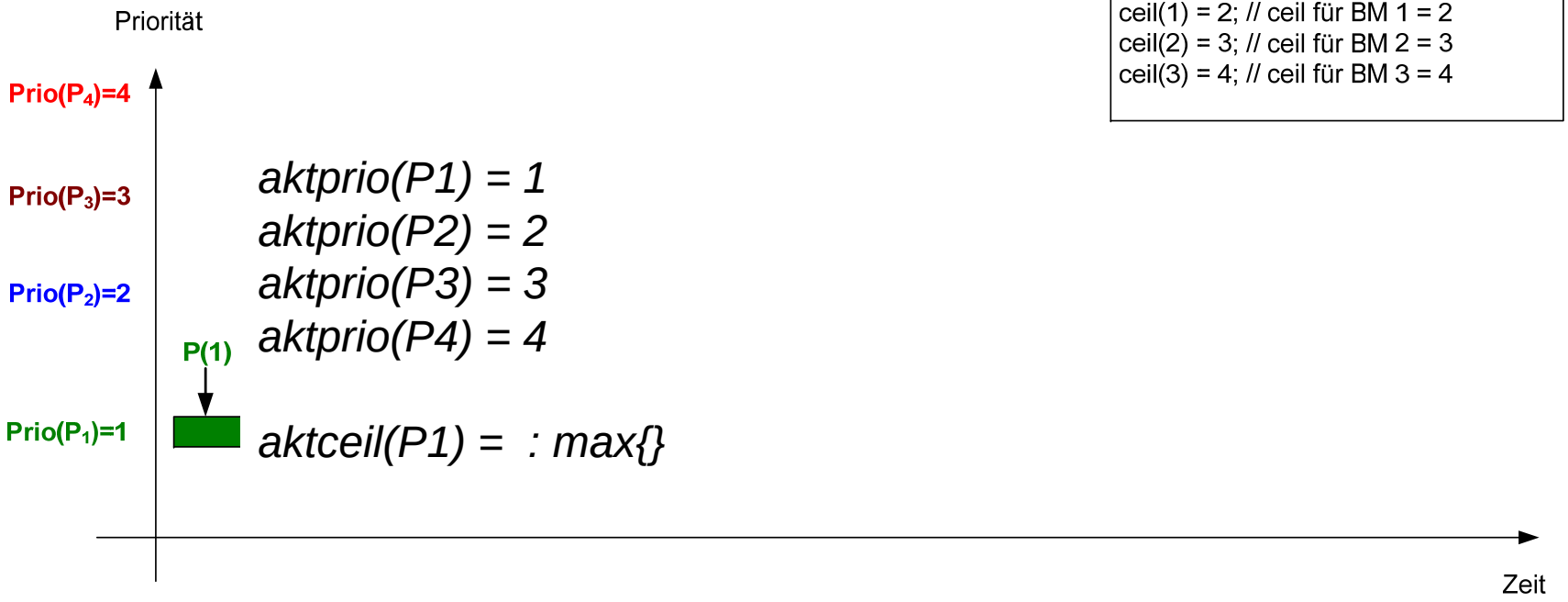
P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4



Beispiel: Prioritätsobergrenzen

Legende:

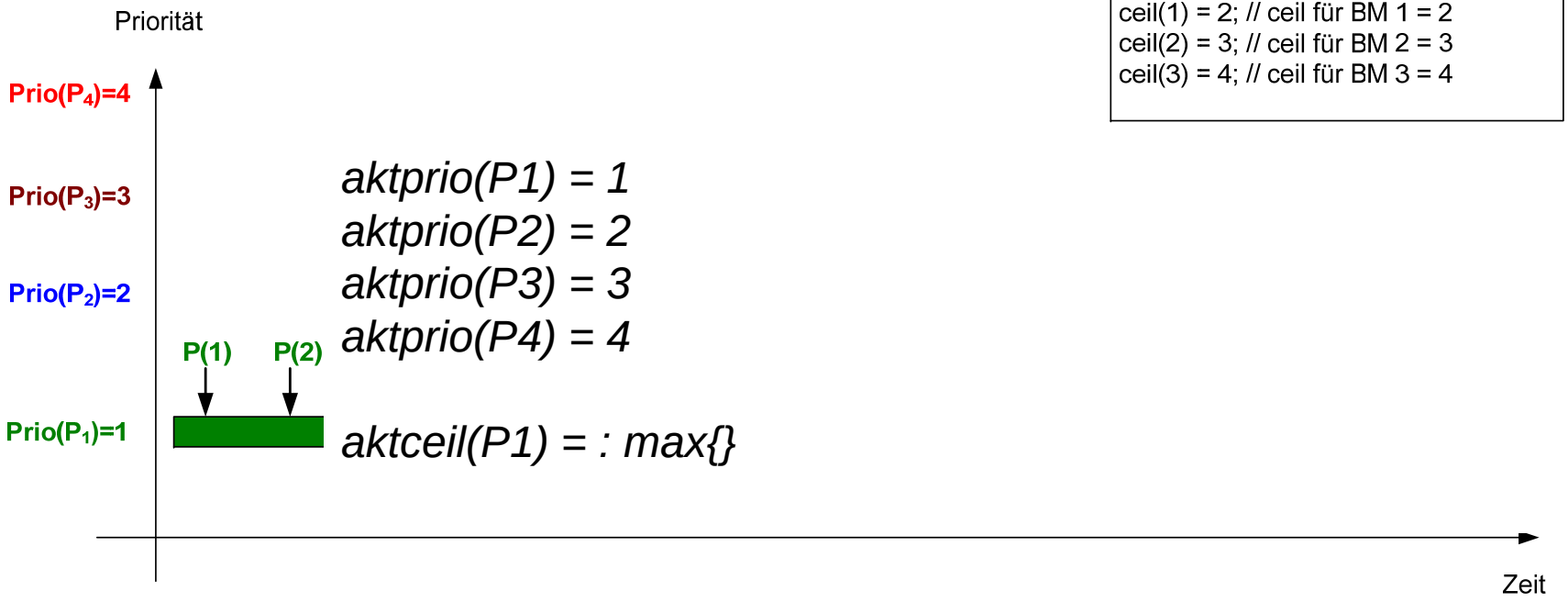
P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4



