

# Contents

<b>1</b>	<b>placeholder</b>	<b>1</b>
<b>2</b>	<b>Zentrale Beschreibgrößen</b>	<b>1</b>
2.1	placeholder . . . . .	1
2.1.1	Technischer Prozess . . . . .	1
2.1.2	Rechenprozesse . . . . .	2
2.1.3	Systemsoftware . . . . .	3
2.2	Realzeitbedingungen . . . . .	3
2.2.1	Auslastungsbedingung . . . . .	3
2.2.2	Rechtzeitigkeitsbedingung . . . . .	4
2.2.3	Harte und weiche Realzeit . . . . .	4
2.3	Systemaspekte . . . . .	5
2.3.1	Unterbrechbarkeit . . . . .	5
2.3.2	Prioritäten . . . . .	6
2.3.3	Ressourcenmanagment . . . . .	6
<b>3</b>	<b>Systemsoftware</b>	<b>6</b>
3.1	Firmware . . . . .	6
3.2	RT-OS . . . . .	7
3.2.1	Systemcall-Interface . . . . .	8
3.2.2	Taskmanagment . . . . .	8
3.2.3	Scheduling . . . . .	12
3.2.4	Singlecore Scheduling . . . . .	13

## 1 placeholder

## 2 Zentrale Beschreibgrößen

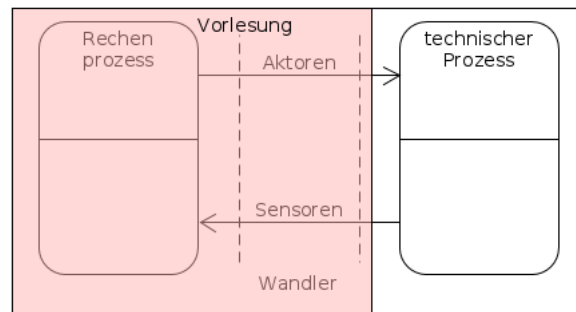
### 2.1 placeholder

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

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

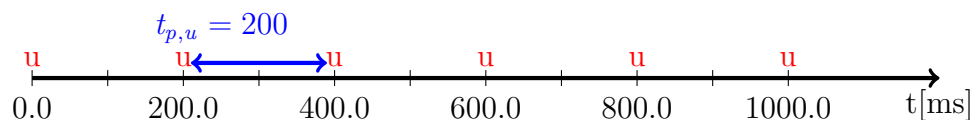
#### 2.1.1 Technischer Prozess

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.

$$t_{Pmin,i} = \text{minimal} \Rightarrow t_{max,i} = \frac{1}{t_{Pmin,i}}$$

$$t_{Pmax,i} = \text{maximal} \leq \text{uninteressant}$$

$$t_{Dmin,i} = \text{minimal zulässige Reaktionszeit}$$

$$t_{Dmax,i} = \text{maximal zulässige Reaktionszeit}$$

Airbag:

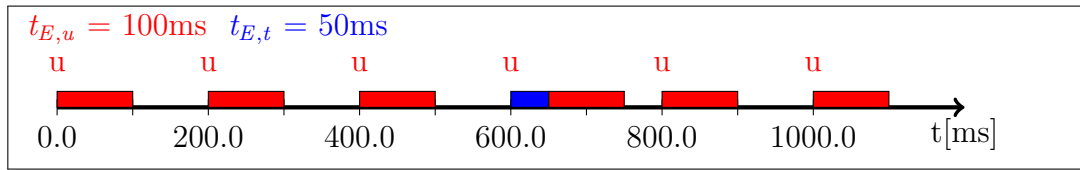
$$t_{Dmax} = 50\text{ms}(\text{Zeit bis zum Aufschlag}) - 30\text{ms}(\text{Zeit zum aufblasen}) = 20\text{ms}$$

$$t_{Dmin} = 0\text{ms}$$

Phase = minimal Zeitlicher Abstand zwischen zwei **unterschiedlicher** RZ-Anforderungen  $t_{Ph,i,j}$

