Echtzeitsysteme

1. Einführung

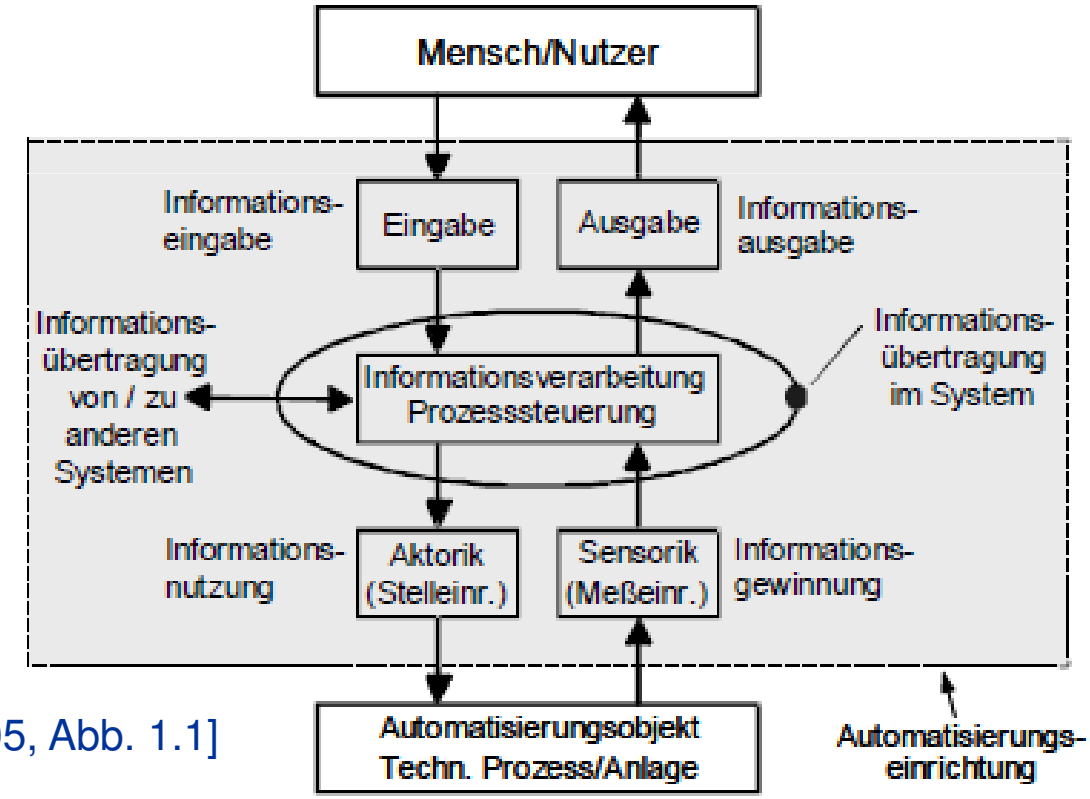
Prof. Dr. Roland Dietrich

Ziele der Vorlesung

- Verständnis der besondere Anforderungen von Echtzeitsystemen (engl. *real-time systems*) im Vergleich zu anderen (nicht-Echtzeit-) Systemen
- Kenntnis wesentlicher Methoden, diese Anforderungen zu erfüllen
  - Planungsmethoden (*real-time scheduling*)
  - Aufgaben von Echtzeitbetriebssysteme (*real-time operation systems, RTOS*)
  - Konzepte von Programmiersprachen zur Unterstützung von Echtzeitanforderungen

Begriffsbildung

- **Eingebettete Systeme** (*embedded systems*)
  - Rechner ist Bestandteil eines größeren technischen Systems
  - Rechner ist über Schnittstellen (Sensoren und Aktoren) mit dem technische System verbunden
  - Zusätzlich möglich: Interaktion mit einem Nutzer und/oder anderen Systemen
  - Aufgabe des Rechners: Überwachen und Steuern der technischen Prozesse (→ „Prozessrechner“)
  - *Echtzeitsysteme sind überwiegend eingebettete Systeme*



[Quelle: Wörn & Brinkschulte 2005, Abb. 1.1]

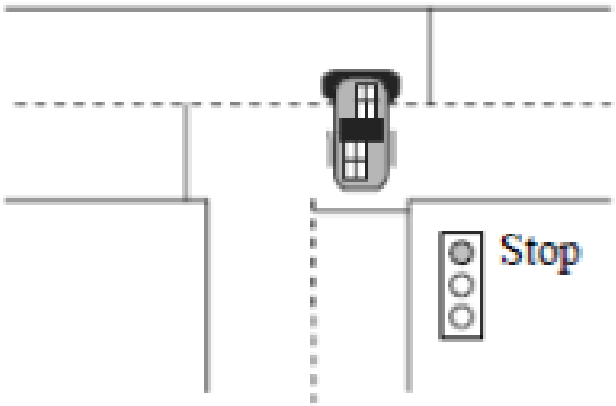
Begriffsbildung

- **Echtzeitsystem** (EZS)?
  - DIN 44300

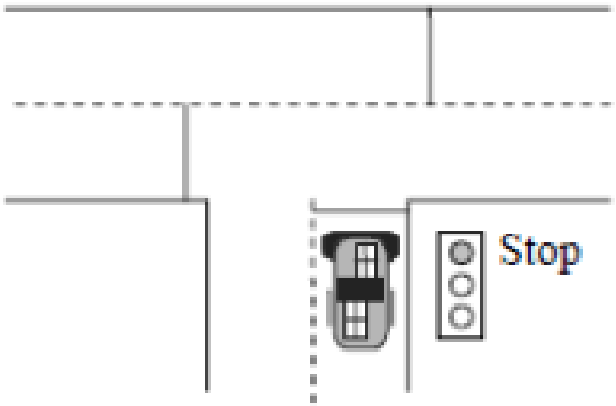
    Betrieb eines Rechnersystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind.
  - Anforderungen an ein EZS nach [*Wörn, Brinkschulte 2005*]
    - **Rechtzeitigkeit**: Ein Ergebnis für einen zu steuernden Prozess muss *rechtzeitig* vorliegen
    - **Gleichzeitigkeit**: viele Anforderungen müssen *parallel* (gleichzeitig), jede mit ihren eigenen Zeitanforderungen ausgeführt werden
    - **Spontane Reaktion auf Ereignisse:** ein EZS muss auf zufällige und spontan eintretende Ereignisse (innerhalb oder außerhalb des Prozesses) innerhalb einer definierten Zeit reagieren.
  - Konsequenzen:
    - Die Korrektheit eines EZS hängt nicht nur vom logischen Ergebnis der Berechnungen ab, sondern auch vom Zeitpunkt, an dem sie vorliegen
    - Nicht rechtzeitige Berechnung ist oft so schlecht wie falsche Berechnung