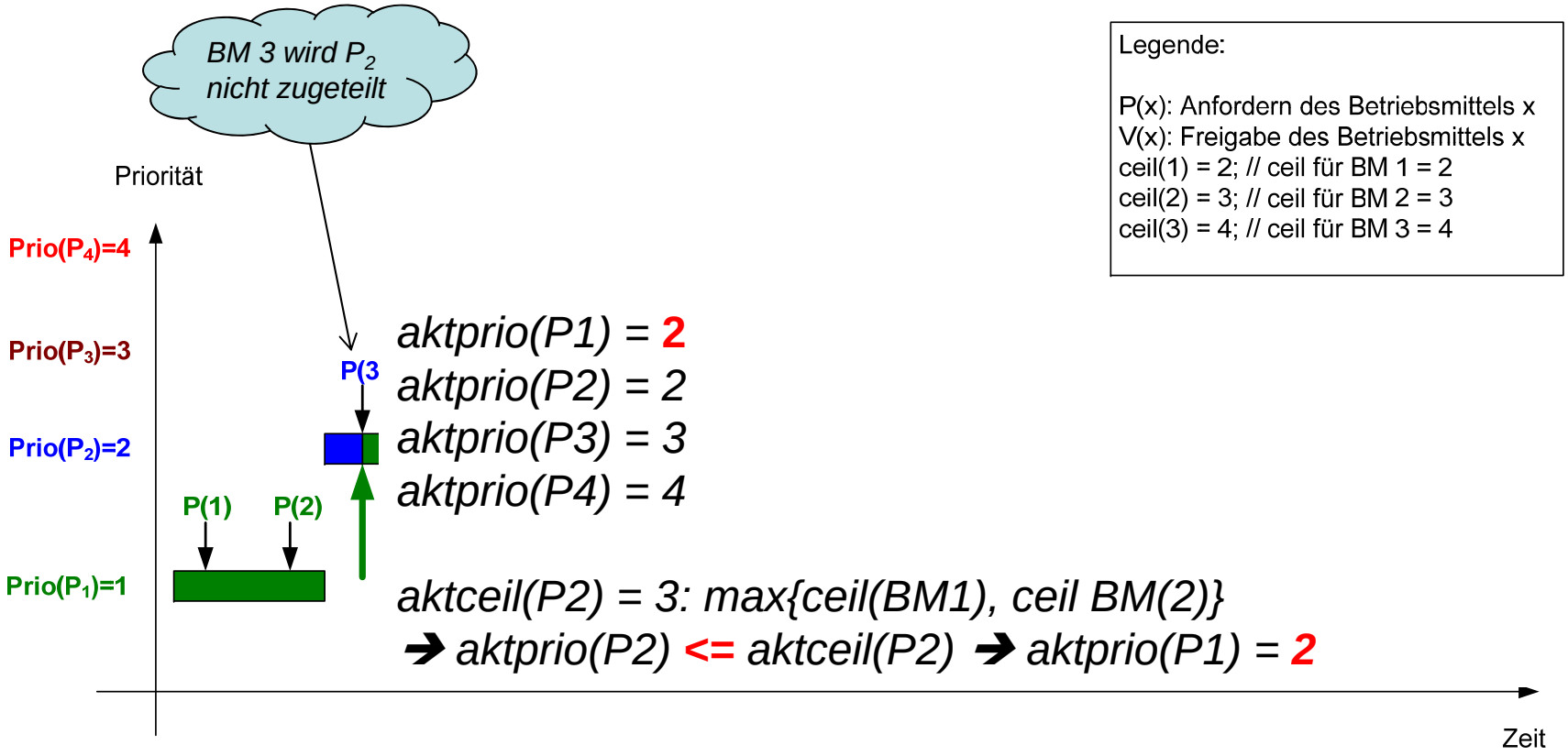
Beispiel: Prioritätsobergrenzen

Legende:

P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4



Beispiel: Prioritätsobergrenzen



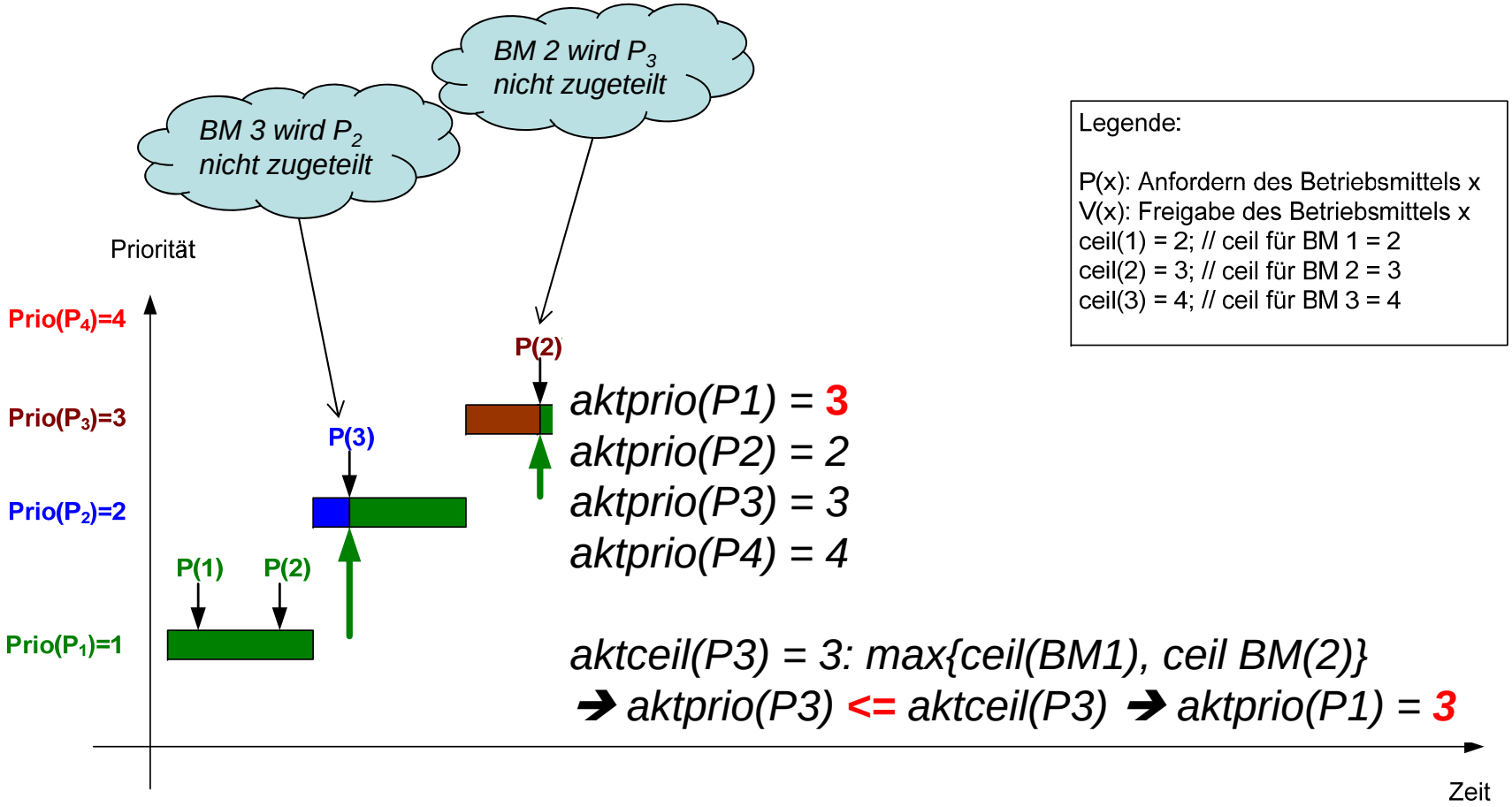
*aktprio(P1) = **2***
aktprio(P2) = 2
aktprio(P3) = 3
aktprio(P4) = 4

aktceil(P2) = 3: max{ceil(BM1), ceil BM(2)}
*➔ aktprio(P2) <= aktceil(P2) ➔ aktprio(P1) = **2***

Legende:

P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4

Beispiel: Prioritätsobergrenzen



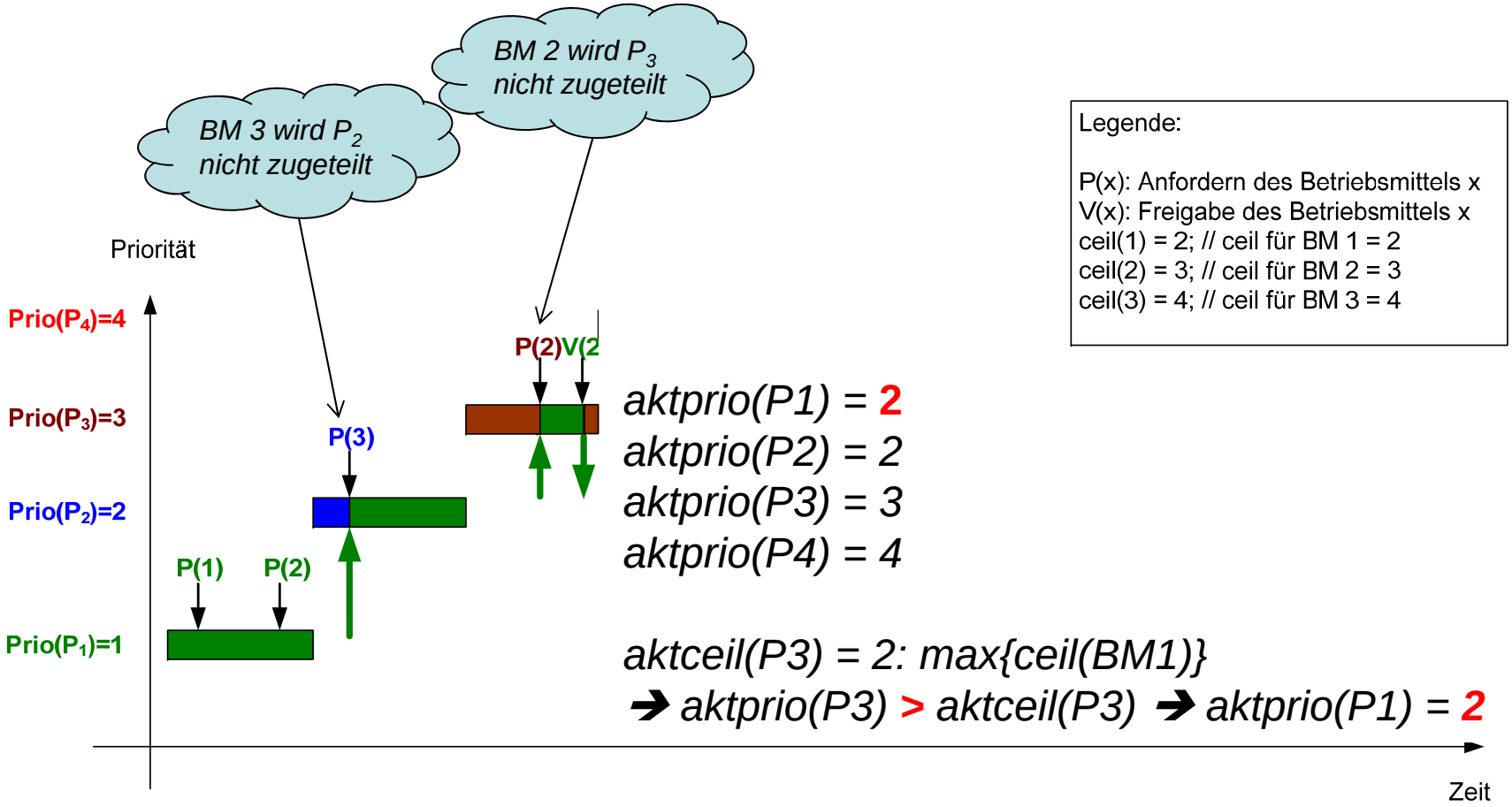
Legende:

P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4

*aktprio(P1) = **3***
aktprio(P2) = 2
aktprio(P3) = 3
aktprio(P4) = 4

aktceil(P3) = 3: max{ceil(BM1), ceil BM(2)}
*➔ aktprio(P3) **<=** aktceil(P3) ➔ aktprio(P1) = **3***

Beispiel: Prioritätsobergrenzen



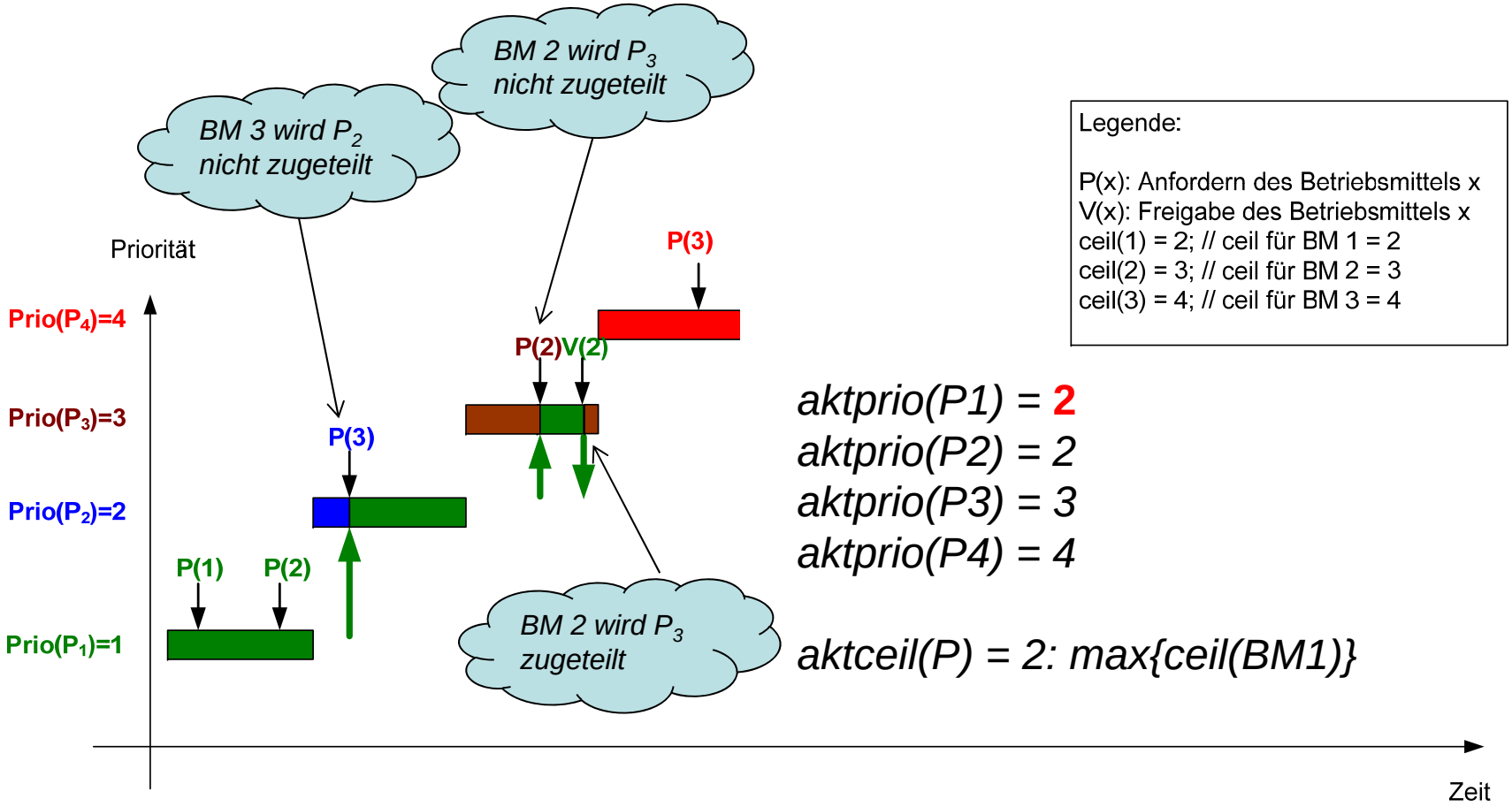
Legende:

P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4

*aktprio(P1) = **2***
aktprio(P2) = 2
aktprio(P3) = 3
aktprio(P4) = 4

aktceil(P3) = 2: max{ceil(BM1)}
*➔ aktprio(P3) > aktceil(P3) ➔ aktprio(P1) = **2***

Beispiel: Prioritätsobergrenzen



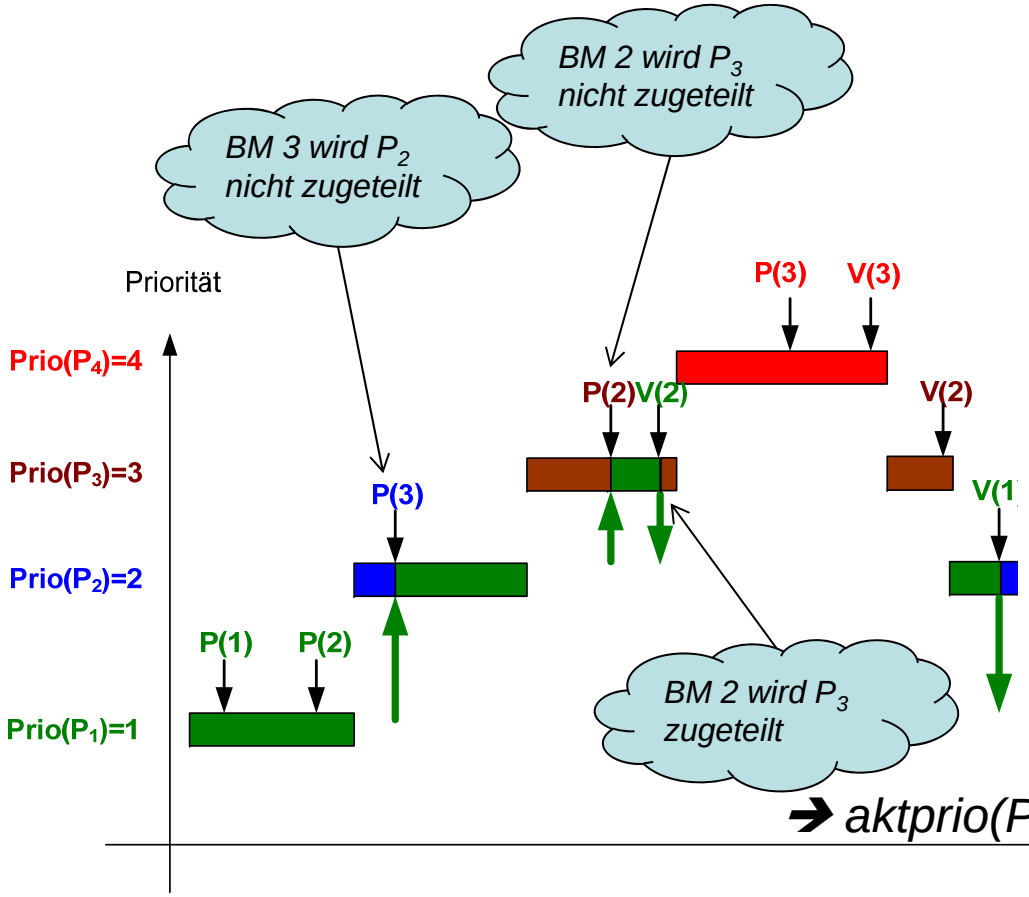
Legende:

P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4

*aktprio(P1) = **2***
aktprio(P2) = 2
aktprio(P3) = 3
aktprio(P4) = 4

aktceil(P) = 2: max{ceil(BM1)}

Beispiel: Prioritätsobergrenzen



Legende:

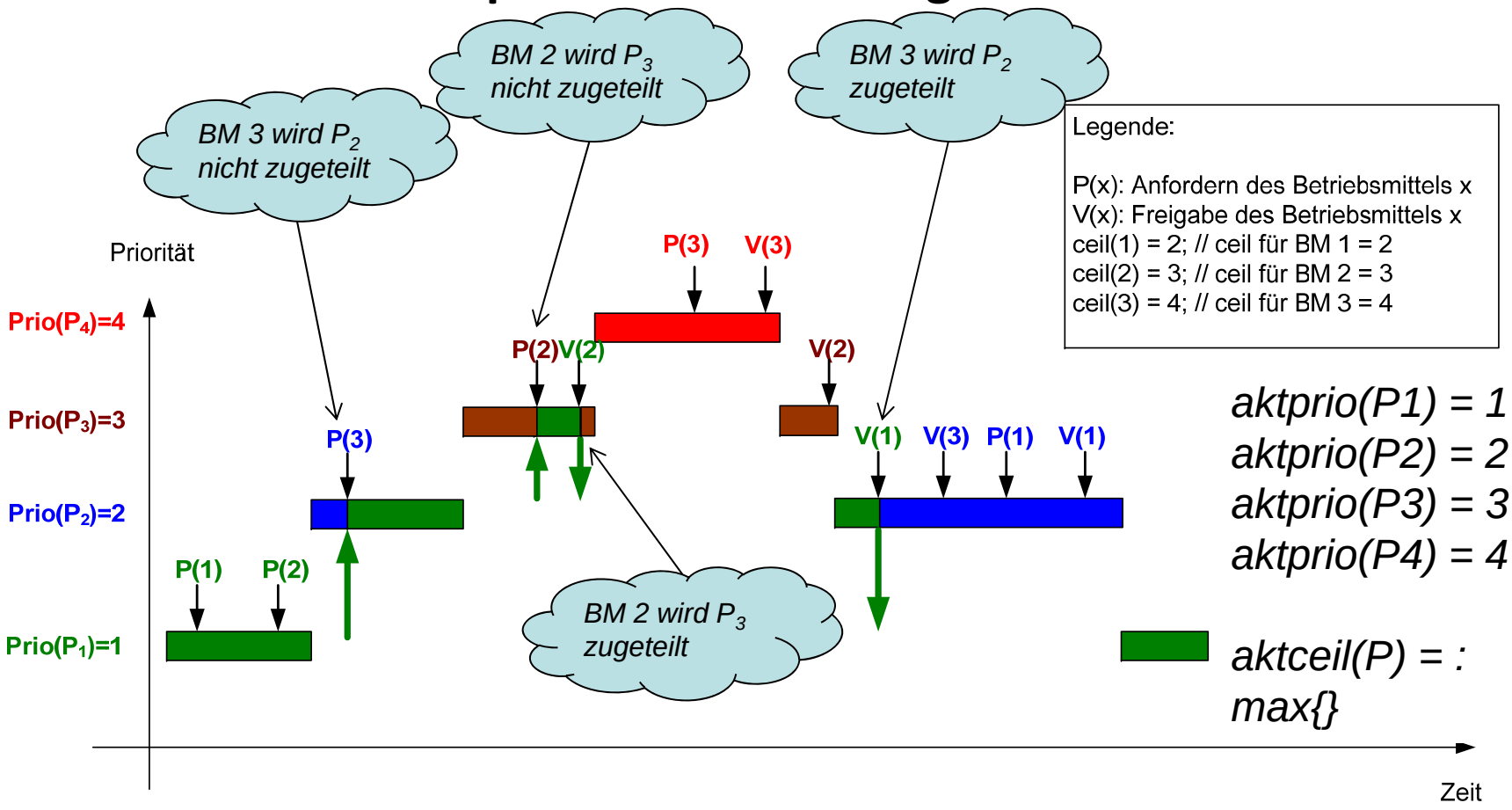
P(x): Anfordern des Betriebsmittels x
V(x): Freigabe des Betriebsmittels x
ceil(1) = 2; // ceil für BM 1 = 2
ceil(2) = 3; // ceil für BM 2 = 3
ceil(3) = 4; // ceil für BM 3 = 4

