Vorlesung Real-Time Systems
1. Einführung

Übersicht

1. Motivation
2. Was ist ein Realzeit-System?
3. Organisation der Veranstaltung
4. Historische Perspektive
5. Anwendungen
6. Was ist das Problem?
7. Literatur

1. Motivation

Der Einsatz von Computer-Systemen für die Steuerung von alltäglichen Prozessen ist inzwischen eine Selbstverständlichkeit geworden. Beispiele dafür sind zu finden in:

- Flugzeugen
- medizinischen Geräten (z.B. Respiratoren)
- Automobilen z.B. Motor+ Bremsen

Meistens sind die Anforderungen für das eingesetzte Computer-System sehr hoch, denn es muss mindestens so sicher und **zuverlässig** sein wie die konventionelle Technologie, die es normalerweise ersetzt. Zusätzlich ermöglichen dabei Computer **neue Funktionalitäten**, die wiederum das System komplexer machen.
Die Gründe für das Ersetzen rein mechanischer oder elektrischer Systeme sind wie folgt:

- **Reduzierung von Produktionskosten**: z.B. lassen sich die Kabelkosten beim Bau eines Autos und durch den Einsatz eines Kommunikations-Netzwerks, an das viele Komponenten angeschlossen werden, **erheblich** senken.

1. Motivation

- Der **Funktionsumfang** lässt sich einfach steigern: Durch eine Drive-by-Wire-Lenkung (d.h. es gibt keine mechanische Verbindung von Lenkrad und Rädern) lassen sich bessere Fahreigenschaften erzielen

Natürlicherweise tauchen hierbei auch **Probleme** auf. Das größte ist: Ein **Software-Fehler** kann zum Stillstand/Ausfall des Systems führen.

Überhaupt fallen die meisten gewohnten physischen und mechanischen Größen mit Ausnahme der **Zeit** weg. Wird also ein Computer-System eingesetzt zur Regelung/Steuerung von elektromechanischen Komponenten, ist es sehr wichtig, die korrekte Arbeitsweise zu gewährleisten und zwar

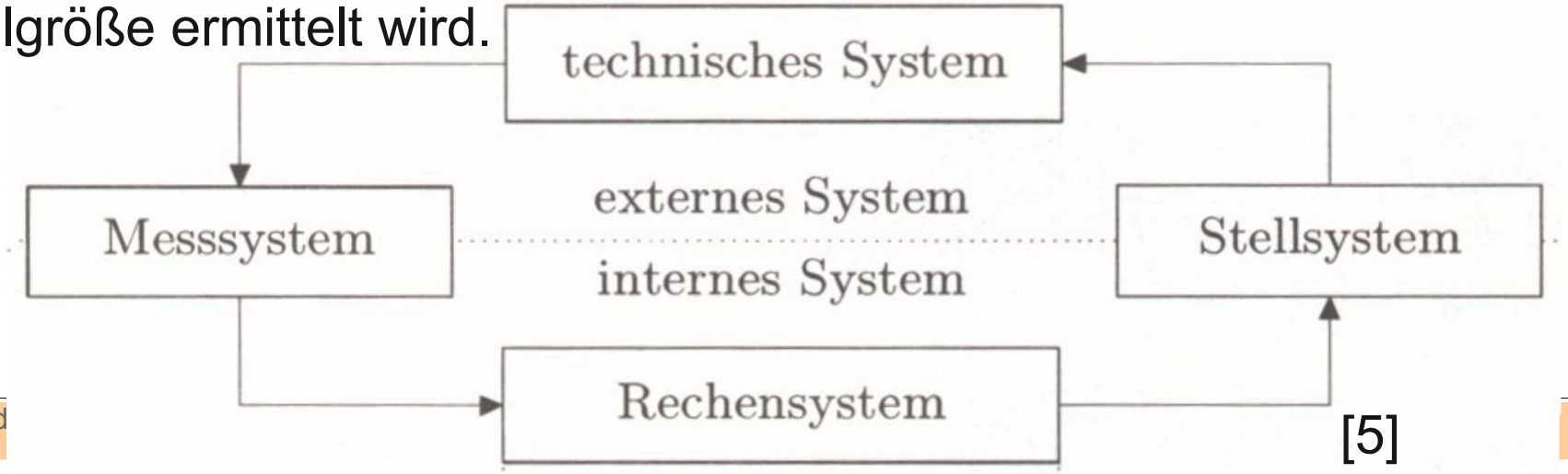
- sowohl in der Funktionalität
- als auch in der zeitlichen Dimension.

Daher wird ein solches System **Echtzeitsystem** (Realzeit-System, Real-Time System) genannt.

2. Was ist ein Realzeit-System?

Ein Realzeit-System **(RTS)** steht typischerweise in Wechselwirkung mit einem physikalischen Prozess:

- Dabei ermittelt einerseits ein **Sensor** (Messsystem) physikalische Daten (z.B. Spannung, Temperatur, Druck,.....), die in digitale Daten zur Verarbeitung durch den Computer verwandelt werden.
- Andererseits bildet ein **Aktuator** (Stellsystem) das Gegenstück dazu, indem angelieferte Stellgröße an die Elektro-Mechanik weitergegeben wird.
- Um z.B. die Drehzahl eines Motors zu regeln, misst man also den aktuellen Wert durch den Sensor und stellt über den Aktuator den gewünschten Wert ein. Das beteiligte Computer-System führt einen **Reglungsalgorithmus** aus, in dem aus dem gelesenen Wert eine neue Stellgröße ermittelt wird.