Begriffsbildung

- Logische vs. zeitliche Korrektheit
  - Nicht-Echtzeitsysteme: korrekt = logisch korrekt
  - Echtzeitsysteme: Korrektheit = logisch korrekt + zeitlich korrekt
- Beispiel: Steuerung eines autonomen Fahrzeugs
  [Quelle: Wörn & Brinkschulte 2005, Abb. 5.1]



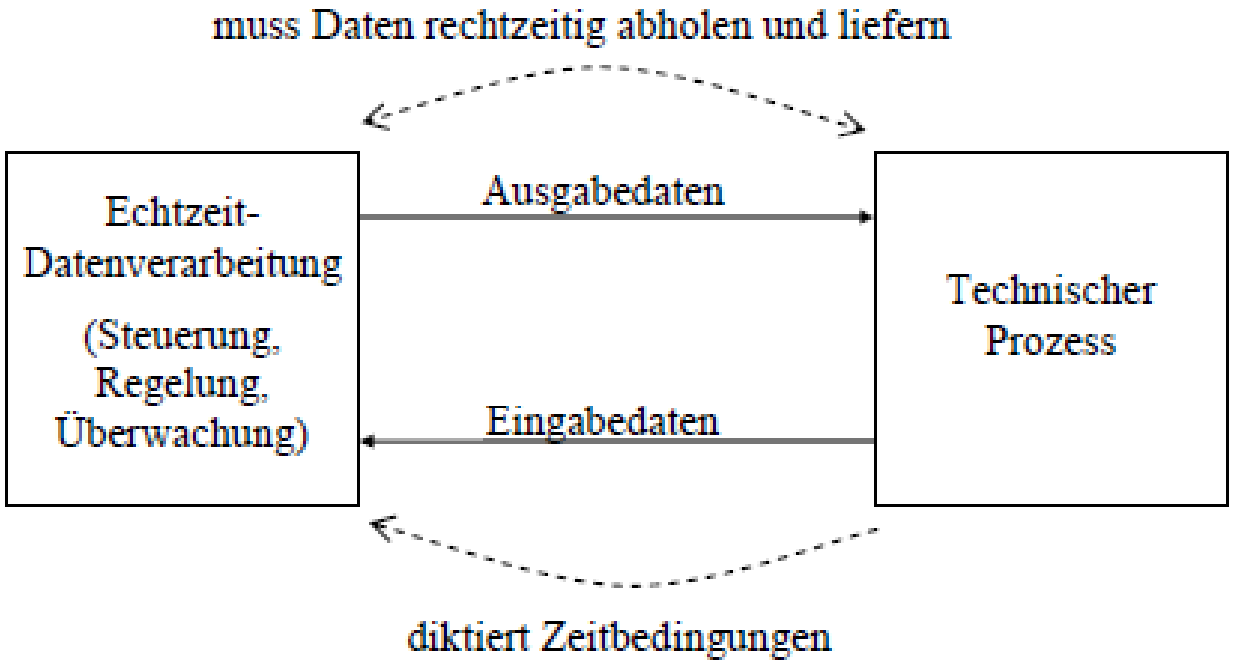
Logisch korrekt, zeitlich inkorrekt



Logisch und zeitlich korrekt

Begriffsbildung

- **Rechtzeitigkeit**
  - Ausgabedaten müssen „rechtzeitig“ berechnet werden
  - Eingabedaten müssen „rechtzeitig“ zur Verfügung stehen
  - Was „rechtzeitig“ ist, definiert der Technische Prozess!



[Quelle: Wörn & Brinkschulte 2005, Abb. 5.2]

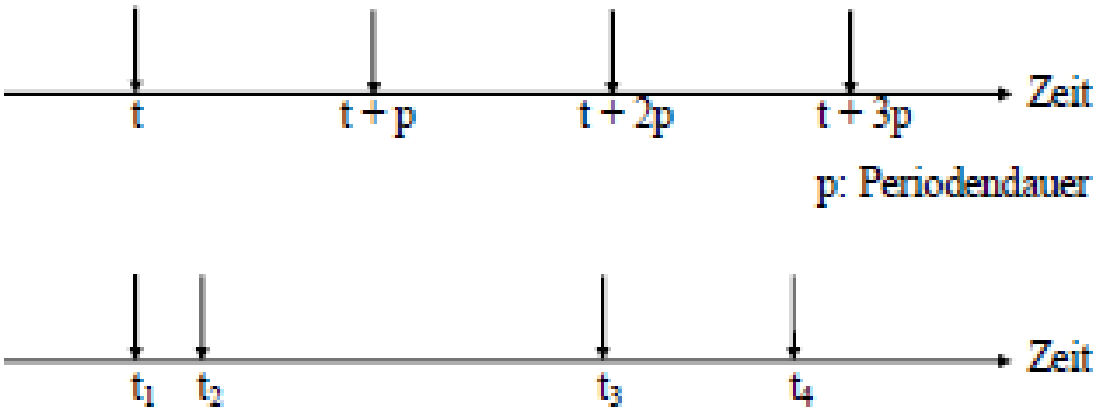
Begriffsbildung

- Wann ist eine Aktion *A* **rechtzeitig**?
  - **a)** **Genauer Zeitpunkt:** Es wird ein Zeitpunkt ***t*** vorgegeben, an dem *A* ausgeführt werden muss
  - **b)** **Spätester Zeitpunkt:** Es wird ein maximaler Zeitpunkt t_{max} vorgegeben, *bis zu dem A* spätestens ausgeführt sein muss
  - **c)** **Frühester Zeitpunkt:** Es wird ein minimaler Zeitpunkt t_{min} vorgegeben, *vor dem* A nicht ausgeführt werden darf
  - **d)** **Zeitintervall:** A darf nicht vor t_{min} ausgeführt werden und muss vor t_{max} ausgeführt sein.