*aktprio(P1) = **1***
aktprio(P2) = 2
aktprio(P3) = 3
aktprio(P4) = 4

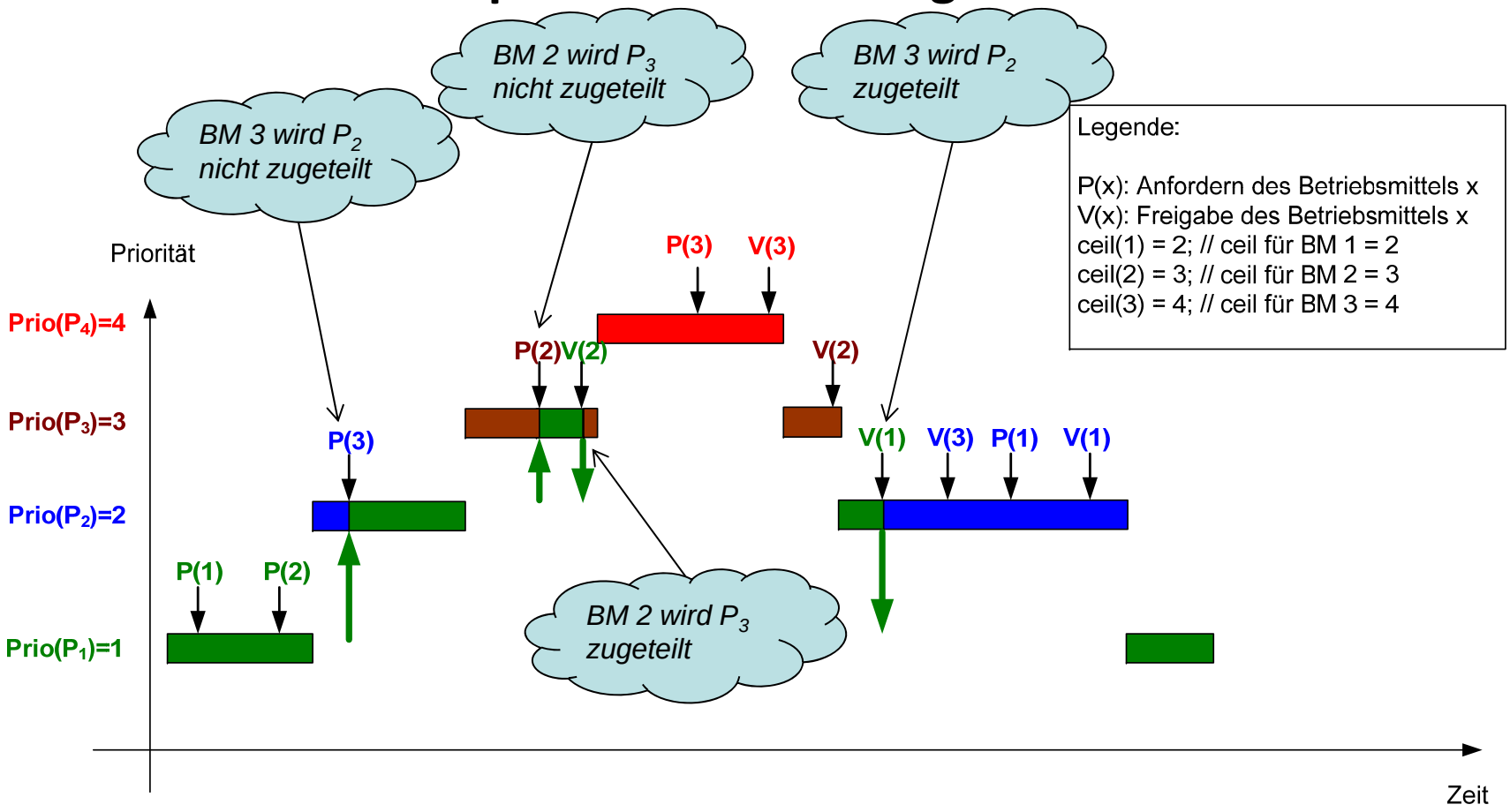
aktceil(P) = : max{}

➔ *aktprio(P2) > aktceil(P2)* ➔ *aktprio(P1) = **1***

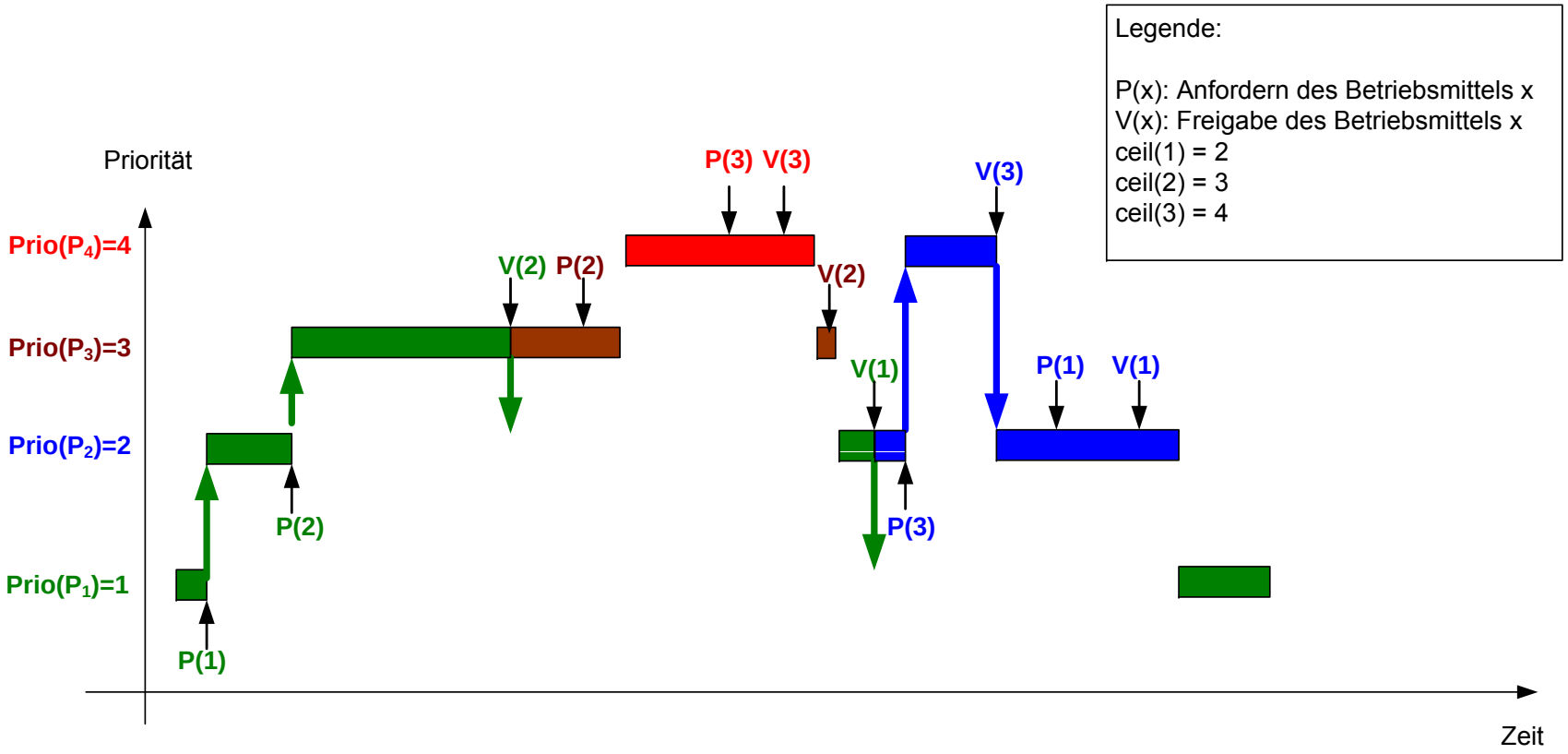
Beispiel: Prioritätsobergrenzen



Beispiel: Prioritätsobergrenzen



Beispiel: Immediate Priority Ceiling

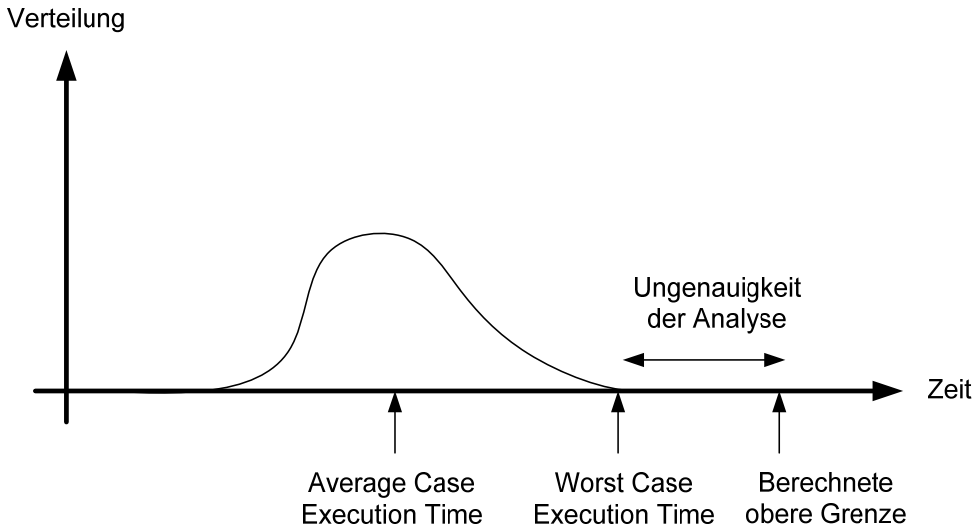


Scheduling

Exkurs: WCET (Worst Case Execution Time) - Analyse

WCET Analyse

- Ziel der Worst Case Execution Time Analyse ist die Abschätzung der maximalen Ausführungszeit einer Funktion



- Die Laufzeit ist abhängig von den Eingabedaten, dem Prozessorzustand, der Hardware,…

Probleme bei der WCET Analyse

- Bei der Abschätzung der maximalen Ausführungszeiten stößt man auf einige Probleme:
  - Es müssen unter anderem die Auswirkungen der Hardwarearchitektur, des Compilers und des Betriebssystems untersucht werden. Dadurch erschwert sich eine Vorhersage.
  - Zudem dienen viele Eigenschaften der Beschleunigung des allgemeinen Verhaltens, jedoch nicht des Verhaltens im schlechtesten Fall, z.B.:
    - Caches, Pipelines, Virtual Memory
    - Interruptbehandlung, Präemption
    - Compileroptimierungen
    - Rekursion

	Zugriffszeit	Größe
Register	0.25 ns	500 bytes
Cache	1 ns	64 KB
Hauptspeicher	100 ns	512 MB
Festplatte	5 ms	100 GB

Zugriffszeiten für verschiedene Speicherarten

  - Noch schwieriger wird die Abschätzung falls der betrachtete Prozess von der Umgebung abhängig ist.

Unterscheidungen bei der WCET-Analyse

- Die Analyse muss auf unterschiedlichen Ebenen erfolgen:
  - Was macht das Programm?
  - Was passiert im Prozessor?

- Bei der Analyse werden zwei Methoden unterschieden:
  - **statische** WCET Analyse: Untersuchung des Programmcodes
  - **dynamische** Analyse: Bestimmung der Ausführungszeit anhand von verschiedenen repräsentativen Durchläufen

Statische Analyse

- Aufgaben:
  - Bestimmung von Ausführungspfaden in der Hochsprache
  - Transformation der Pfade in Maschinencode