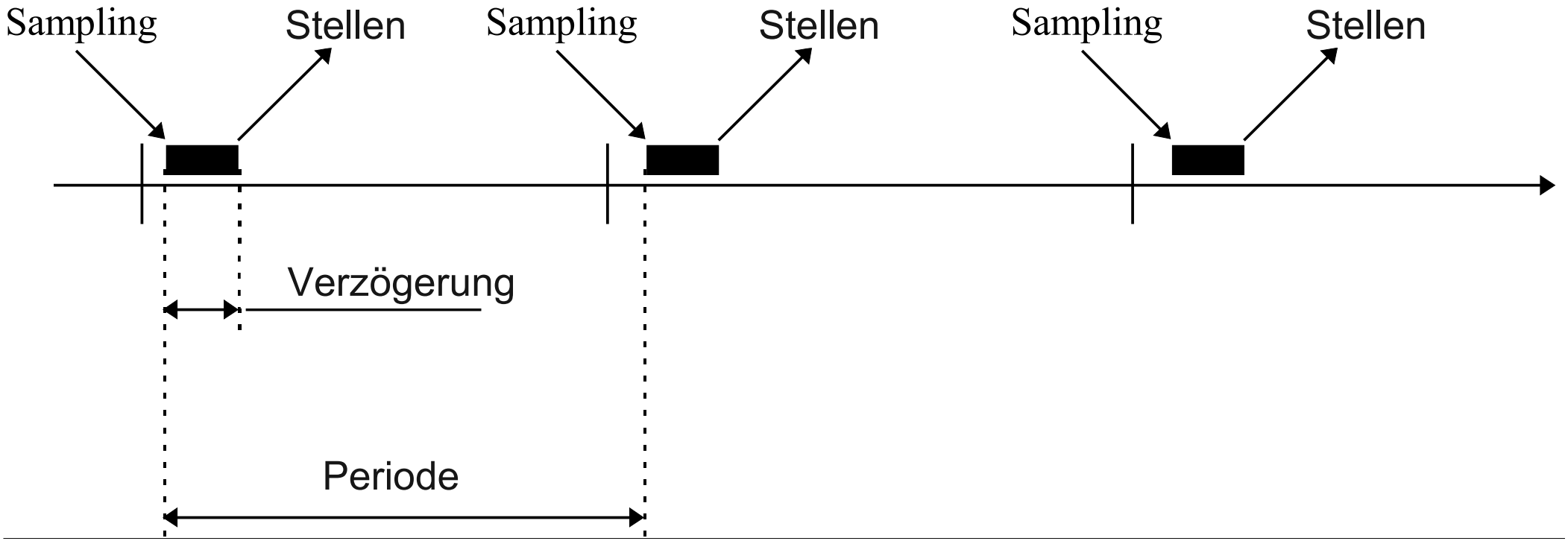
[5]

2. Was ist ein Realzeit-System?

Aus der Sicht der Computer ergibt sich folgendes **periodisches Szenario**:

1. Die **Messdaten** werden ermittelt (**Sampling**)
2. Der **Reglungsalgorithmus** wird angewendet (→ zeitliche Verzögerung = Ausführungszeit auf dem Prozessor)
3. Die **Stellgröße** wird an den Aktuator geliefert

Grafische Darstellung [2]:



2. Was ist ein Realzeit-System?

Daraus können folgende **zeitliche Anforderungen** abgeleitet werden:

1. **Wie oft** müssen Werte gelesen bzw. gestellt werden (d.h. welche Taktrate, Periodenlänge), damit der gewünschte Regelungseffekt erreicht wird?
2. **Wie schnell** müssen die Daten durch das Computer-System verarbeitet werden (Ausführungszeit)?

Beide Fragen werden normalerweise durch die Analyse bzw. das Wissen eines Regelungstechnikers beantwortet. Die Beantwortung der beiden Fragen kommt dem Stellen von **Anforderungen an das Computer-System** gleich:

1. Angabe der **Periodenlänge** (bzw. Taktrate)
2. Angabe einer **Deadline** (Zeitschranke)

2. Was ist ein Realzeit-System?

Es wird ausdrücklich darauf hingewiesen, dass ein RTS kein besonders schnelles System sein muss. Statt dessen ist es wichtig, dass sich alle beteiligten Komponenten vorhersagbar verhalten. Nur so ist ein bestimmtes Zeitverhalten zu garantieren.

Definition lt. [Kopetz]:

"A real-time system is a computer system in which the correctness of the system behaviour depends not only on the logical results of the computation, but also on the physical instant at which these results are produced."

Anders als in „normalen" Systemen, kommt es in einem RTS darauf an, dass Berechnungen bis zu einem bestimmten **Zeitpunkt (Deadline)** zu erfolgen haben.
Ist dies nicht der Fall, kann dies u. U. fatale Folgen haben. Solche Systeme werden harte RTS genannt und sind vorwiegend im Bereich von eingebetteten Systemen zur Regelung und Steuerung zu finden.

2. Was ist ein Realzeit-System?

Ein eingebettetes System wird **definiert** als:

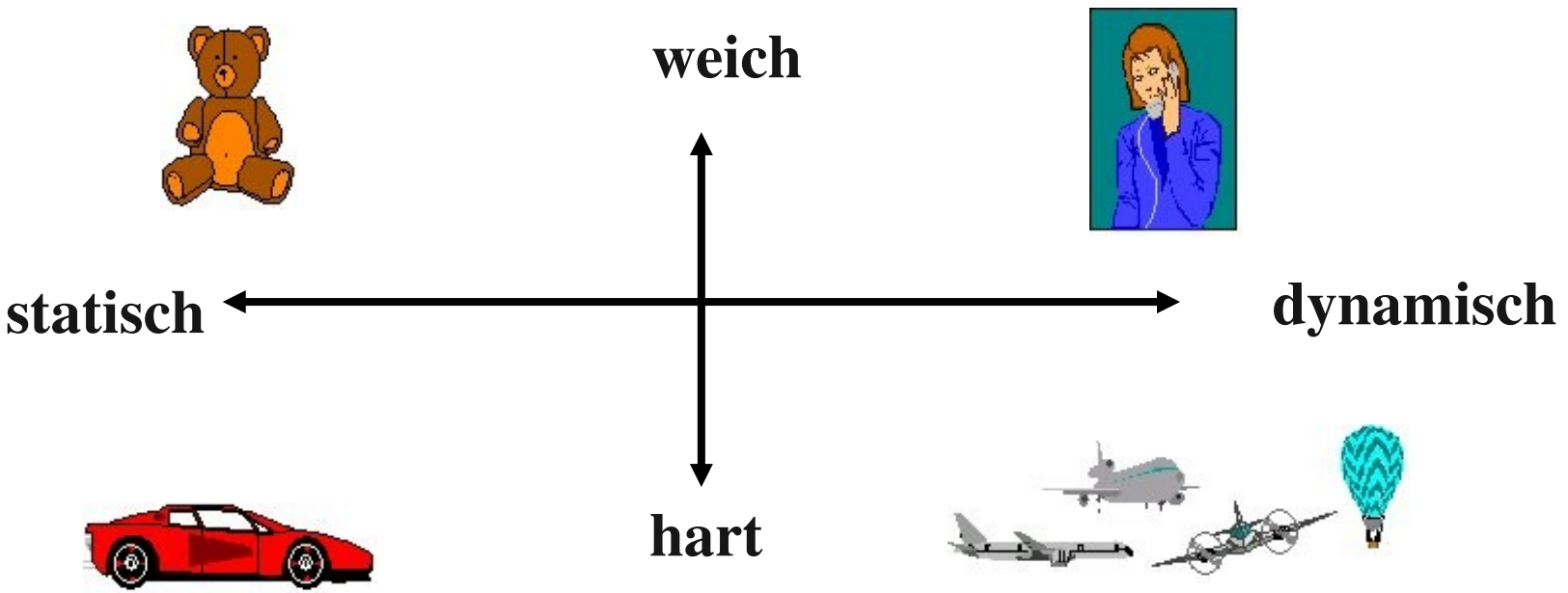
“Unter einem eingebetteten System versteht man ein Computer-System, das fester Bestandteil eines Gerätes oder einer Anlage ist und für das Gesamtsystem bestimmte funktionale und leistungsmäßige Anforderungen erfüllt.“ [5]

