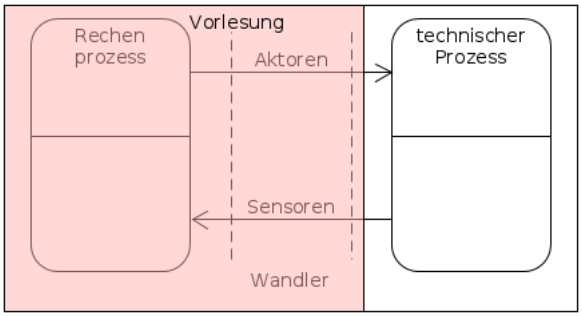
# 1 placeholder

# 2 Zentrale Beschreibgrößen

## 2.1 placeholder

**Defintion:** Realzeitsystem haben neben Funktionalen Anforderungen auch zeitliche Anforderungen.

Ein Realzeitsystem besteht softwaretechnisch aus einer Reihe von Tasks und aus der System-Software.

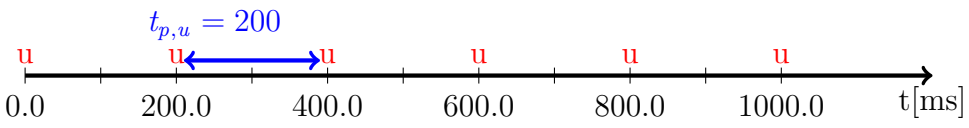



### 2.1.1 Technischer Prozess

Rechenzeitanforderung = Ereignis von technischen Prozess Releasetime = Zeitpunkt des Auftretens der RZ-Anforderung (RZ/RT = Realzeit)

**Beispiel:** periodisches Signal u alle 200ms

$t_{Release,u,1} = 0ms$ $t_{Release,u,2} = 200ms$
$t_{Release,u,3} = 400ms$ $t_{Release,u,4} = 600ms$




Prozesszeit = zeitlicher Abstand zwischen zwei RZ-Anforderungen gleichen Typs.

$$t_{Pmin,i} = minimal => t_{max,i} = \frac{1}{t_{Pmin,i}}$$
$$t_{Pmax,i} = maximal <= \text{uninteressant}$$

$t_{Dmin,i}$ = minimal zulässige Reaktionszeit
$t_{Dmax,i}$ = maximal zulässige Reaktionszeit

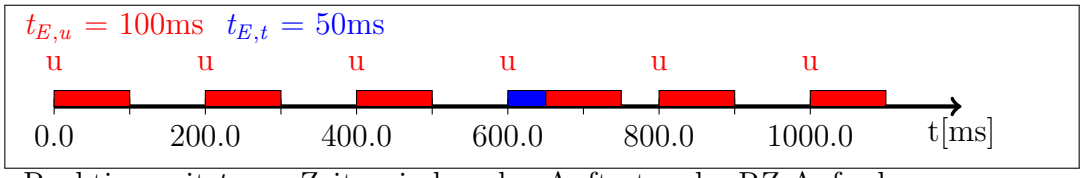
Airbag:
$t_{Dmax}$ = 50ms(Zeit bis zum Aufschlag) - 30ms(Zeit zum aufblasen) = 20ms
$t_{Dmin}$ = 0ms
Phase = minimal Zeitlicher Abstand zwischen zwei unterschiedlicher RZ-Anforderungen $t_{Ph,i,j}$

### 2.1.2 Rechenprozesse

- Ausführuntgszeit (Executiontime) = Rechenzeit für eine RZ-Anforderung (ohne Warte oder Schlafzeiten)

  - WCET $t_{Emax,i}$ –> Erfahrung oder Messen Worstcase

  - BCET $t_{Emin,i}$ = 0 Bestcase




- Reaktionszeit $t_{R,i}$ = Zeit zwischen den Auftreten der RZ-Anforderungen i und dem Ende der Bearbeitung.