  - Bestimmung der Laufzeit einzelner Maschinencodesequenzen
- Probleme:
  - Ausführungspfade lassen sich oft schlecht vollautomatisch ableiten (zu pessimistisch, zu komplex)
  - Ausführungspfade häufig abhängig von Eingabedaten
- Lösungsansatz: Annotierung der Pfade mit Beschränkungen (wie z.B. maximale Schleifendurchläufe)

Dynamische Analyse

- Statische Analysen können zumeist die folgenden Wechselwirkungen nicht berücksichtigen:
  - Wechselwirkungen mit anderen Programmen (siehe z.B. wechselseitiger Ausschluss)
  - Wechselwirkungen mit dem Betriebssystem (siehe z.B. Caches)
  - Wechselwirkungen mit der Umgebung (siehe z.B. Interrupts)
  - Wechselwirkungen mit anderen Rechnern (siehe z.B. Synchronisation)
- Durch dynamische Analysen können diese Wechselwirkungen abgeschätzt werden.
- Problem: Wie können die Testläufe sinnvoll ausgewählt werden.

Dimensionierung der Rechenleistungen

- Aufstellen der Worst-Case Analyse:
  - Rechenaufwand für bekannte periodische Anforderungen
  - Rechenaufwand für erwartete sporadische Anforderungen
  - Zuschlag von 100% oder mehr zum Abfangen von Lastspitzen
- Unterschied zu konventionellen Systemen:
  - keine maximale Auslastung des Prozessors
  - keine Durchsatzoptimierung
  - Abläufe sollen determiniert abschätzbar sein

Scheduling

Zusammenfassung

Zusammenfassung

- Kenntniss der Schedulingkriterien (Einhalten von Fristen, Fairness,…) , sowie der verschiedenen Prozessparameter (Startzeit, Laufzeit, Deadline, Priorität).
- Klassische Verfahren (EDF, LST, RM) und Anforderungen an die Optimalität dieser Verfahren
- Problem der Prioritätsinversion, sowie Lösungsverfahren
- Problematik der WCET-Analyse

Kapitel 6

Echtzeitfähige Kommunikation

Inhalt

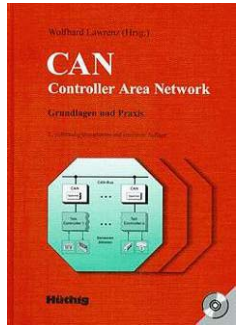
- Grundlagen
- Medienzugriffsverfahren und Vertreter
  - CSMA-CD: Ethernet
  - CSMA-CA: CAN-Bus
  - Tokenbasierte Protokolle: Token Ring, FDDI
  - Zeitgesteuerte Protokolle: TTP
  - Real-Time Ethernet