[Quelle:
Wörn & Brinkschulte 2005, Abb. 5.3]

Begriffsbildung

- **Periodische** Zeitbedingungen:
  - wiederholen sich in *regelmäßigen* zeitlichen Abständen (*time triggered*)
- **Aperiodische** Zeitbedingungen:
  - treten in unregelmäßigen Abständen auf
  - meist durch **Ereignisse** ausgelöst (*event triggered*)



periodische

Zeitbedingungen

aperiodische

[Quelle: Wörn & Brinkschulte 2005, Abb. 5.4]

Begriffsbildung

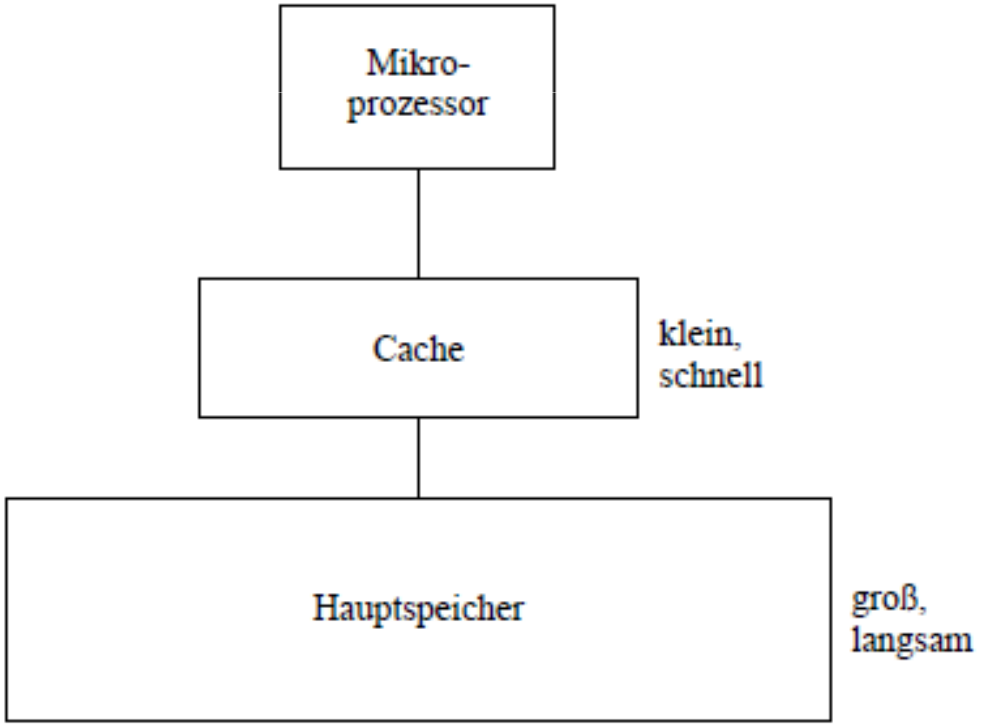
- **Absolute** Zeitbedingung
  - Angabe eines Zeitpunkts
  - *Bsp.: Ein Prozess muss um 12:00 Uhr gestartet werden*
- **Relative** Zeitbedingung
  - Angabe einer Zeitbedingung relativ zu einer vorherigen Zeitbedingung oder Ereignis
  - *Bsp.: Der Stellwert muss 0,5 sec. nach Messung eines Sensorwerts erzeugt werden*

Begriffsbildung

- **Anmerkungen**
  - Formen und Ausprägungen von Zeitbedingungen können bei Echtzeitsystemen in beliebiger Kombination auftreten!
  - Wesentlich für die Einhaltung von Zeitbedingungen sind
    - **Hinreichende Verarbeitungsgeschwindigkeit:** die Aktivitäten laufen schnell genug ab, um die Einhaltung der Zeitbedingungen zu ermöglichen
    - **Zeitliche Vorhersagbarkeit:** Das Einhalten der Zeitbedingungen kann in jedem Fall garantiert werden

      → ***Worst Case Execution Time*** ist entscheidend!
    - *Beispiel:*

      Mikroprozessor-Architektur mit Cache:



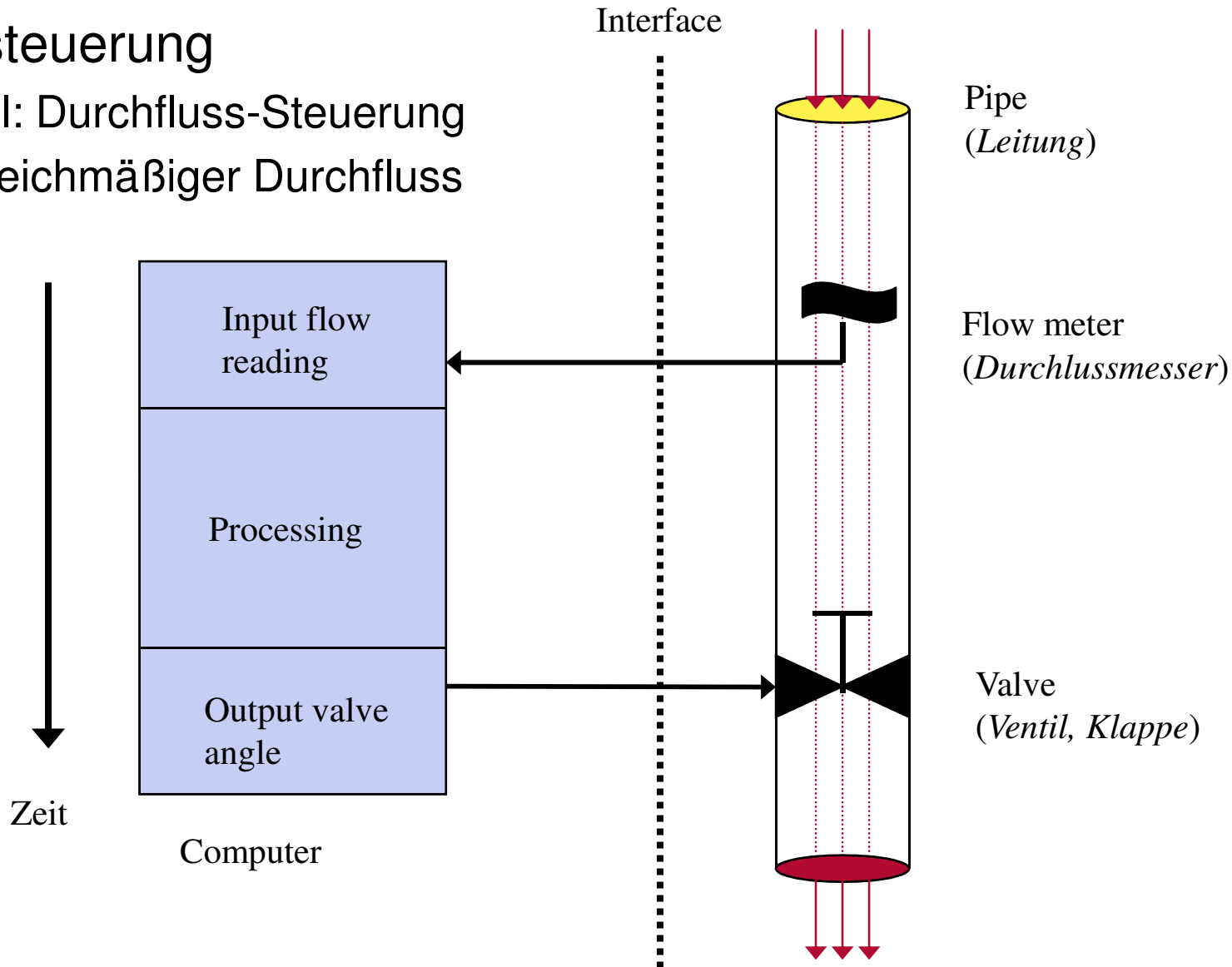
[Quelle: Wörn & Brinkschulte 2005, Abb. 2.11]