Ein Beispiel ist ein ABS (Antiblockier-System) im Auto. Aus nahe liegenden Gründen ist es dabei nicht nur wichtig, dass der korrekte Brems-Druck errechnet wird, sondern das dies auch rechtzeitig geschieht.

2. Was ist ein Realzeit-System?

Bei weichen (soften) RTS ist das Verpassen einer Deadline hingegen nicht so dramatisch. Beispiele hierfür sind im Bereich von Multimedia-Systemen zu finden. Allerdings ist z.B. ein Hersteller von Fernsehern dennoch auf eine exzellente Bildqualität angewiesen.

Eine **Klassifizierung** von RTS ist der folgenden Abbildung zu entnehmen:



2. Was ist ein Realzeit-System?

Die X-Achse in der Abbildung zeigt an wie stark **zeit- bzw. ereignisgesteuert** das System ist:

- Ein strikt **zeitgesteuertes** d.h. statisches System macht nichts anderes, als einen simplen Fahrplan **(Schedule)** abzuarbeiten.
- In einem **ereignisgesteuerten** System hingegen hängt die Ausführung von äußeren Ereignissen zur Laufzeit ab.

Ebenso gibt es auf der Y-Achse einen Zusammenhang von harter Echtzeit und ausreichend vorhandenen Ressourcen, denn nur so lässt sich diese garantieren:

- Das o. g. ABS-System ist also ein statisches HRT-Problem.
- Ein Telefon-System dagegen ist ereignisorientiert, denn die Kunden entscheiden willkürlich über ein Gespräch. Es ist außerdem begrenzt in seinen Ressourcen und ein Besetztton erklingt, wenn die (Netz-)Kapazität verbraucht ist.

3. Organisation der Veranstaltung

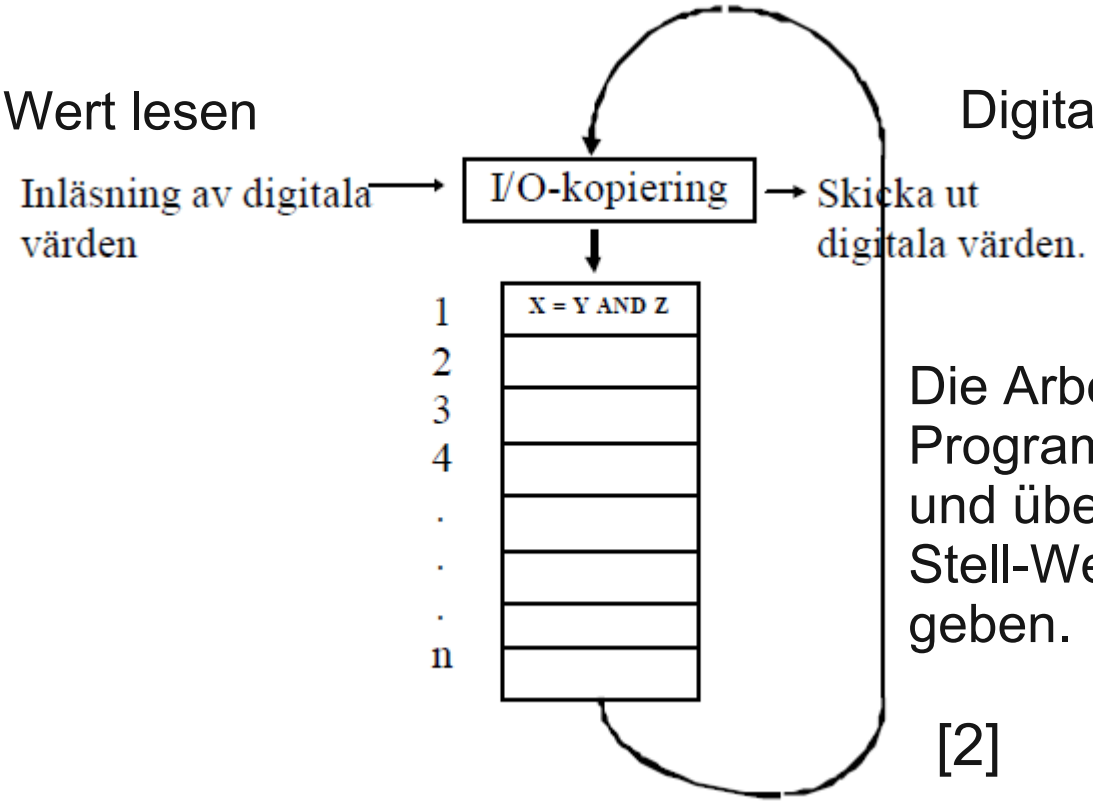
Diese Vorlesung beschäftigt sich mit einer Vielzahl von verschiedenen Aspekten der Real-Time-Systeme:

- Einführung
- Real-Time Scheduling
- Erweitertes Real-Time Scheduling
- Real-Time Scheduling in verteilten Systemen — *Peter Altenbernd*
- Echtzeit-Betriebssysteme
- RT-Kommunikation
- Verteilte RTS
- Entwurf von RTS — *Ralf Mayer*

4. Historische Perspektive

- Bereits im alten China wurde der Wasserstand auf Reisfeldern (ausschließlich durch den Menschen selbst) reguliert.
- Bei der Erfindung der Dampfmaschine wurden die Öffnungszeitpunkte von Ventilen durch eine entsprechende Mechanik implementiert.
- PLC (Programable Logic Controller) sind extrem simple Computer-Architekturen, die besonders wegen ihrer Einfachheit sehr beliebt sind. [2]

Digitalen Wert lesen



Digitalen Wert versenden

Die Arbeitsweise besteht daraus, das Programm in einer Schleife auszuführen und über die IO-Schnittstelle Lese- und Stell-Wert zu ermitteln bzw. zurück zu geben.