## 2.1.2 Rechenprozesse

- Ausführungszeit (Executiontime) = Rechenzeit für eine RZ-Anforderung (ohne Warte oder Schlafzeiten)
  - WCET  $t_{Emax,i} \rightarrow$  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 - **Interrupt Latenzzeit** - **Tasklatenzzeit**

## 2.2 Realzeitbedingungen

### 2.2.1 Auslastungsbedingung

$\rho_i = \frac{t_{E,i}}{t_{P,i}}$  Auslastung der 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

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

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

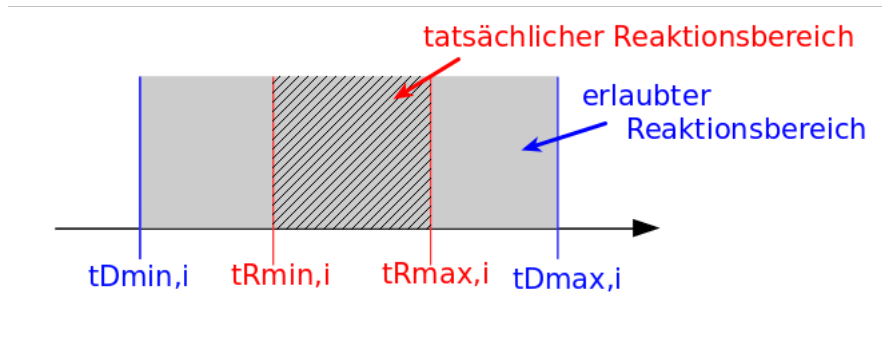
$$\rho_{max,ges} = \rho_{max,A} + \rho_{max,B} = 0.7 = 70\%$$