Begriffsbildung

- **Harte Echtzeitsysteme** (*hard real-time systems*)
  - Echtzeit-Anforderungen müssen unbedingt eingehalten werden
  - Falls nicht, droht Schaden!
    - Beispiel: autonomes Fahrzeug muss vor einer roten Ampel anhalten
- **Weiche Echtzeitsysteme** (*soft real-time systems*)
  - Echtzeit-Anforderungen sind wichtig, Einhaltung wünschenswert
  - Wenn Sie „gelegentlich“ nicht eingehalten werden, funktioniert das System dennoch korrekt
  - Typisch: Vorgaben für Toleranzen
    - Toleranzen für die Häufigkeit der Zeitfehler
    - Toleranzen für die Größe der Zeitfehler
  - Typisch: die Brauchbarkeit des Ergebnisses nimmt ab, je stärker die Abweichungen von den Echtzeit-Anforderungen
- **Feste Echtzeitsysteme** (*firm real-time systems*)
  - Nicht-Einhaltung der Echtzeit-Anforderungen unschädlich, macht aber das Ergebnis wertlos (Informationen haben „Verfallsdatum“)

Beispiele für EZS

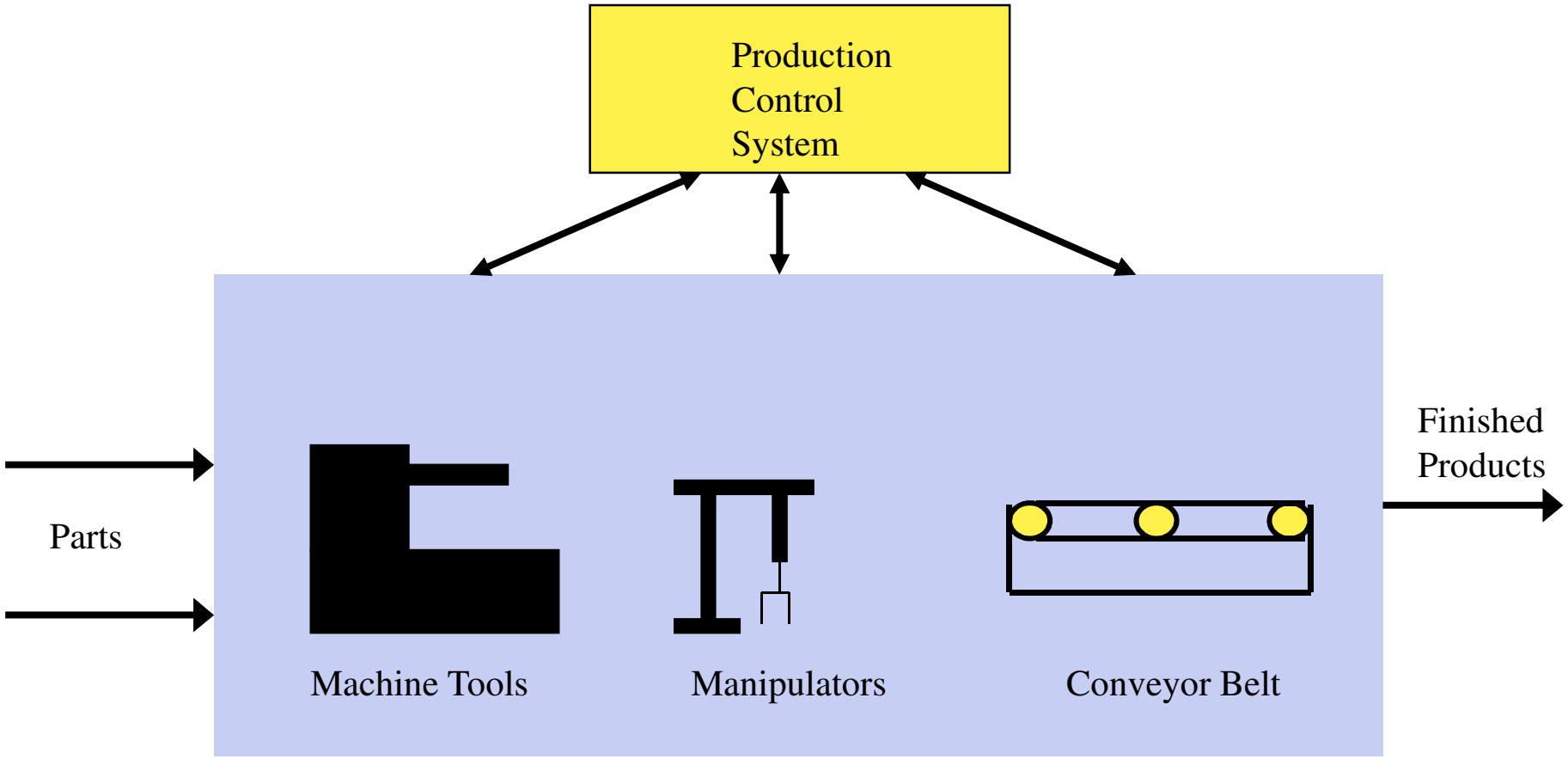
- Prozesssteuerung
  - Beispiel: Durchfluss-Steuerung
  - Ziel: Gleichmäßiger Durchfluss



[Quelle: Burns & Wellings 2009, Abb. 1.1]