[2]

4. Historische Perspektive

- **Micro-Prozessor-gesteuerte Systeme** bilden das oben genannte PLC-Verhalten in einer komplexeren bzw. komfortableren Umgebung ab, indem die Software i.W. das folgende Aussehen hat:

```
void main (void){
  int state1, state2,…, staten;

  initializeClock();

  while(1){
    readSensors(…);

    if (CalculateNewSetValues(…))
        ActuateProcess(…);
    else
        Error();
    WaitForNextCycle(…);
  }
}
```

Zu Beginn jedes Durchlaufs wird der Sensor-Wert gelesen und danach ggf. neue Werte berechnet und zurück gegeben. Am Ende wird auf ein Clock-Signal gewartet, das die nächste Runde einleitet.

4. Historische Perspektive

Außerdem ist es möglich (durch Interrupt-Routinen) gleichzeitig nicht-periodische Funktionen zu implementieren, **z.B.** Autos zu zählen, die eine bestimmte Straße benutzen.

Solche Systeme sind sehr klein (Speicher, Funktionen) und werden **z.B.** in Autos für Fensterheber benutzt.

- Heute gibt es je nach Anwendungen sehr unterschiedlich ausgeprägt RTS, die von PLCs über Micro-Prozessoren bis zu einfachen bzw. vollwertigen PCs reichen.

5. Anwendungen

Real-Time-Systeme können **eingebettete** oder **nicht eingebettete** Systeme sein. Ein **eingebettetes System** ist immer Teil eines größeren Systems oder einer Maschine:

- Häufig befindet sich die Software für ein eingebettetes System auf einer Karte, ROM, EPROM oder FLASH. Über eine digitale Schnittstelle nutzen Anwendungen wie Waschmaschinen, Mikrowellen, Video-Geräte oder Motorsteuerungen diese Software.
- Dabei kann ein (Realzeit-)Betriebssystem zur Verfügung stehen oder die Funktion so einfach sein, das nur ein einziges Programm ohne Betriebssystem läuft.
- Eingebettete Systeme sind zu finden in
  - Autos: ABS, ESP, Zündung,....
  - Mobiltelefonen
  - Heim-Elektronik: TV, Video, Waschmaschine,...
  - Robotern
  - Gebäude- Automatisierung (Klima, Alarm,....)

5. Anwendungen

Nicht-eingebettete Systeme sind:

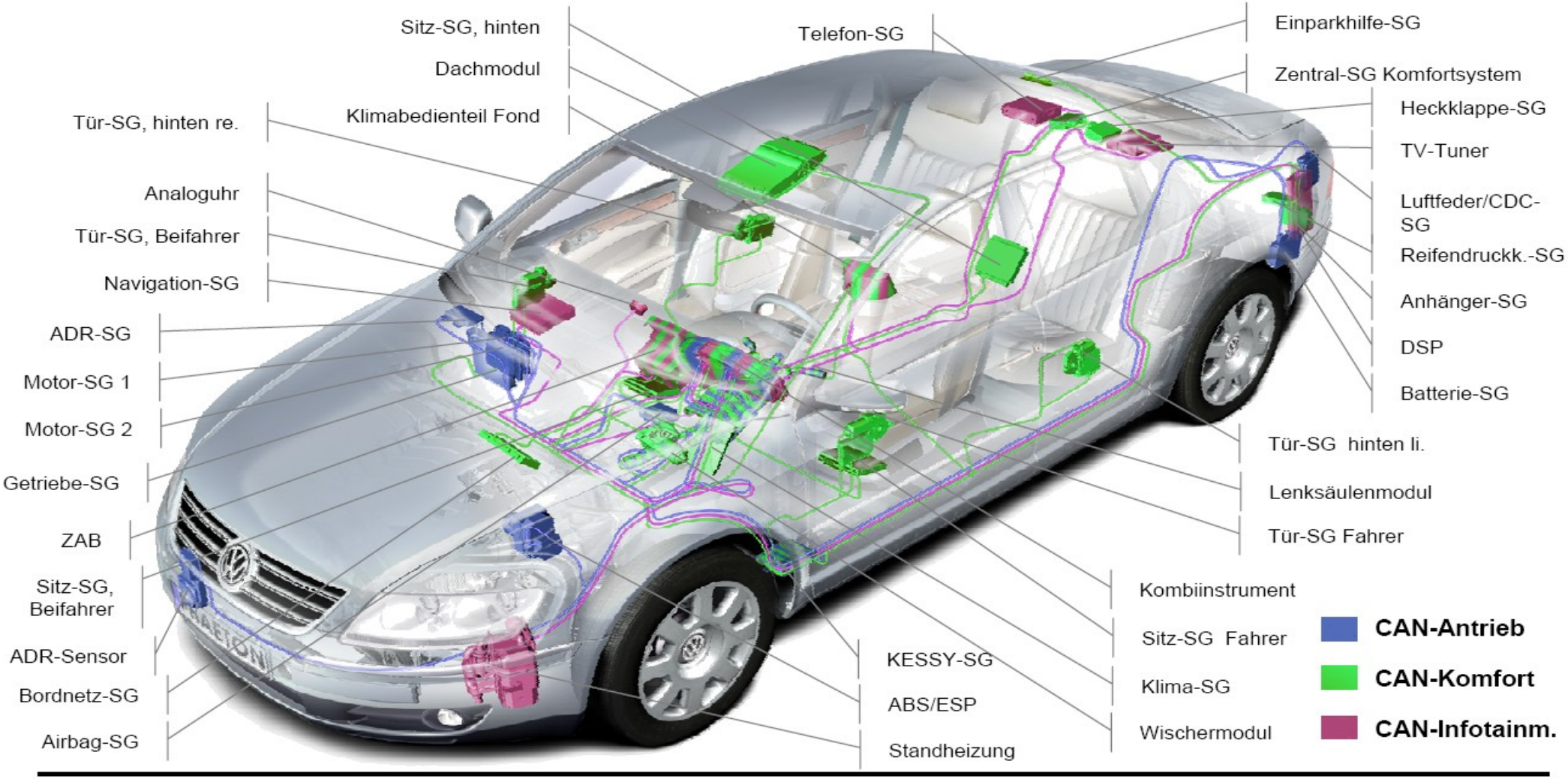
- Telefon-Systeme
- Fertigungsüberwachung und -steuerung (Stahl, Papier, Chemie)
- Elektizitäts-Netzwerke
- PCs
- Bank-Systeme
- .....