Annahme Singlecore  $c = 1$   $\rho_{max,ges} \leq c \Rightarrow 0.7 \leq 1$

Auslastungsbedingung erfüllt

### 2.2.2 Rechtzeitigkeitsbedingung

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

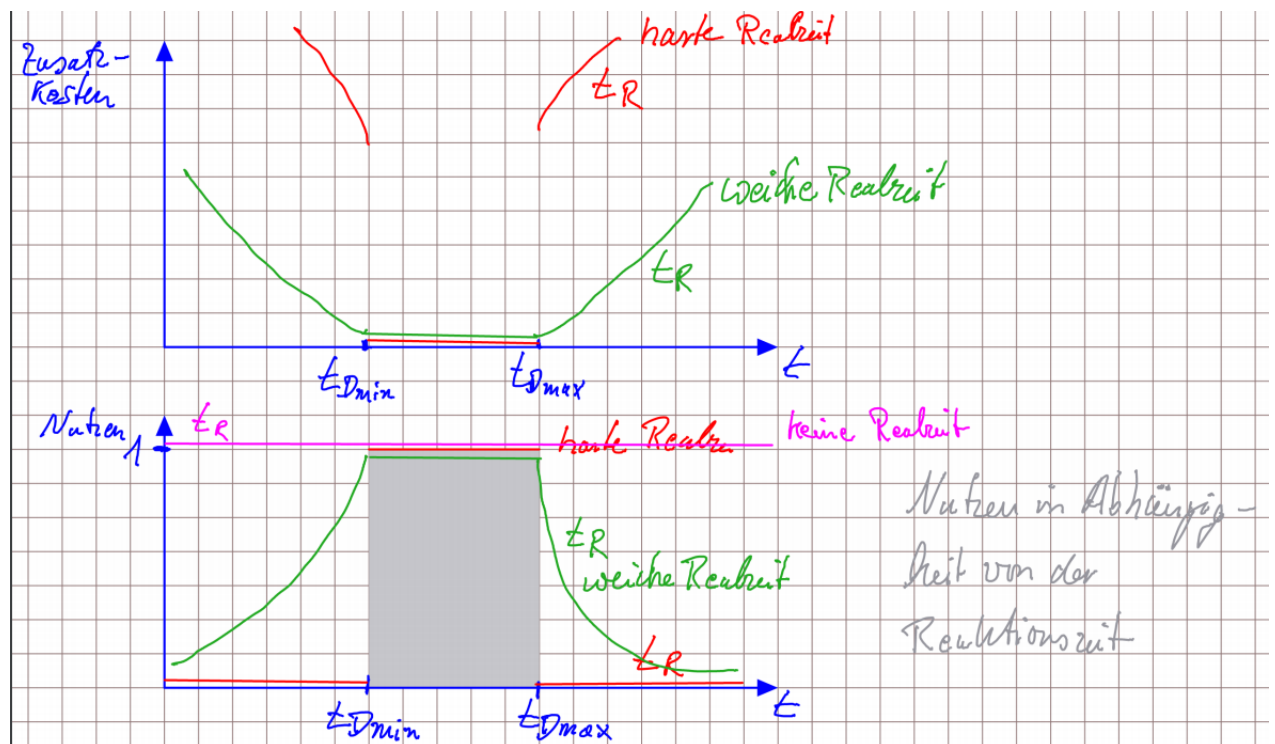


## 2. RT Bedingung

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 abgearbeitet 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

$$t_{Pmin,w} = 100ms$$

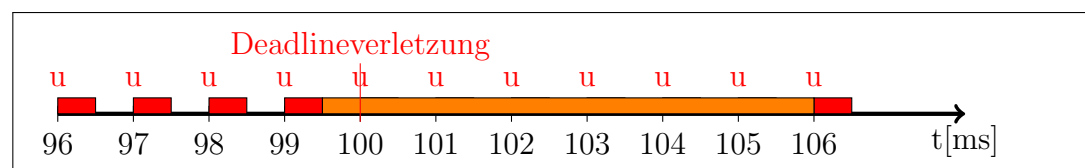
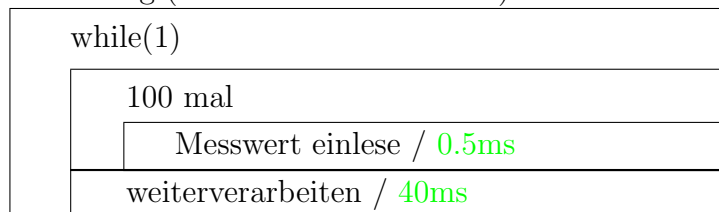
$$t_{Dmin,w} = 0ms$$

$$t_{Dmax,w} = 100ms$$

$$t_{Dmax,w} = 100ms$$

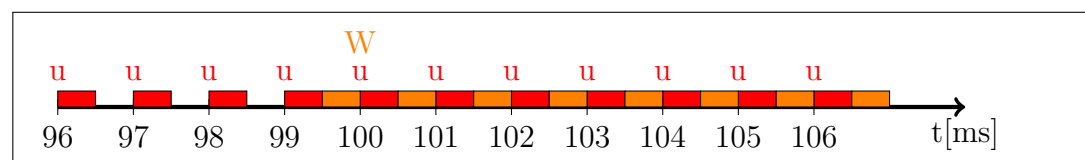
$$t_{Emin,w} = t_{Emax,w} = 40ms$$

Lösung (ohne Unterbrechbarkeit):

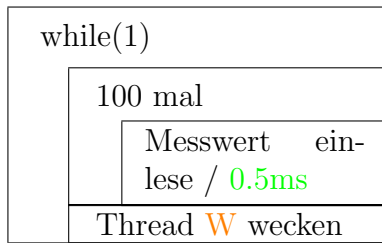


$$\rho_{max,u} = \frac{0.5ms}{1ms} = 0.5; \rho_{max,w} = \frac{40ms}{100ms} = 0.4 \Rightarrow \rho_{max,ges} = 0.9$$