Beispiele für EZS

- Produktionssteuerung
  - Computer steuert eine Menge von mechanischen Geräten



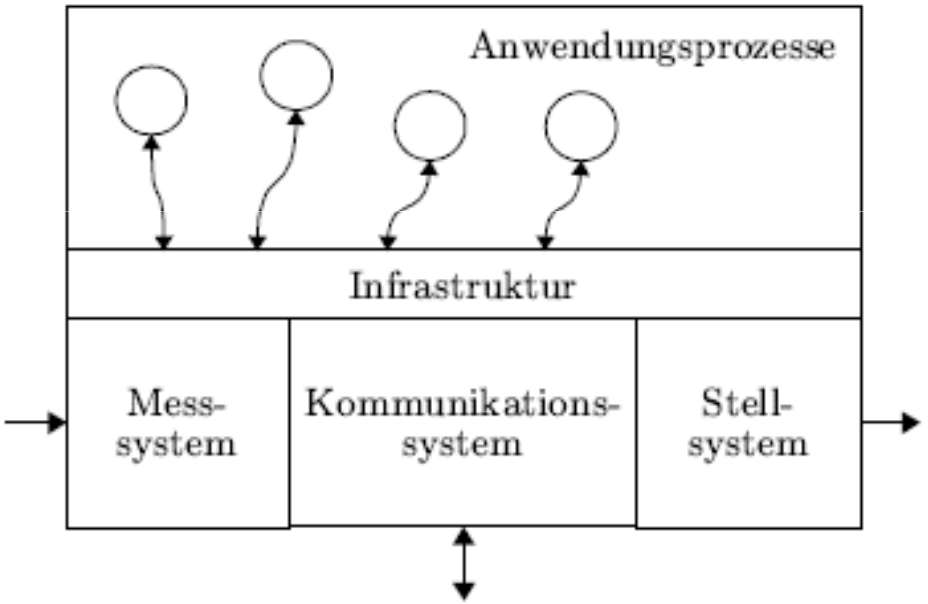
[Quelle: Burns & Wellings 2009, Abb. 1.1]

Beispiele für EZS

- Produktionssteuerung
  - Häufig: Im Produktionsprozess werden Roboter eingesetzt
  - Jeder Roboter hat seine eigene, unabhängige Steuerung für
    - Sensoren (z.B. Abstandssensoren)
    - Bewegliche Teilen (z.B. Räder, Gelenke, Greiffinger)
    - Bildverarbeitung und Mustererkennung
  - Jeder Roboter für sich ist ein Echtzeitsystem
    - Häufig: Ein *hartes* Echtzeitsystem
    - z.B. wenn mit Menschen mit Robotern in der selben physikalischen Umgebung zusammenarbeiten

Beispiele für EZS

- Automotive Systeme
  - ca. 30 – 100 vernetzte Steuergeräte in einem Fahrzeug
  - Aufgaben
    - Antrieb und Fahrwerk
    - Karosserie und Komfort
    - Telematik
    - Diagnose und Wartung

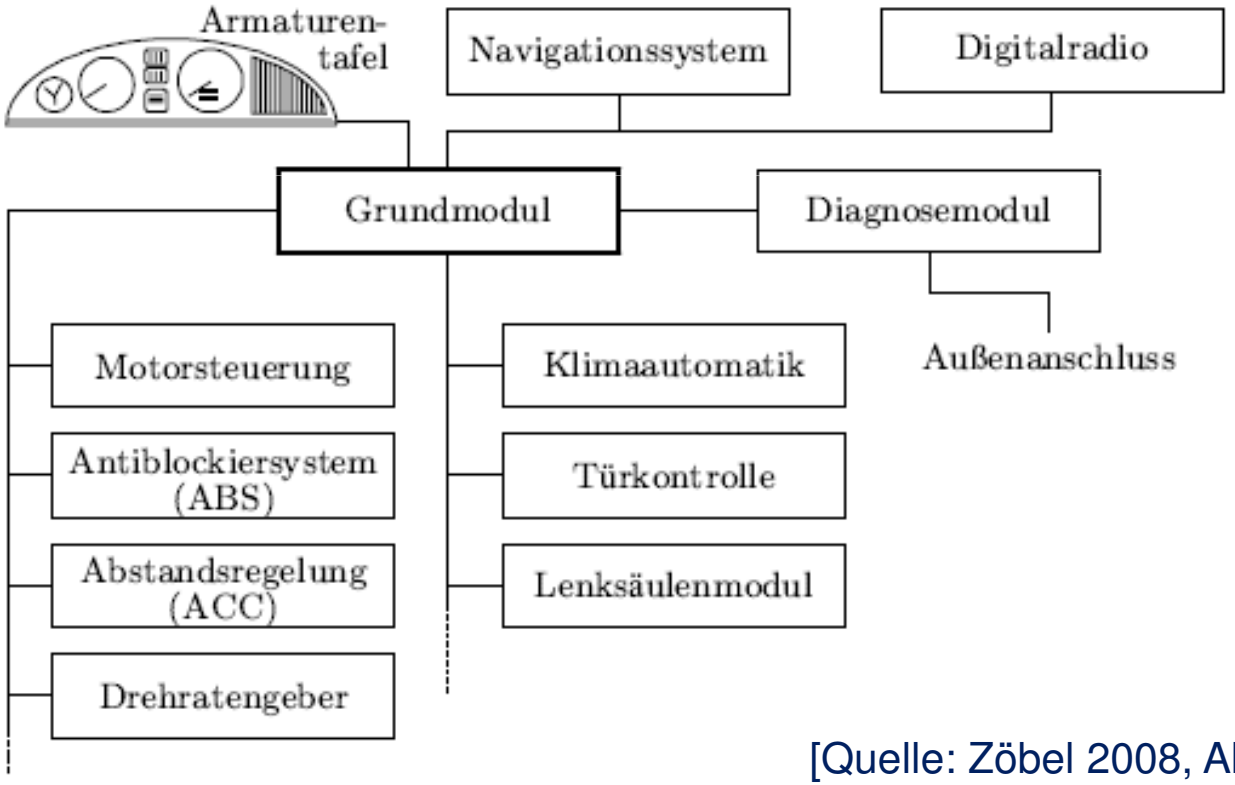


Blockdiagramm eines typischen Kfz-Steuergerätes

[Quelle: Zöbel 2008, Abb. 1.17]