Dabei mögen die genannten Systeme wiederum eingebettete Teilsysteme enthalten. Insgesamt gibt es eine Vielzahl von Anwendungen aus dem RTS-Bereich.

5. Anwendungen

5.1. Automobile

In einem modernen Fahrzeug werden Dutzende von **vernetzten** Steuergeräten (ECUs) verbaut, viele davon mit Echtzeit-Anforderungen.



5. Anwendungen

5.2. Automatisierungstechnik

Fertigungsprozesse selbst sind oft zeitkritisch. Hinzu kommt die Steuerung und Regelung einzelner Maschinen. Es kommt eine Vielzahl von verschiedenen Computer-Systemen zum Einsatz.

5.3. Audio

Das menschliche Gehör ist feinfühlig genug, bereits Abweichungen von 0,5 ms wahrzunehmen. Entsprechend hoch sind die Anforderungen bei der Aufnahme und Wiedergabe von Musik mit entsprechenden Endgeräten im Heim-,Studio- und PC-Bereich.

5.4. Computer-Spiele

Das Problem, 30Bild/s zu erreichen, wird von Spiele-Produzenten meist mit (zu) hoher Prozessorleistung entgegen getreten. Werden die Spiele in einem Netzwerk (per Internet) zusammen geschaltet, entsteht dort ein Bottleneck. Dies lässt sich normalerweise umgehen, indem nicht komplette **Videoströme** versendet werden, sondern bestimmte Daten (wie Position, Δ Bewegung), die helfen das Bild lokal zu erzeugen **(rendern).**

5. Anwendungen

5.5. Video

Lange wurde Video nicht als RT-Anwendung betrachtet, doch kann Video-Software und DVD/B-HW definitiv als (Soft) RT aufgefasst werden. Die Hersteller von Home-Entertainment-Geräten sind dabei besonders auf die makellose Qualität ihrer Produkte angewiesen und können sich daher keine Ruckler erlauben.

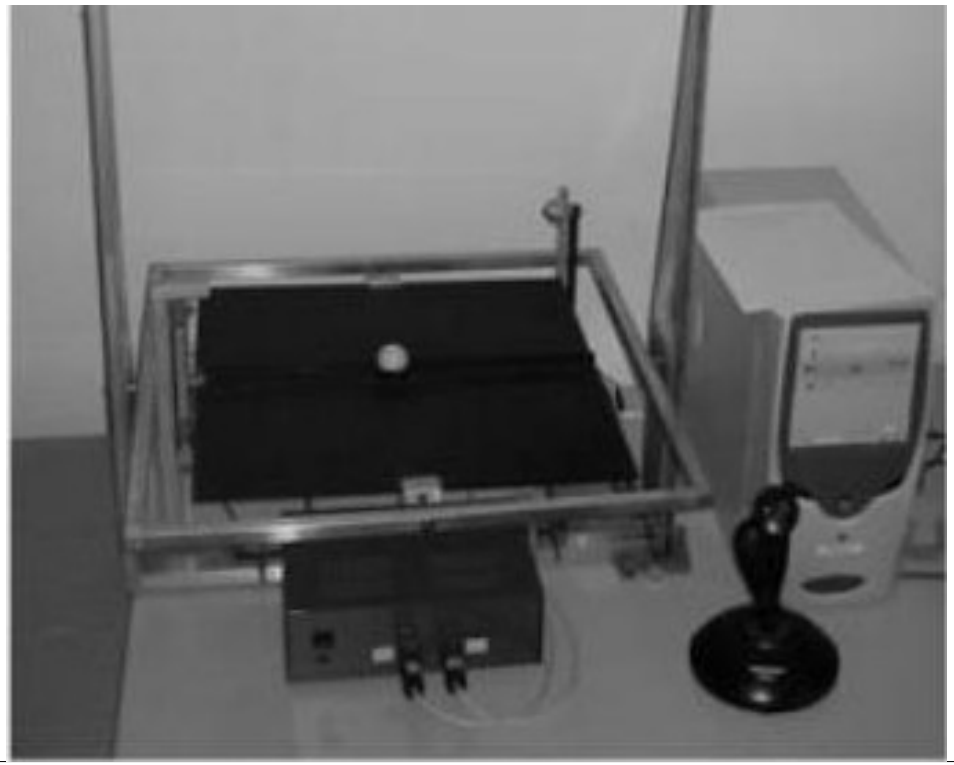
5.6. Verteilte Multimedia

Verschiedene Multimedia-Anwendungen sind oft über ein Netzwerk miteinander verbunden. Ein Beispiel ist ein Video-on-Demand System, in dem ein Hotelgast von einem Server einen Film abruft und ihn **sofort** ansieht. Oft verhält sich dabei das Netzwerk **unvorhersagbar**, so dass es beim **Streaming** zu Störungen kommen kann.
In der Welt von nicht-eingebetteten Computer-Netzwerken wird anstelle von RT der Begriff **Quality-of-Service** benutzt. QoS-Mechanismen beinhalten meistens Multimedia-Datenströmen einen verbesserten Service zu geben, sie also zu bevorzugen. Da die unterliegenden Netzwerk-Protokolle (z.B. IP) sich **nicht-deterministisch** verhalten, sind garantierte Übertragungszeiten normalerweise nicht möglich.