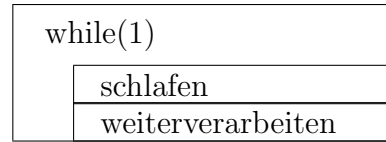
Lösung mit Unterbrechbarkeit:



**Konsequenzen:**



Thread U



Thread W

- 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

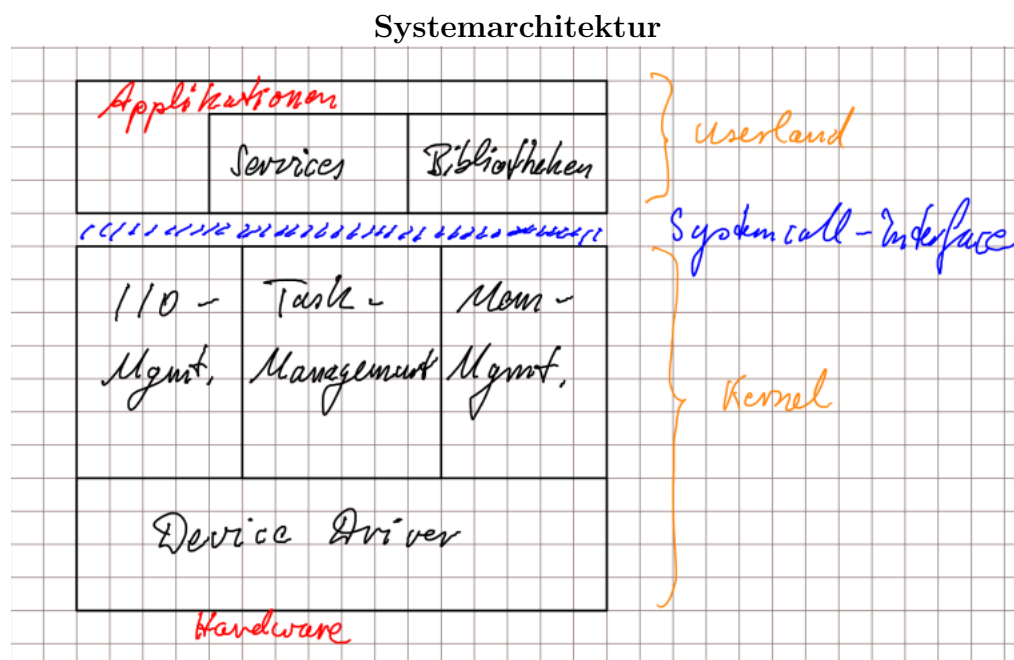
**Defintion.:** Bezeichnung für alle Software-Komponenten, die

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

### Anforderungen:

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

**Beispiele:** Sämtliche Betriebssysteme



### 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 `EAX = 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 Hardwaretechnisch die Daten aus

### 3.2.2 Taskmanagment

**Aufgabe:** Verwaltung der Ressource CPU

-> quasi parallele Verarbeitung auf einzelnen CPU-Kernen

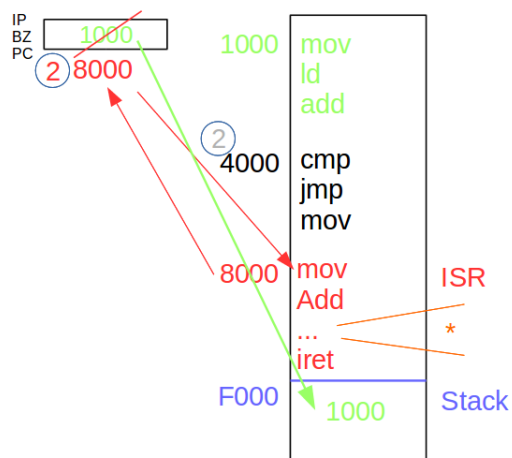
-> real parallele Verarbeitung auf Multicore-Rechnern