Literatur

- Spezifikationen:
  - TTTech Computertechnik AG, Time Triggered Protocol TTP/C High-Level Specification Document, 2003 (http://www.vmars.tuwien.ac.at/projects/ttp/)
  - http://www.can-cia.org/
  - http://standards.ieee.org/getieee802/portfolio.html

Andrew S. Tanenbaum, Computernetzwerke, 2005

Wolfhard Lawrenz: CAN Controller Area Network. Grundlagen und Praxis, 2000

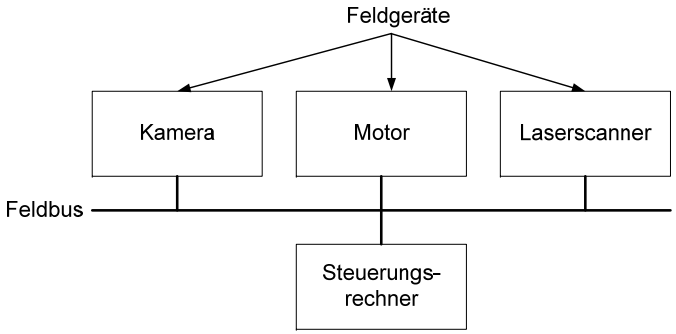


Anforderungen

- Echtzeitsysteme unterscheiden sich in ihren Anforderungen an die Kommunikation von Standardsystemen.
- Anforderungen speziell von Echtzeitsystemen:
  - vorhersagbare maximale Übertragungszeiten
  - kleiner Nachrichtenjitter
  - garantierte Bandbreiten
  - effiziente Protokolle: kurze Latenzzeiten
  - teilweise Fehlertoleranz
- Kriterien bei der Auswahl:
  - maximale Übertragungsrate
  - maximale Netzwerkgröße (Knotenanzahl, Länge)
  - Materialeigenschaften (z.B. für Installation)
  - Störungsempfindlichkeit (auch unter extremen Bedingungen)
  - Kosten, Marktproduktpalette

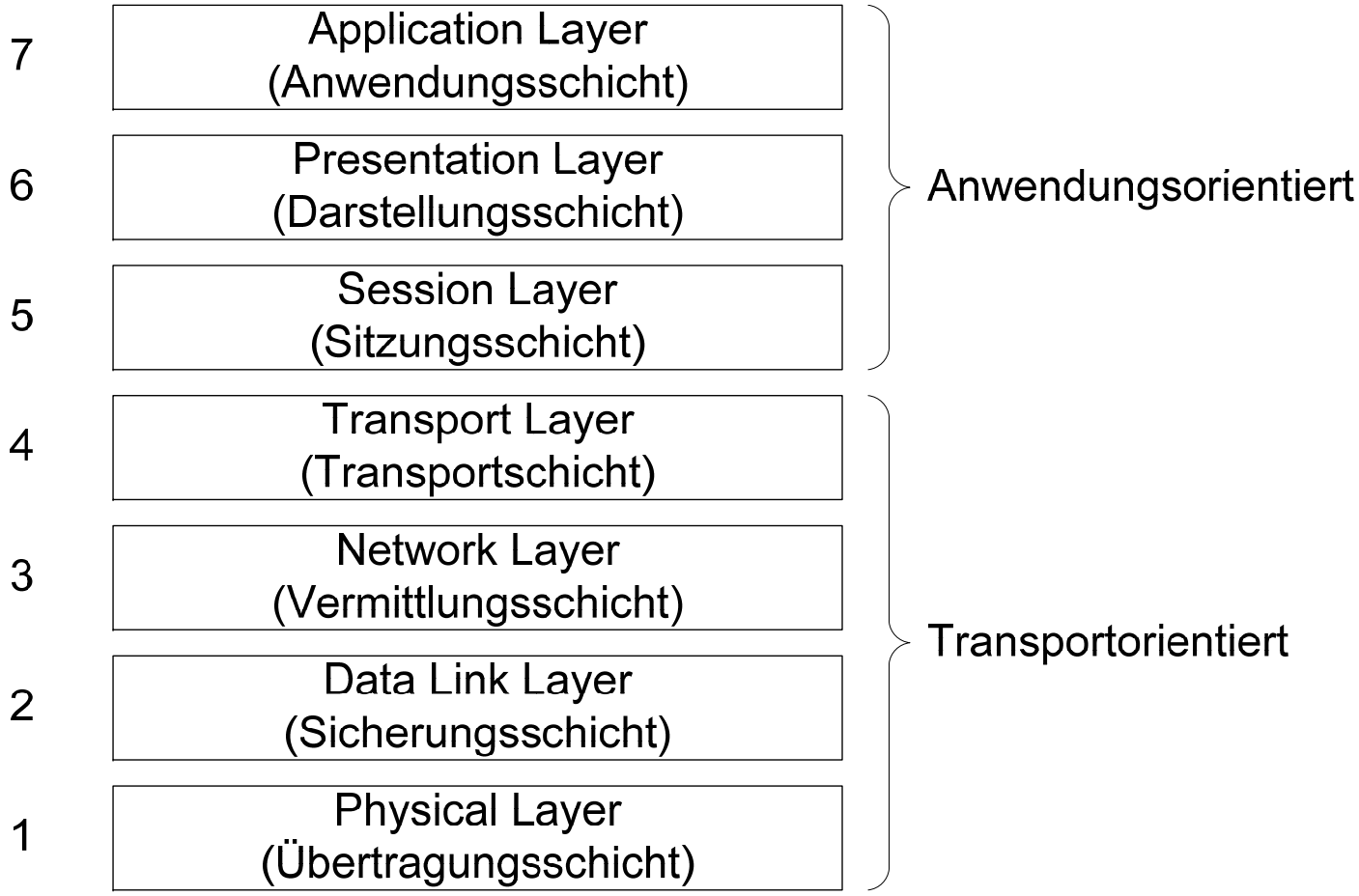
Definition Feldbus

- Die Kommunikation in Echtzeitsystemen erfolgt häufig über **Feldbusse**:



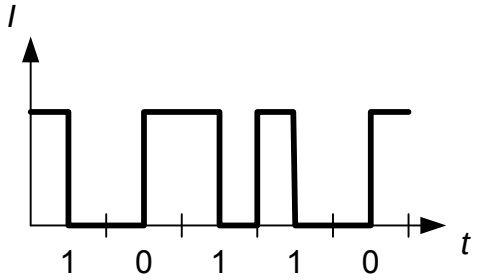
- Feldgeräte sind dabei Sensoren/Aktoren, sowie Geräte zur Vorverarbeitung der Daten.
- Der Feldbus verbindet die Feldgeräte mit dem Steuerungsgerät.
- Beobachtung: echtzeitkritische Nachrichten sind in der Regel kürzer als unkritische Nachrichten.
- Es existiert eine Vielzahl von Feldbus-Entwicklungen: MAP (USA - General Motors), FIP (Frankreich), PROFIBUS (Deutschland), CAN (Deutschland – Bosch), …