$T_{Rmax,i}$ = maximale Reaktionszeit
$T_{Rmin,i}$ = minimale Reaktionszeit
$T_{R,i} = t_{W,i} + t_{E,i}$ wobei $t_{W,i}$ Summe aller Wartezeiten

### 2.1.3 Systemsoftware

- Latenzzeit $t_{L_i}$ = Zeit zwischen dem Auftreten einer RZ-Anforderung und dem Start der Bearbeitung - Interrup Latenzzeit - Tasklatenzzeit

## 2.2 Realzeitbedingungen

### 2.2.1 Auslastungsbedingung

$\rho_i = \frac{t_{E,i}}{t_{P,i}}$ Auslastung dur RZ-Anforderung i

$\rho_{max,i} = \frac{t_{Emax,i}}{t_{Pin,i}}$ Worstcase, max. Auslastung

1. RT Bedingung

$$\rho_{max,ges} = \sum_{j=1}^{n} \frac{t_{Emax,i}}{t_{Pin,i}} \leq c$$

j = für alle RZ-Anforderungen, c = Anzahl der Rechnerkerne

Beispiel: 2 RZ-Anforderungen A und B

$$\left.\begin{aligned} t_{Pmin,A} &= 2ms \\ t_{Emax,A} &= 0.8ms \end{aligned}\right\} \rho_{max,A} = \frac{0.8ms}{2ms} = 0.4ms$$

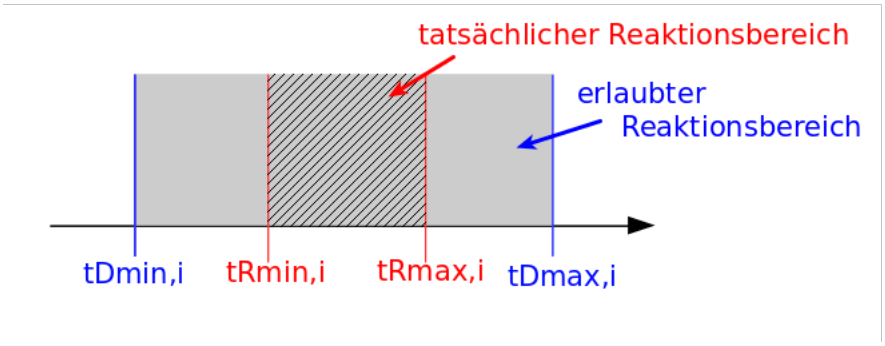
$$\left.\begin{aligned} t_{Pmin,B} &= 1ms \\ t_{Emax,B} &= 0.3ms \end{aligned}\right\} \rho_{max,B} = \frac{0.3ms}{1ms} = 0.3ms$$

$\rho_{max,ges} = \rho_{max,A} + \rho_{max,B} = 0.7 = 70\%$

Annahme Singlecore c = 1 $\rho_{max,ges} \leq c$ => $0.7 \leq 1$
Auslastungsbedingung erfüllt