Beispiele für EZS

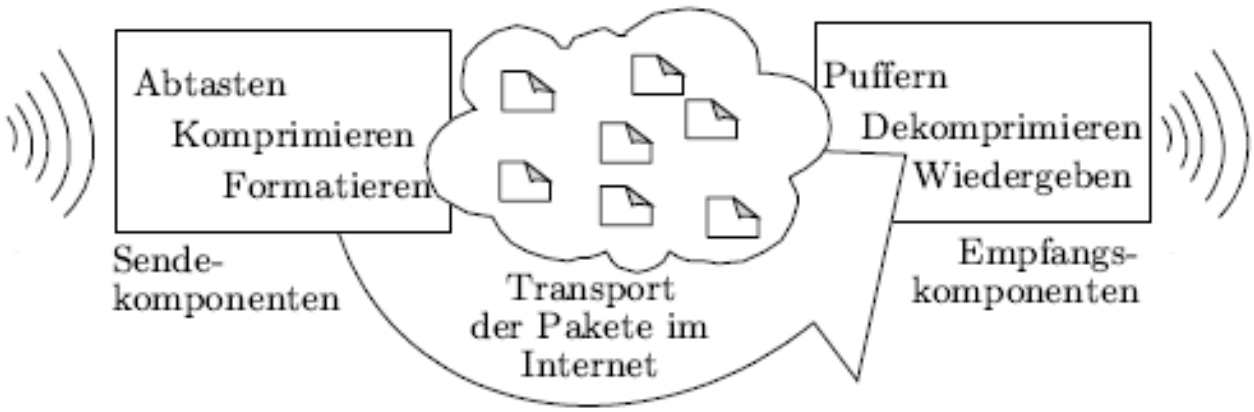
- Automotive Systeme
  - Typische Architektur von Steuergeräten im Fahrzeug
    - Von einem „Grundmodul“ sind mehrere Bussysteme sternförmig angeordnet
    - Jedes Bussystem verbindet eine Gruppe von Steuergeräten



[Quelle: Zöbel 2008, Abb. 1.18]

Beispiele für EZS

- Multimedia, z.B. Internet-Telefonie (*voice over IP*)
  - Sendekomponente digitalisiert analoge Signale und bildet Daten-„Pakete“
  - Empfangskomponente mit Puffer bildet aus digitalen Paketen analoge Signale
    - Problem: Pakete werden mit unterschiedlichen Verzögerungen übertragen (→ „Flattern“, engl. *jitter*)
    - Pakte, die „zu spät“ kommen, werden ignoriert (→ Datenverlust)



[Quelle: Zöbel 2008, Abb. 1.19]

Beispiele für EZS

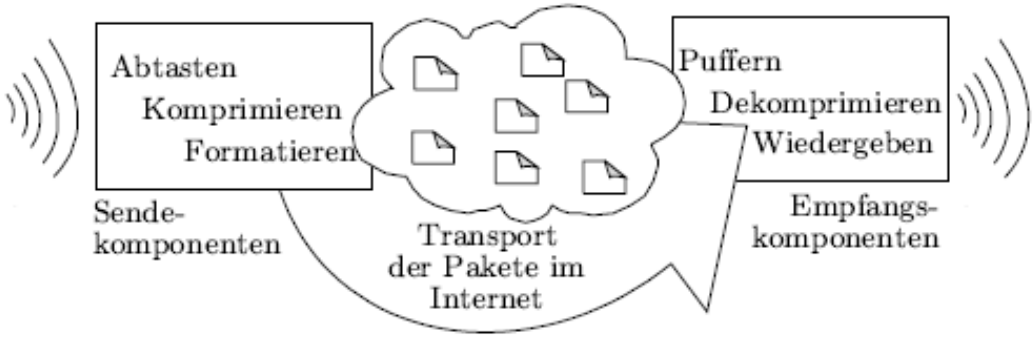
- Multimedia, z.B. Internet-Telefonie (*voice over IP*)
  - Vorgabe *International Telecommunication Union* (*ITU*): Verzögerung weltweit ≤ *150 ms*
  - Rahmenbedingungen:
    - 20.000 km Übertragung: ≥ *67ms*
    - Verarbeitung in Sende- und Empfangskomponente ca. *10 ms*

    →Maximal *70 ms* für

    Weiteleitung der Pakete in den Internet-Knotenrechnern

    Übertragung > 20.000 km

    Einreihung der Paket im Puffer der Empfangseinheit



[Quelle: Zöbel 2008, Abb. 1.19]

Anforderungen an EZ-Programmierung

- **Echtzeitfähigkeiten** (***real-time facilities***)
  - Zeitpunkt/Zeitraum spezifizieren für die Ausführung von Aktionen
  - Zeitpunkt/Zeitraum spezifizieren für die Vollendung von Aktionen
  - Systematische Wiederholung von Aktionen (periodisch/aperiodisch)
  - Flattern bei Ein-/Ausgabeoperationen begrenzen
  - Reaktionsmöglichkeiten, wenn Zeitanforderungen nicht eingehalten werden
  - Reaktionsmöglichkeiten, wenn sich Zeitanforderungen dynamisch ändern
- **Vorhersagbarkeit**
  - Das Laufzeitverhalten eines EZS muss sicher vorhersagbar sein
    → **Echtzeitplanung** (*real time scheduling*)

Anforderungen an EZ-Programmierung

- **Multitasking und Parallelität**
  - EZS steuern oft mehrere gleichzeitig existierende und reagierende externe Systeme
  - Mehrere Sensoren/Aktoren müssen gleichzeitig (parallel) bedient werden (Eingabe/Verarbeitung/Ausgabe)
  - Lösungen
    - Ein Prozessor für alles
      - Wenn der Prozessor leistungsfähig genug ist, um alle gleichzeitigen Abläufe nacheinander oder abwechselnd zu bedienen
      - Es wird nicht parallel verarbeitet, aber „es sieht so aus“ (d.h. die Zeitanforderungen werden trotzdem eingehalten!)