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 stehenden Address wird abgearbeitete

2. IR tritt auf

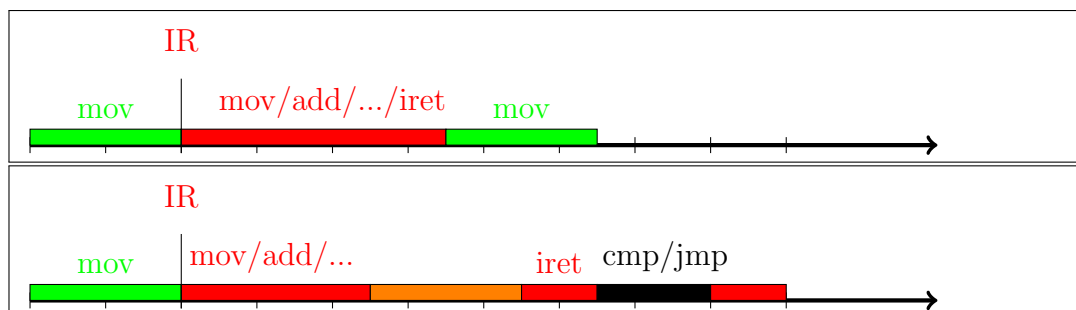
- Inhalt vom IP wird auf den Stack gelegt

- IP wird auf die Adresse der ISR gelegt (CPU arbeitet die ISR ab)

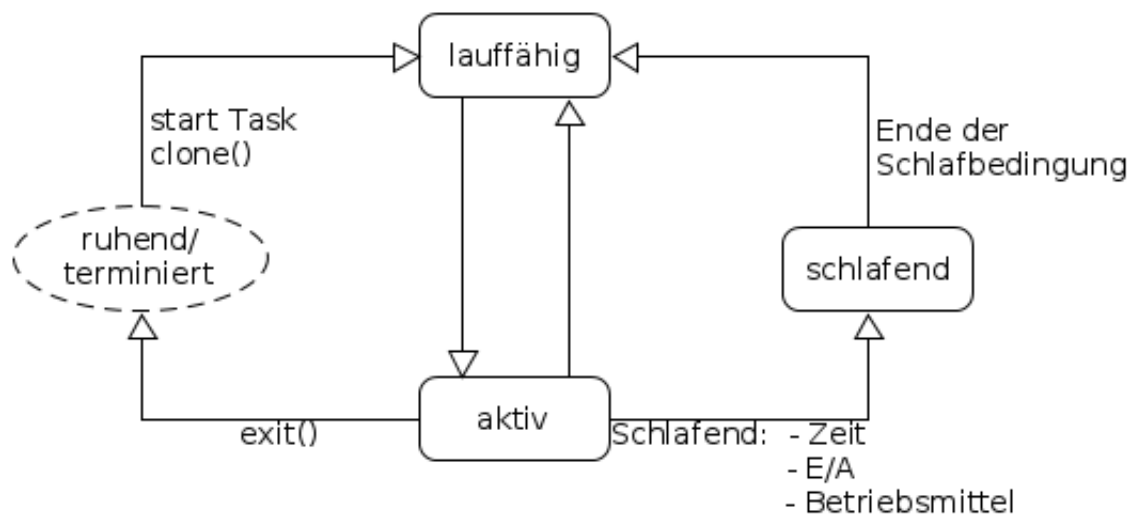
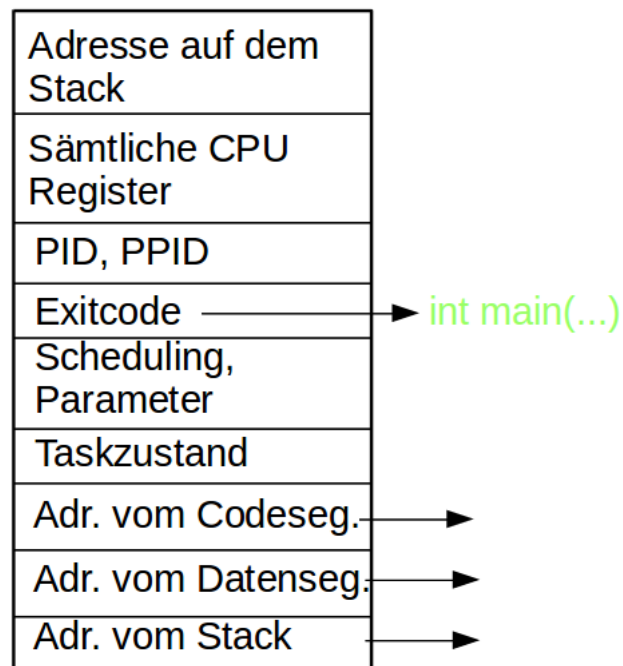
3. Bei ired wird die auf dem stack hinterlegte Adresse zurück auf den IP geladen