5.7 Beispiele

1. Wippe [5]: Im Mittelpunkt steht eine Kugel, die auf einer Fläche gehalten werden soll:

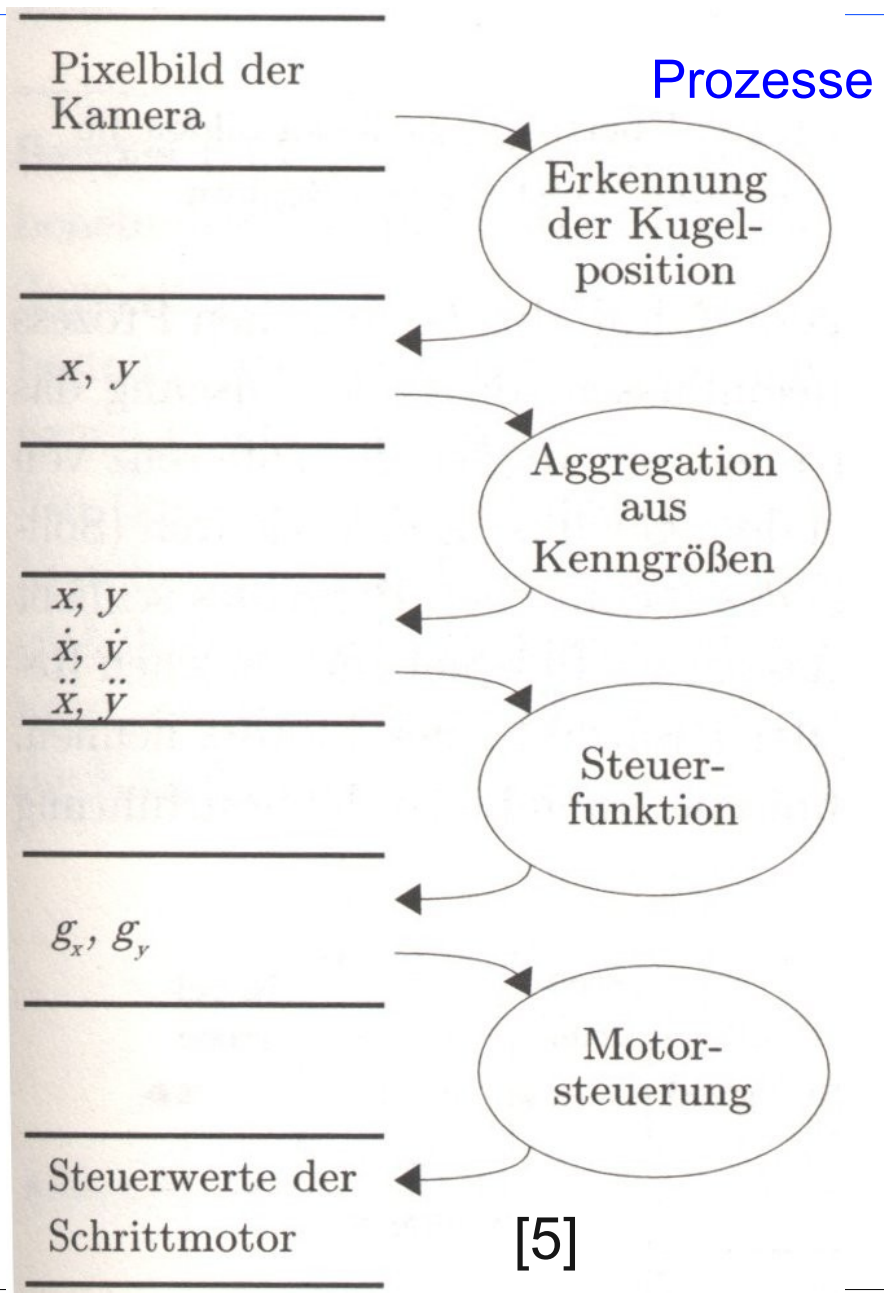
- Die Fläche ist beweglich und kann gezielt um 2 Achsen gedreht werden.
- Am oberen Ende des Rahmens erfasst eine Digital-Kamera Fläche und Kugel.
- Ziel ist es eine geworfene Kugel aufzufangen und in der Mitte der Fläche zum Stehen zu bringen.



5. Anwendungen

Ein **Datenfluss-Diagramm** stellt die notwendigen Prozesse dar. Dabei werden Daten unter Zeitbedingungen zustandsabhängig weiter verarbeitet und wieder anderen Prozessen zur Verfügung gestellt.

- Angestoßen wird der Fluss mit jedem neuen Bild.
- Die Erkennung der Position liefert (x,y)-Koordinaten.
- Bei der Aggregation aus Kenngrößen wird aus vorherigen Positionen auf die aktuelle Bewegung geschlossen (Bewegungsdaten).
- Aus diesen errechnet die Steuerfunktion die Neigungen.
- Diese werden schließlich durch die Motorsteuerung in Stellwerte umgerechnet.



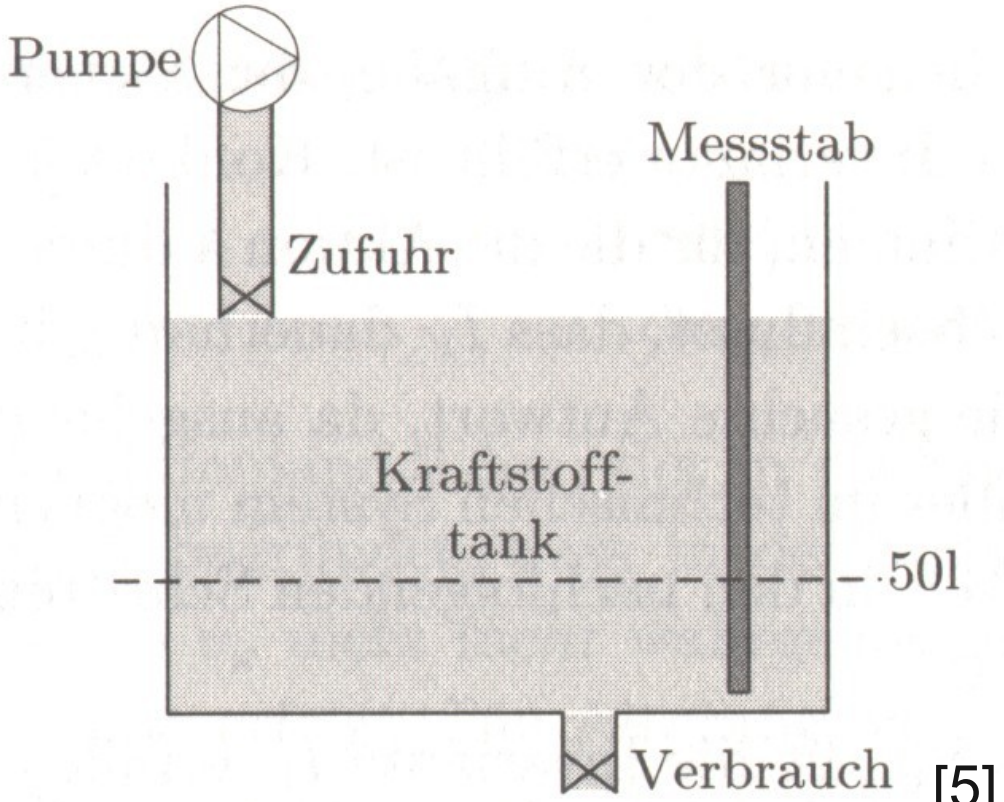
[5]

2. Füllstand-Regelung

Das Ziel ist, den Füllstand eines Kraftstofftanks nicht unter eine gewisse Schwelle sinken zu lassen:

- Ständig wird durch einen Motor Kraftstoff verbraucht.
- Über einen Messstab kann der Füllstand abgefragt werden, um bei Bedarf einen Pumpe an zuschalten, die den Tank wieder befüllt.