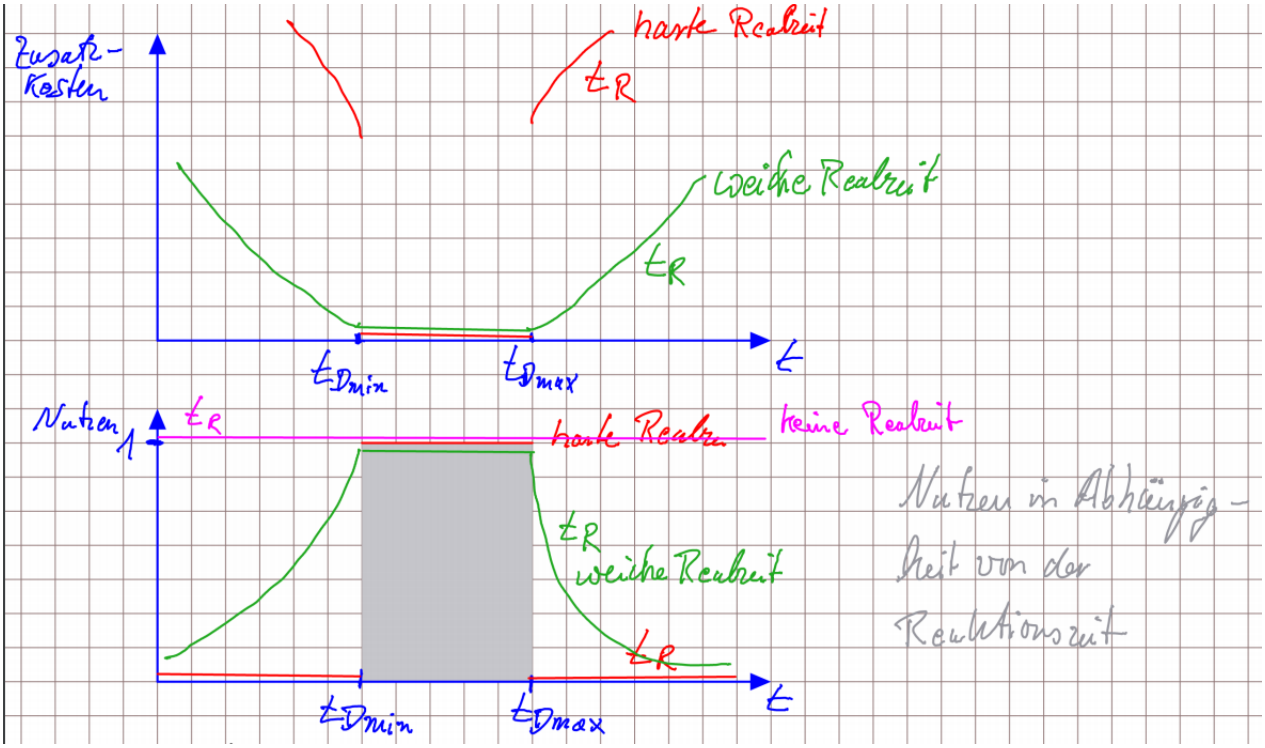
### 2.2.2 Rechtzeitigkeitsbedingung

Für den Realzeitbetrieb muss die tatsächliche Reaktion innerhalb des Zulässigen Reaktionsbereiches erfolt sein.




2. RT Bedingung
Für alle RZ-Anforderungen j muss gelten:
$t_{Dmin,j} \leq t_{Rmin,j} \leq t_{Rmax,j} \leq t_{Dmax,j}$

### 2.2.3 Harte und weiche Realzeit



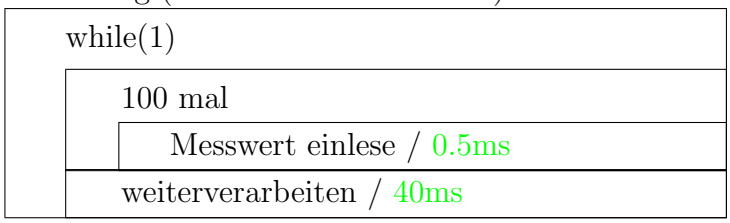

## 2.3 Systemaspekte

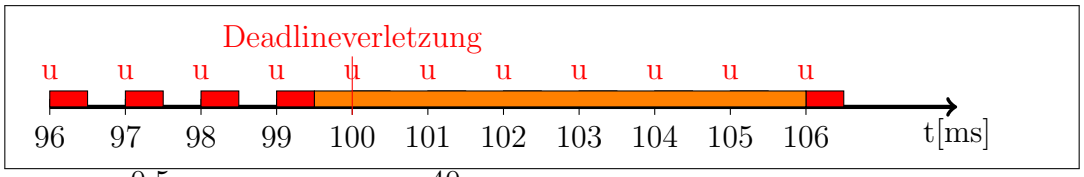
### 2.3.1 Unterbrechbarkeit

**Forderung:** Codesequenzen lassen sich in Teilsequenzen unterteilen, die in Korrekter Reihenfolge aber unabhängig voneinander abgearbeitete werden können.
=> notwendig für den Realzeitbetrieb