-> normale Verarbeitung wird fortgesetzt

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



Eine Datenstruktur wird benötigt, um alle Informationen zu einer Code-sequenzen speichern(Job, Rechenprozess): [Task Controll Block \(TCB\)](#)

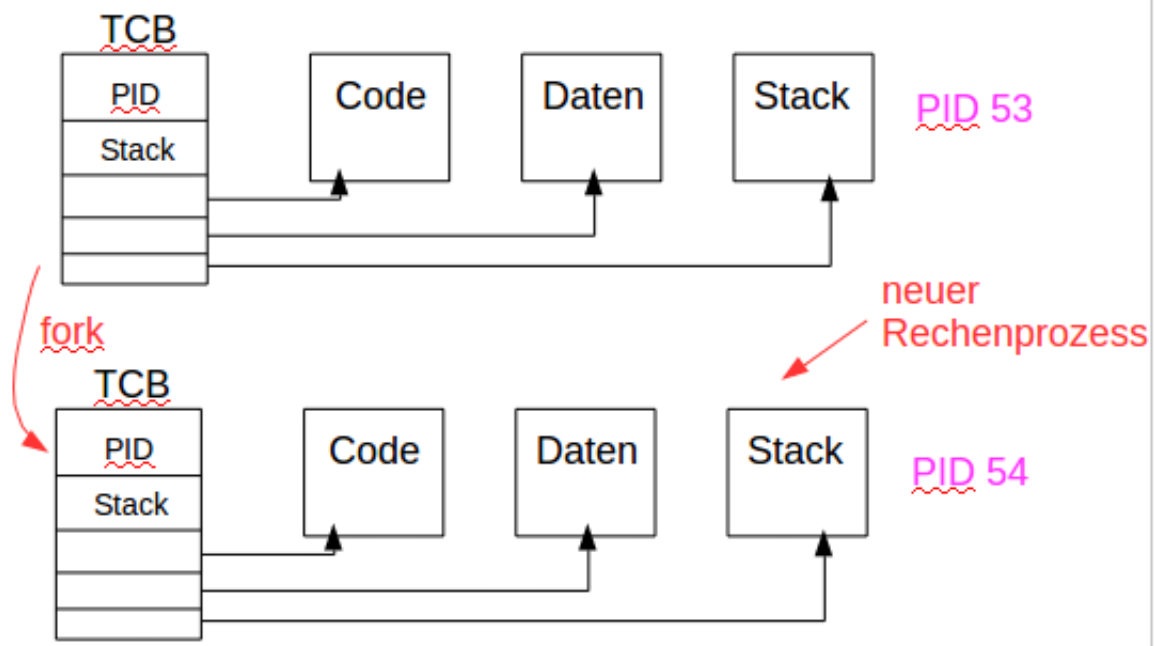


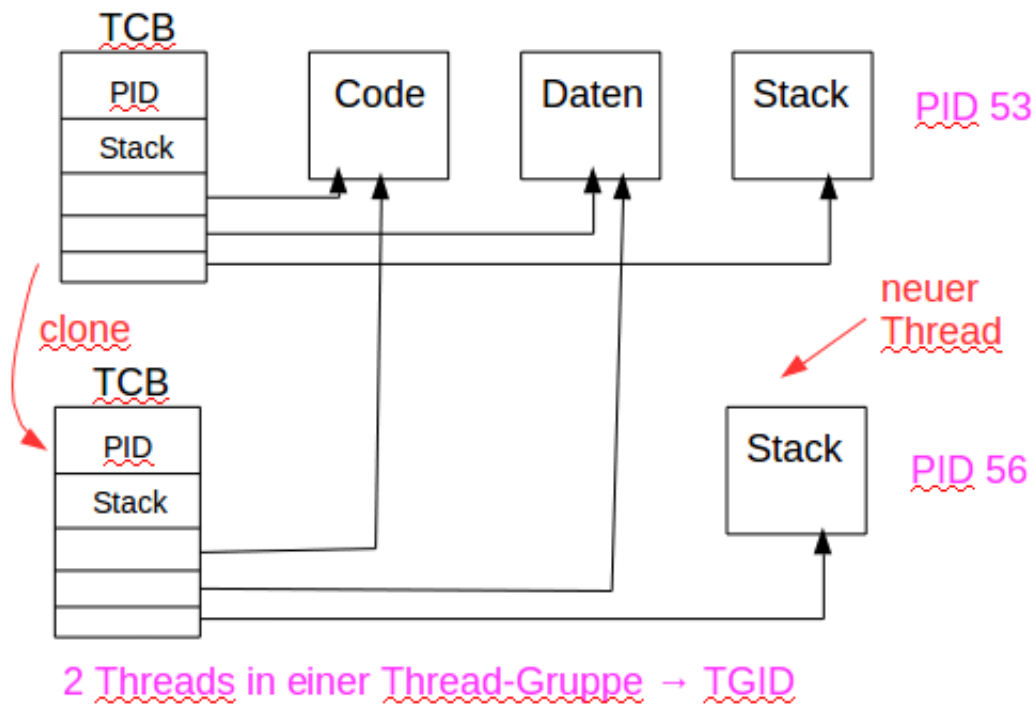
**Erzeugen von Rechenprozessen** Neue Jobs werden durch Kopieren von

- TCB

- Stack-Segment
- Code-Segment
- Daten Segment
- Neue PID vergeben

erzeugt





Threads einer Threadgruppe teilen sich Code- und Datensegment Vorteil:

- Speicherplatzersparnis
- schnell erzeugt
- einfache IPC (Inter Process Communication)

Nachteil:

- Safety: Ein amok laufender Thread bringt die gesamte Threadgruppe in einen inkonsistenten Zustand.

### 3.2.3 Scheduling

**Def.:** Auswahl der Task, die arbeiten/rechnen darf

Content Switch: Aktivierung der ausgewählten Task

Statisches Scheduling: Bedarfsituation ist bekannt

↳ Reihenfolge (Plan) kann im vorhinein festgelegt werden (SPS)

Dynamisches Scheduling: Die Auswahl erfolgt auf Basis der aktuellen Bedarfssituation (z.B. PC, Smartphone)

Singlecore Scheduling: Quasiparallele Verarbeitung auf einem CPU-Kern

Multicore Scheduling: Realparallele Verarbeitung auf mehreren CPU-Kernen

↳ Verteilungsaufgabe Preemption Point: Auftreten einer RF-Anforderung -> Interrup Service Routine(ISR) wird aktiv -> Scheduling