Schichtenmodell: ISO/OSI-Modell

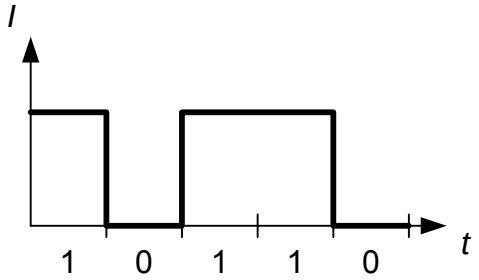


Beschreibung der einzelnen Schichten: Übertragungsschicht

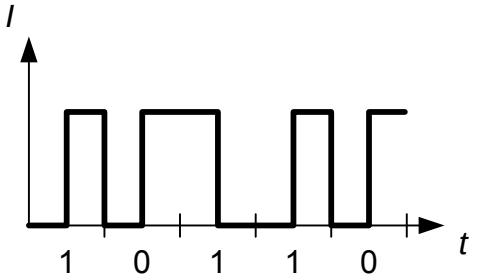
- Aufgaben:
  - Bitübertragung auf physikalischen Medium
  - Festlegung der Medien
    - elektrische, optische Signale, Funk
    - Normung von Steckern
  - Festlegung der Übertragungsverfahren/Codierung
    - Interpretation der Pegel
    - Festlegung der Datenrate



Manchester-Code



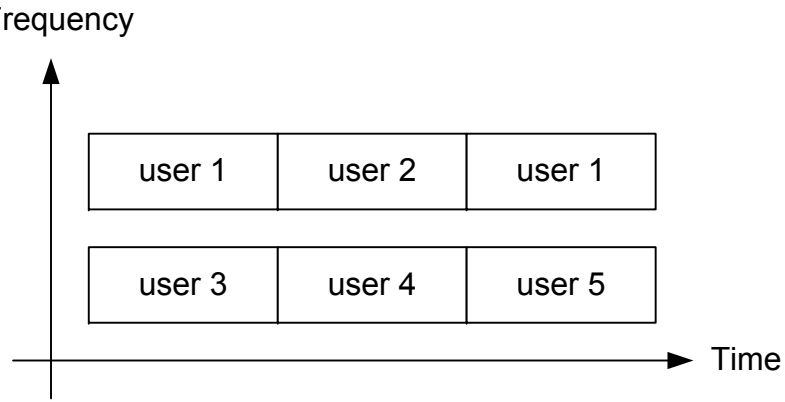
Non-return-to-zero Code



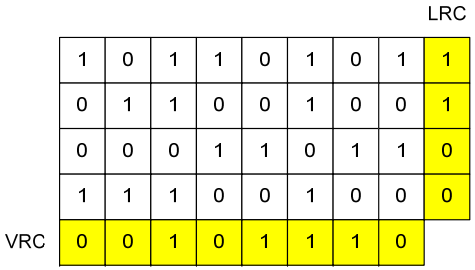
Differentieller Manchester-Code

Beschreibung der einzelnen Schichten: Sicherungsschicht

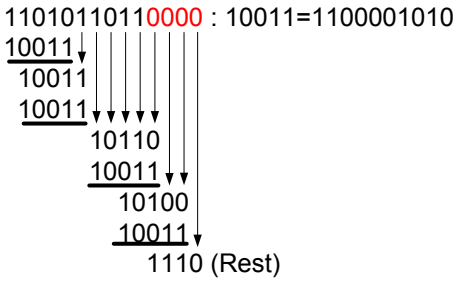
- Aufgaben:
  - Fehlererkennung
    - Prüfsummen
    - Paritätsbits
  - Aufteilung der Nachricht in Datenpakete
  - Regelung des Medienzugriffs
  - Flusskontrolle



TDMA+FDMA



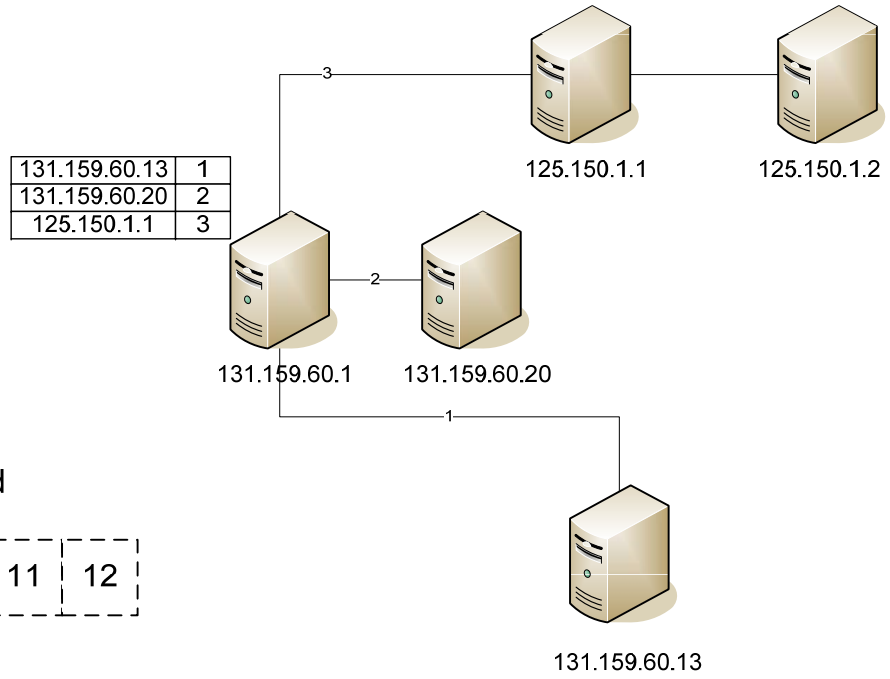
Paritätsbits

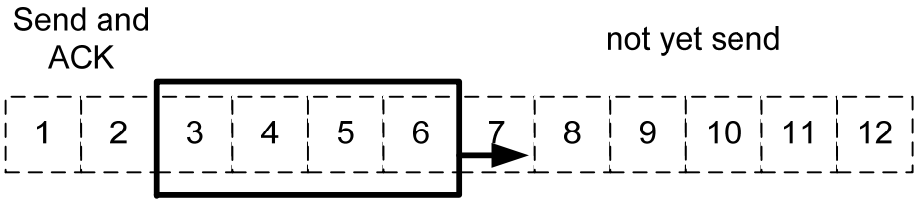


CRC-Verfahren

Beschreibung der einzelnen Schichten: Vermittlungsschicht

- Aufgaben:
  - Aufbau von Verbindungen
  - Weiterleitung von Datenpaketen
    - Routingtabellen
    - Flusskontrolle
    - Netzwerkadressen





Sliding Window Protokoll

Weitere Schichten

- Transportschicht:
  - Transport zwischen Sender und Empfänger (End-zu-End-Kontrolle)
  - Segmentierung von Datenpaketen
  - Staukontrolle (congestion control)
- Sitzungsschicht:
  - Auf- und Abbau von Verbindungen auf Anwendungsebene
  - Einrichten von Check points zum Schutz gegen Verbindungsverlust
  - Dienste zur Organisation und Synchronisation des Datenaustauschs
  - Spezifikation von Mechanismen zum Erreichen von Sicherheit (z.B. Passwörter)
- Darstellungsschicht:
  - Konvertierung der systemabhängigen Daten in unabhängige Form
  - Datenkompression
  - Verschlüsselung
- Anwendungsschicht:
  - Bereitstellung anwendungsspezifischer Übertragungs- und Kommunikationsdienste
  - Beispiele:
    - Datenübertragung
    - E-Mail
    - Virtual Terminal
    - Remote Login
    - Video-On-Demand
    - Voice-over-IP