      → Quasi-Parallelität, Nebenläufigkeit (***concurrency***), ***Multi-Tasking***
    - Echte Parallelität
      - Parallele Abläufe werden auf mehreren Prozessoren ausgeführt

Anforderungen an EZ-Programmierung

- **Hardwarenahe Programmierung**
  - Interaktion mit einem technischen System erfordert Zugriff auf unterschiedlichste Aktoren und Sensoren
  - Häufig: Interaktion über Ein-/Ausgaberegister
  - Häufig: Interaktion über Unterbrechungen (*Interrupts*)
  - Typisch: Diese Dinge übernimmt das Betriebssystem
    - Zugriff auf Gerät = Aufruf einer Betriebssystem-Funktion
  - Problem: manchmal zu ineffizient
    - jeder Funktionsaufruf kostet Zeit und Ressourcen
  - Lösung: Direkte Programmierung in Assembler-Sprache

→ EZ-Programmiersprache muss sowohl Betriebssystem- als auch Hardware-Zugriffe unterstützen

Anforderungen an EZ-Programmierung

- **Numerische Berechnungen**
  - Erforderlich für diskrete/analoge Regelungssysteme im Sinne der Regelungstechnik
  - Häufig: Numerische Lösung von Differentialgleichungen erforderlich
- **Software-Komplexität**
  - Code Umfang von EZS:
    - Ein paar 100 Zeilen Assembler oder C
    - Mehrere Millionen Zeilen ADA
  - Die Technische Umgebung ändert sich
    - Anpassungen/Veränderungen am EZS erforderlich

  → Programmiersprache muss SW-Strukturierung unterstützen
- **Fehlertoleranz und Sicherheit**
  - Hohe Korrektheitsanforderungen (logisch + zeitlich!)
  - Fehler passieren trotzdem

  → Fehlerbehandlung erforderlich

EZ-Programmiersprachen

- Allgemeine Design-Kriterien für Programmiersprachen
  - Sicherheit
    - Entdecken von Programmierfehlern durch Compiler oder Laufzeitsystem
    - Entdeckung logischer Programmierfehler leider nicht möglich!
  - Lesbarkeit
    - Ein Quell-Programm sollte ohne weitere Hilfsmittel durch „Lesen“ verstanden werden können
    - Einflussfaktoren: Wahl von Schlüsselwörtern und Bezeichnern, Typdefinitionen, Modularisierungsmöglichkeiten,…
  - Flexibilität
    - Alle erforderlichen Operationen sollten auf einfache Art und Weise ausgedrückt werden können
    - Ohne Betriebssystemaufrufe oder Einbau von Assembler-Code
  - Einfachheit
    - Korreliert mit Ausdrucksstärke (Einfach → weniger Ausdrucksstark)
    - Korreliert mit Nutzbarkeit (Einfach → leichter zu benutzen)

EZ-Programmiersprachen

- Allgemeine Design-Kriterien für Programmiersprachen
  - Portabilität
    - Ein Programm sollte unabhängig von der HW sein, auf der es läuft
    - Für eingebettete und Echtzeitsysteme schwer zu erreichen!

    → HW-abhängige von HW-unabhängige Code-Teilen isolieren!
  - Effizienz
    - In EZS müssen Antwortzeiten garantiert werden
    - Effizientes und Vorhersagbares Programm-Verhalten erforderlich

    → Auf Mechanismen mit nicht vorhersagbarem Laufzeit-Overhead verzichten!

EZ-Programmiersprachen

- **Assemblersprachen**
  - Symbolische Repräsentation der Maschinensprache
    - D.h. HW-orientiert, nicht problemorientiert
  - Ursprünglich **die** Sprache für eingebettete und Echtzeitsysteme
    - Auf Mikrorechnern meist sonst nichts verfügbar
    - Einzige Möglichkeit, effizient auf HW zuzugreifen
  - Probleme:
    - Programme schlecht lesbar
      → hohe Entwicklungskosten, schlechte Wartbarkeit
    - Programme nicht portierbar
    - bei HW-Wechsel muss Personal neu geschult werden

EZ-Programmiersprachen

- **Sequentielle höhere Programmiersprachen**
  - Beispiele: C, C++, einige spezielle Entwicklungen
  - Positive Merkmale:
    - Maschinenunabhängig
    - Strukturiertes Programmieren möglich
  - Mängel:
    - Keine Parallelität/Nebenläufigkeit
    - Schlechte Echtzeit-Unterstützung
    - Keine Unterstützung von Zuverlässigkeit (z.B. Ausnahmebehandlung)

    → Häufig Betriebssystemaufrufe

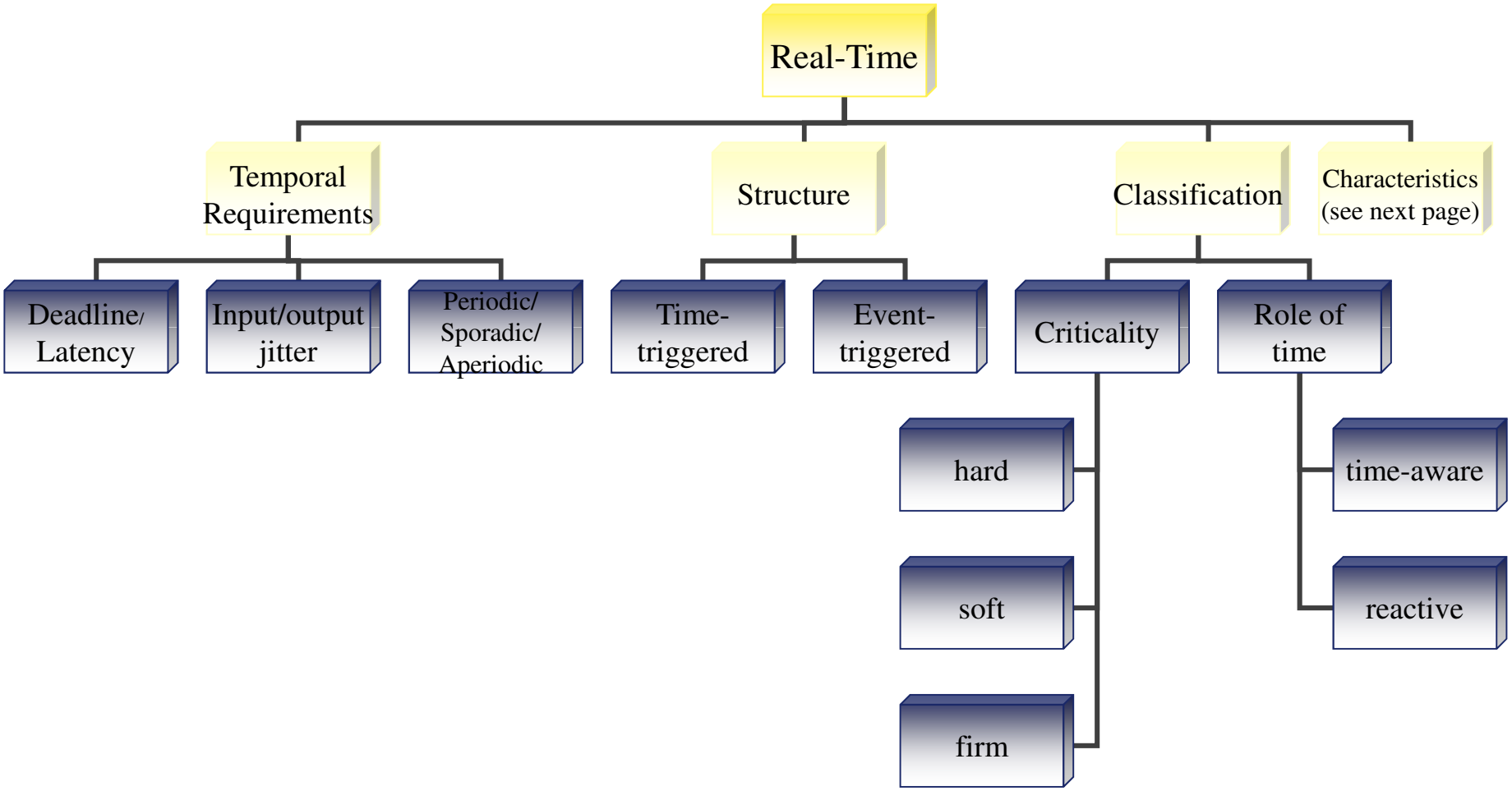
    → Einbau von Assemblercode erforderlich
- **Höhere Programmiersprachen mit Parallelität**
  - Ada
  - Java, Real-Time Java