**Begründung:** Ein Messwert soll kontinuierlich erfasst werden.
$t_{Emin,u} = t_{Emax,E} = 0.5ms$
Jeweils 100 Messwerte (alle 100ms) sollen weiterverarbeitet werden
$t_{Pmin,w} = 100ms$
$t_{Dmin,w} = 0ms$
$t_{Dmax,w} = 100ms$
$t_{Dmax,w} = 100ms$
$t_{Emin,w} = t_{Emax,w} = 40ms$

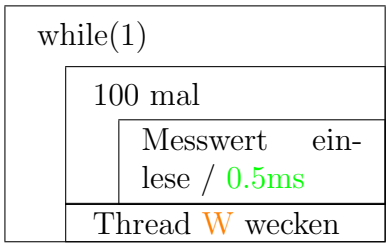
Lösung (ohne Unterbrechbarkeit):



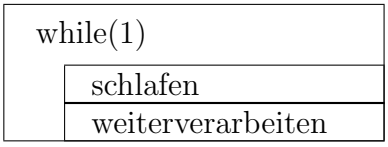




$\rho_{max,u} = \frac{0.5ms}{1ms} = 0.5$; $\rho_{max} = \frac{40ms}{100ms} = 0.4 => \rho_{max,ges} = 0.9$

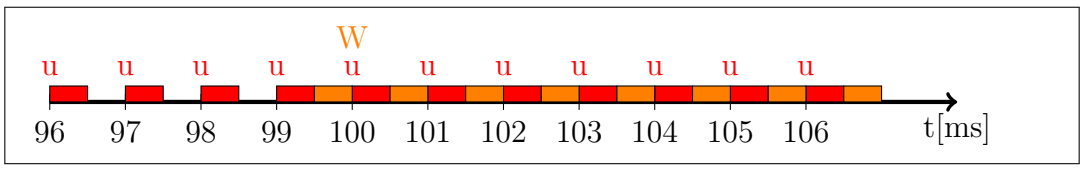
Lösung mit Unterbrechbarkeit:




Thread U




Thread W




**Konsequenzen:**

a) Inter-prozess-Kommunikation (IPC) (Sync, Datenaustausch)

b) Multithreading/Multitasking

### 2.3.2 Prioritäten

**Forderung:** Der Systemarchitekt muss einfluss auf die Abarbeitungsreihenfolge mehrerer Tasks nehmen können z.B. über Prioritäten.

### 2.3.3 Ressourcenmanagment

–> später

# 3 Systemsoftware

## 3.1 Firmware

**Aufgabe:**

- Basisinitialisierung der Hardware
- Diagnose
- Betriebinitialisierung
- Laden + Aktivieren von Codes
- Runtime Services

**Ausprägungen:**

- BIOS
- UEFI
- Bootloader ("Das U-Boot")
- Monitor Software

## 3.2 RT-OS

**Defintion.:** Bezeichnung für alle Software-Komponenten, die

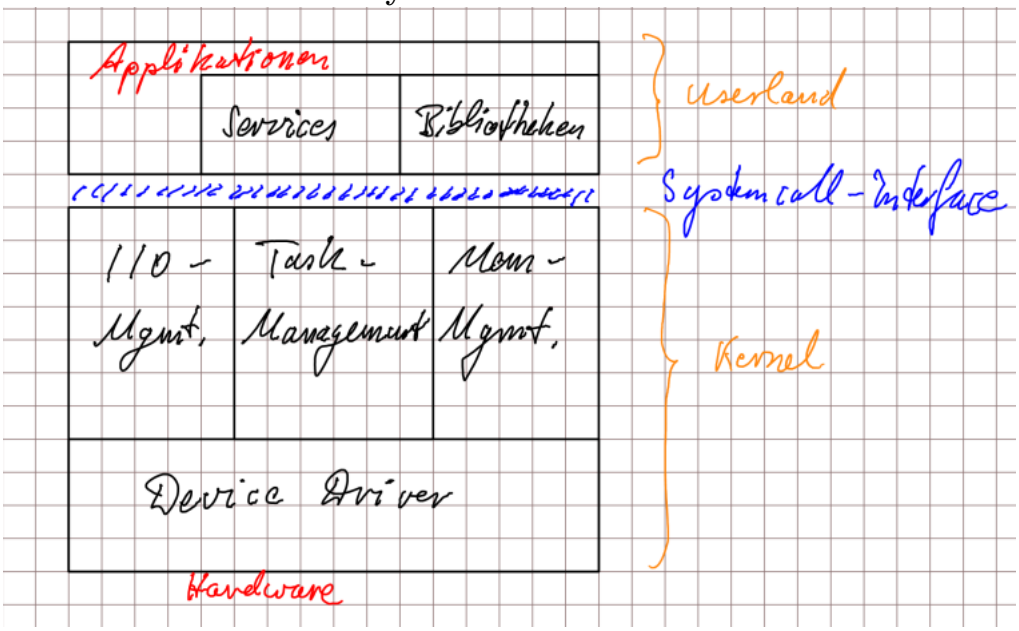
- die Ausführung der Applikationen und
- die Verteilung der Betriebsmittel (Memory, Files, CPU, Drucker, ...) ermöglichen, steuern und überwachen.