Datenfluss-Diagramm?



[5]

5. Anwendungen

3. Autonomes Fahrzeug

- Ein autonomes Fahrzeug hat die Aufgabe einer vorgegebenen Linie zu folgen.
- Das Fahrzeug verfügt über einen Sensor, mit dem festgestellt werden kann, wie weit das Fahrzeug von der Linie entfernt ist.
- Der Lenkung wird dabei der Kurs (als Winkel in einem Koordinaten-System) mitgeteilt: Ist das Fahrzeug zu weit rechts, wird der Winkel nach links korrigiert und umgekehrt.
- Das Fahrzeug wird simuliert, d.h. es gibt (mindestens) 2 unabhängige Prozesse:
  - das Fahrzeug selbst
  - die Darstellung auf dem Bildschirm

Dieses Beispiel wird später (im Praktikum) noch einmal aufgegriffen.

Aufgaben:

- Datenfluss-Diagramm
- Überlegen Sie sich einen Algorithmus für die Regelung, d.h. das Verhalten des Fahrzeugs!

5. Anwendungen

5.7 Zusammenfassung

So unterschiedlich die genannten Anwendungen auch sein mögen – eines haben sie alle gemeinsam: Es gibt zumeist **periodische Zeit-Restriktionen**, die eingehalten werden müssen/sollten und zwar auf

- **Endgeräten**/Knoten und
- einem ggf. vorhandenen **Netzwerk**.

In der Praxis wird dies oft dadurch gelöst, die Hardware der Systeme zu **überdimensionieren**, was **sehr teuer** ist, denn die meisten Anwendungen sind Gegenstand von **Massenproduktionen**.

Dagegen lernen Sie bei dieser Veranstaltung, wie man Real-Time-Probleme, bereits beim Entwurf **konzeptionell verhindert.** Dies wird hauptsächlich mit Hilfe von **Scheduling** (also der zeitlichen Anordnung von Teilaufgaben der Ressourcen CPU+ Netzwerk) gemacht.

6. Was ist das Problem?

6.1 Laufzeiten, Prioritäten, Scheduling

Um Aussagen über das Zeitverhalten machen zu können, muss man (selbstverständlich) die **Laufzeiten** der beteiligten Tasks ermitteln. Dies hat sich zu einer eigenständigen Disziplin entwickelt und wird **WCET-Analyse** genannt (WCET = Worst-Case Execution Time).

Sind **mehrere unabhängige Tasks** beteiligt, konkurrieren diese um die CPU, so dass der Zugriff darauf geregelt werden muss (**Scheduling**).

Dies geschieht mit Hilfe von **Prioritäten**: Von den gerade *bereiten* Tasks wird immer die mit der höchsten Priorität ausgewählt.

6. Was ist das Problem?

Aber woran kann sich die **Vergabe von Prioritäten** orientieren?

Offensichtlich sollten

- wichtige Tasks die höchste **Priorität** bekommen

  (aber was heißt eigentlich „wichtig“?), oder
- vielleicht besser die, die eine **kurze Antwort** erfordern?

Insgesamt ist dies also eine schwierige Fragestellung. Wir werden im nächsten Kapitel sehen, wie

- die Prioritätsvergabe **analysiert** werden kann und
- wie Prioritäten **automatisch** vergeben werden können.

Das Ziel wird sein, allen Tasks eine bestimmte Antwortzeit zu garantieren.

6. Was ist das Problem?

Real-Time-Anwendungen werden häufig mit Hilfe eines **Echtzeit-Betriebssystems** (Real-Time Operating System – RTOS) betrieben. Ein RTOS hat normalerweise recht **schlank** zu sein, um für Eingebettete Systeme geeignet zu sein. Es verhält sich selbst **vorhersagbar** und benutzt den folgenden Algorithmus zur Vergabe von Ressourcen:

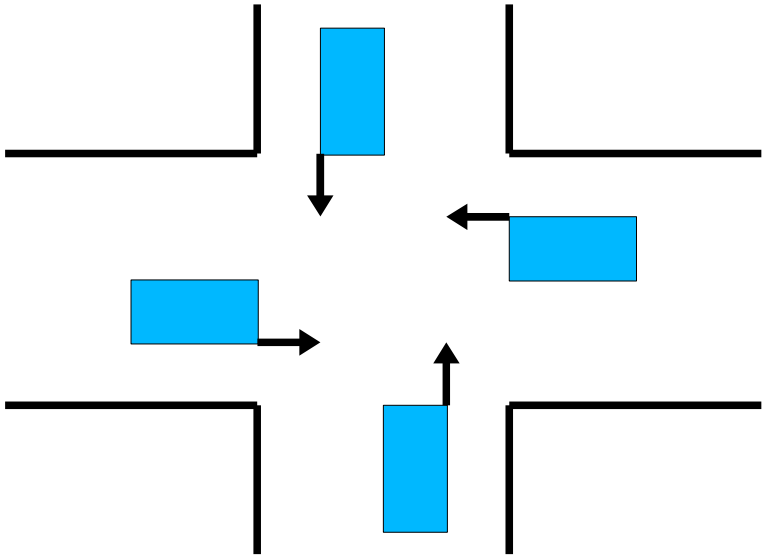
- Jede CPU-Task (Prozess, Thread, Job) bekommt eine Priorität.
- Eine Task mit höherer Priorität wird gegenüber einer Task mit niedriger Priorität bevorzugt.
- Bei jedem Clock-Tick werden die Prioritäten neu geprüft, denn sie könnten sich geändert haben bzw. eine neue Task mit noch höherer Priorität könnte dazu gekommen sein (**preemptives** Scheduling).
- Die Kommunikation zwischen verschiedenen Tasks erfolgt über Warteschlangen, in die Datenpakete gelegt werden, um vom Empfänger konsumiert zu werden.