EZ-Programmiersprachen

- **Programmierbeispiele in dieser Vorlesung**
  - C/Real Time POSIX
    - POSIX: Familie von standardisierten Schnittstellen für Betriebssystem-Aufrufe (APIs)
      - IEEE Standard 1003.1-1988 bzw. ISO/IEC 9945
      - Speziell: Echtzeiterweiterungen (POSIX.1b)
    - Betriebssysteme können mehr oder weniger „POSIX-compliant“ sein
    - basiert auf UNIX/Linux, es gibt Umgebungen für Windows
  - Ada
    - Entwickelt vom US DoD 1977 – 1983
    - Sollte mehrere 100 Programmiersprachen ersetzen
    - Ursprünglich für eingebettete und Echtzeitsysteme
    - Seit 1995 allgemeine Programmiersprache
      - Systemprogrammierung, Numerik, Objektorientierung
    - Aktuell ADA 2005 (ISO/ANSI-Standard)
  - Java mit Echtzeit-Erweiterungen
    - *Real-Time Specification for Java (RTSJ)*
      - Möglichkeiten, z.B. Garbage Collection zu unterbinden

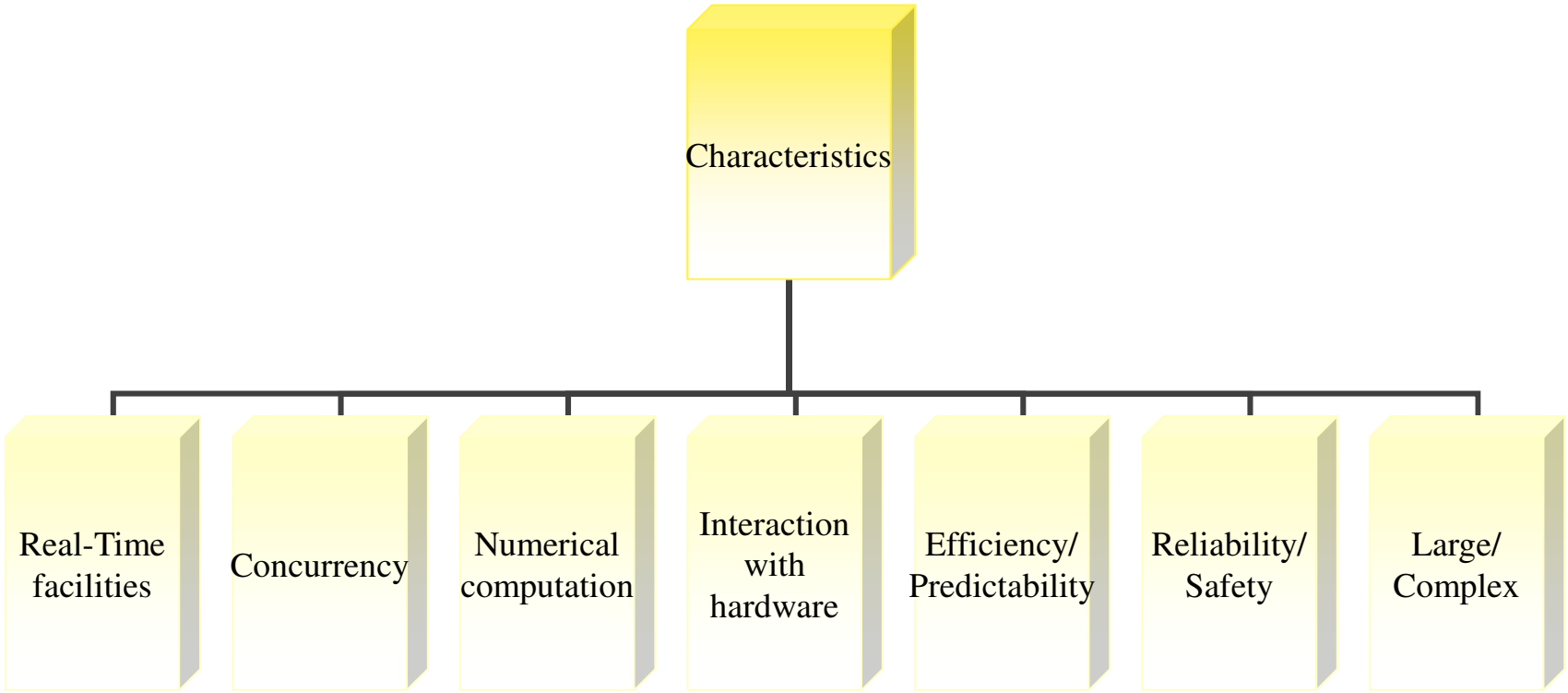
Zusammenfassung

- Aspekte von Echtzeitsystemen [Burns & Wellings 2009, Abb.1.8]



Zusammenfassung

- Aspekte von Echtzeitsystemen [Burns & Wellings 2009, Abb.1.8]



Literatur

[Burns & Wellings 2009] Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages. Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, 2009.

[Wörn & Brinkschulte 2005] Heinz Wörn, Uwe Brinkschulte: *Echtzeitsysteme*. Springer, 2005.

[Zöbel 2008] Dieter Zöbel: *Echtzeitsysteme. Grundlagen der Planung*. Springer, 2008.