**Anforderungen:**

- Zeitverhalten
- Ressourcenverbraucht
- Zuverlässigkeit und Stabilität
- Sicherheit
- Flexibilität und Kompatibilität
- Portierbarkeit
- Skalierbarkeit

**Beispiele:** Sämtliche Betriebsysteme

**Systemarchitektur**




### 3.2.1 Systemcall-Interface

Systemcall = Dienst des Kernels –> 300-400 Dienste

**Beispiele:** open(), close(), read(), write(), exit(), fork(), clone(), clock_nanosleep(), kill(), adjtime(),...
Technische Realisierung: SW-Interrupt

**Ablauf:** ret = write(fd, "Hello World", 13);
↓ Systemcall "write" per SW-Interrupt
"int 0x80", "trap", "sysenter"
Systemcall mit EAR = 4 <– x86 Register

ISR (SW-Interrupt 0x80)
↓ EAX = 4 –> bedeutet write
vfs_write()
↓ wertet die übrigen CPU-Register aus
↓ fd –> entscheidet über den zu nutzenden Gerätetreiber
driver_write() –> gibt Hardwaretehcnisch die Daten aus

### 3.2.2 Taskmanagment

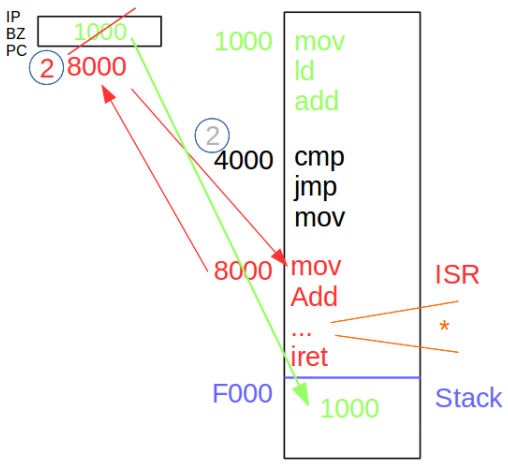
**Aufgabe:** Verwaltung der Ressource CPU

–> quasi parallele Verarbeitung auf einzelenen CPU-Kernen

–> real parallele Verarbeitung auf Multicore-Rechnern

Scheduling = Auswahl des als nächsten zu bearbeiten Jobs
Content Switch = Aktivierung eines Jobs

**Singelcore-Scheduling** Realisierung: Modifikation der Rücksprungadresse auf dem Stack beim Interrupt.




1. Code an der im IP stehenen Address wird abgearbeitete
2. IR tritt auf
   - Inhalt vom IP wird auf den Stack gelegt
   - IP wird auf die Addresse der ISR gelegt (CPU arbeitet die ISR ab)
3. Bei iret wird die auf dem stack hinterlegte Addresse zurück auf den IP geladen

–> normale Verarbeitung wird fortgesetzt

* zusätlicher Code der die auf dem Stack liegende Adresse ändert
z.B. die 1000 wird mit 4000 überschrieben