6. Was ist das Problem?

6.2 Deadlocks

Ein Deadlock entsteht, wenn zwei (ober mehr) Tasks sich gegenseitig blockieren, weil jeweils auf eine geschützte aber belegte Ressource zugreifen wollen. **Beispiel** aus der realen Welt:

4 Autos (Porzesse/Threads) gleichzeitig an Kreuzung ohne Vorfahrtsschilder



Der entstehende **Stillstand** ist besonders für RTS schwer zu verkraften. In Computer-Systemen können Deadlock deswegen entstehen, weil Ressourcen durch bestimmte Mechanismen **geschützt (d.h. gesperrt)** werden (müssen). Dies kann **z.B.** durch die Benutzung von **Semaphoren** gemacht werden. Ist eine Ressource geschützt, **blockiert** jede Task, beim Versuch das Semaphoren zu passieren.

6. Was ist das Problem?

Ein einfacher und pragmatischer Algorithmus, Deadlocks zu vermeiden, ist immer dieselbe Vergabe-Reihenfolge zu erzwingen. Wir werden später sehen, dass dennoch das Problem entstehen kann, dass dabei eine hoch priorisierte Task durch eine niedrig priorisierte Task an ihrer Arbeit gehindert wird.

6.3 Inkonsistente Daten

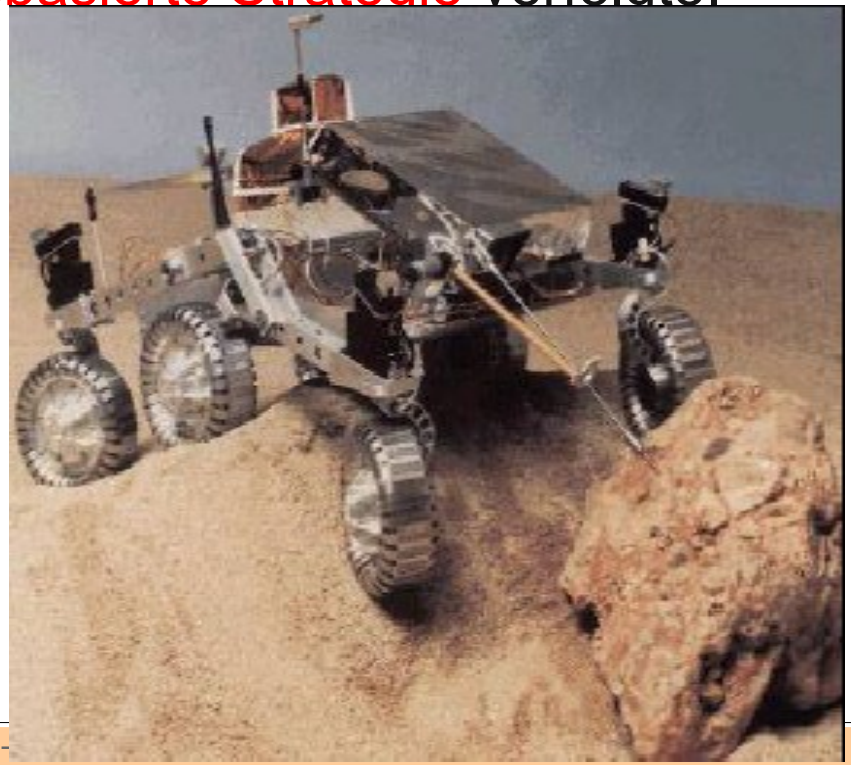
Zu inkonsistenten Daten kann es trotz Schutzmechanismen wie Semaphoren kommen, wenn zwei Knoten eines Netzwerks **nicht-synchronisierte Uhren** besitzen, denn dann trifft der Konsument einer bestimmten Information ggf. eine falsche Entscheidung, weil womöglich ein falscher Zeit-Stempel dafür vorliegt.

6. Was ist das Problem?

6.4 Beispiel: Der Mars-Pathfinder

Der ***Pathfinder*** war ein autonomer Roboter, der zum Mars geschickt wurde, um meteorologische Daten und Bilder zu sammeln und zu verschicken. Die Mission war solange erfolgreich, bis nach ein paar Tagen – inmitten der Ermittlung von Daten – sich das Computer-System plötzlich neu startete.

Pathfinder benutzte ein RTOS namens *VxWorks* (Windows war diesmal also schuldlos), das eine prioritätsbasierte Strategie verfolgte.



6. Was ist das Problem?

Als Grund des Übels stellte sich das Zusammenspiel der folgenden **3 Tasks** heraus:

- Eine **niedrig** priorisierte **Task L** zur Sammlung von Daten, das das Kommunikationsnetzwerk zum Verschicken davon nutzte. Um letzteres zu tun, sperrte es zuerst ein Semaphor, um dann Daten auf den Bus zu legen.
- Eine **hoch** priorisierte **Task H** lief in kurzen Abständen, um Daten auf und vom System zu bringen. Eines Tages konnte **H** nicht auf dem Bus zugreifen, da dieser noch von **L** gesperrt/benutzt wurde, und **blockierte**.
- Eine **mittelhoch** priorisierte **Task M**, deren genaue Funktion hier unwichtig ist, die aber eine lange Berechnungszeit aufwies. **M** verdrängte **L** von der CPU.

Das prinzipielle Problem dabei ist, dass nun eigentlich **H** zum Zuge hätte kommen müssen, dies aber wegen der Blockade des Busses durch das inzwischen untätige **L** nicht möglich ist war. **H** und **M** „tauschten“ also ihre Prioritäten **(Priority Inversion Problem).**

6. Was ist das Problem?

Als Folge davon glaubte ein Überwachungsprozess, dass irgend etwas Fatales geschehen sei (weil sich **H** schon so lange nicht mehr gemeldet hatte) und startete das System neu.

Durch dieses Problem drohte die Mission zu scheitern. Erst durch eine entsprechende Analyse und das Einspielen neuer Software wurde die Panne behoben.

Man hätte also vorher jemanden Fragen sollen, der sich damit auskennt ...

7. Literatur

[1] Andy Wellings, Alan Burns: ***Real-Time Systems and Programming Languages*** – *third edition,* Pearson / Addison Wesley

[2] Christer Norström, Kristian Sandström, Jukka Mäki-Turja, Hans Hansson, Henrik Thane, Jan Gustafsson, Damir Isovic: ***Robusta Realtidsystem***, Mälardalen Real-Time Research Centre (schwedisch ist nicht schwer! ☺)

[3] Giorgio C. Buttazzo: ***Hard Real-Time Computing Systems***, Kluwer AP

[4] Andrew S. Tanenbaum: ***Moderne Betriebssysteme***, 3., aktualisierte Auflage, Pearson Studium – Prentice Hall

[5] Dieter Zöbel: ***Echtzeitsysteme – Grundlagen der Planung***. Springer-Verlag 2008