### 3.2.4 Singlecore Scheduling

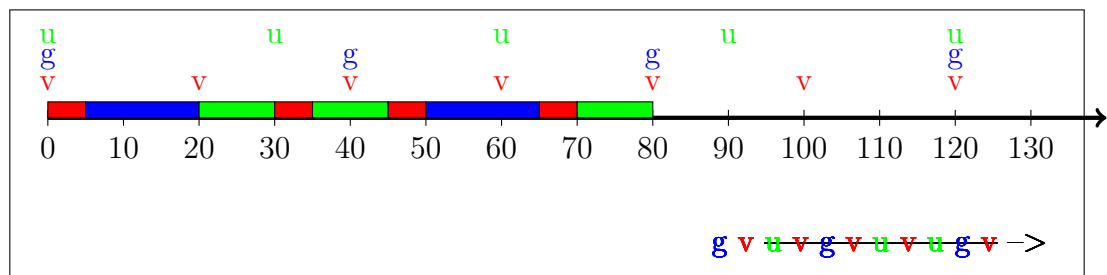
Table 1: Beispieldaten:

RZ-Anf	$t_{Pmin}$	$t_{Dmin}$	$t_{Dmax}$	$t_{Emin}$	$t_{Emax}$
v	20ms	0ms	20ms	1ms	5ms
g	40ms	0ms	40ms	1ms	15ms
u	30ms	0ms	30ms	1ms	10ms

First come First Serve (FCFS, FIFO)

- Lauffähige Jobs werden gemäß Auftrittszeitpunkt in eine Queue eingeführt
- Der erste Job in der Liste wird ausgewählt
- Er darf solange die CPU benutzen (rechnen) bis
  - a) er sich schlafen legt
  - b) er sich beendet

Ein "aufgeweckter" Job wird an den Anfang der Queue gestellt und unterbricht den laufenden Job nicht.

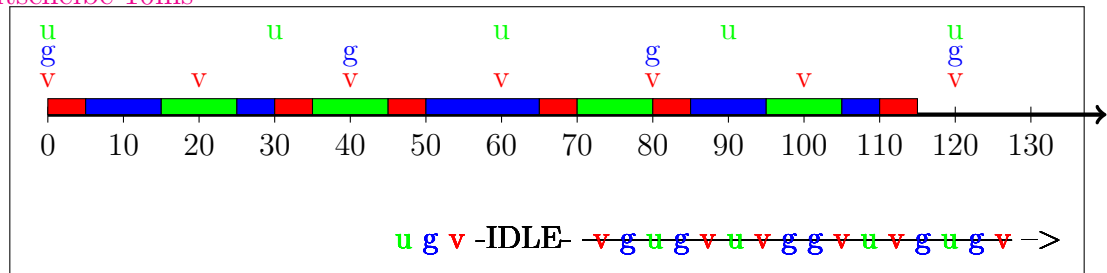


Zeitscheibenverfahren (Round Robin, Rate Monotonic)

- Lauffähige Jobs werden gemäß Auftrittszeitpunkt in eine Queue eingehängt
- Der erste Job in der Liste wird ausgewählt
- Er darf die CPU benutzen bis
  - a) er sich schlafen legt

- b) er sich beendet
- c) seine Zeitscheibe (Quantum) aufgebraucht ist
- Ein aufgeweckter Job wird ans Ende der Queue angehängt

Zeitscheibe 10ms



Prioritätengesteuertes Scheduling

- Jedem Job wird eine Priorität zugewiesen
- Der **Lauffähige** Job mit der höchsten Priorität wird ausgewählt
- Er darf rechnen bis
  - a) er sich schlafen legt
  - b) er sich beendet
  - c) ein Job mit höherer Prio Lauffähig wird

Problem: Verteilung der Prioritäten

kurze  $t_{E_{max}}$  und kurze  $t_{P_{min}}$   $\rightarrow$  hohe Prio

Gibt es zwischen  $t_{E,i}$  und  $t_{P,i}$  keine Korrelation hilft bei der Prioritätenverteilung nur ausprobieren!

**Beispiel:** V = 1 (höchste Prio)

G = 3 (niedrigste Prio)

U = 2 (mittlere Prio)