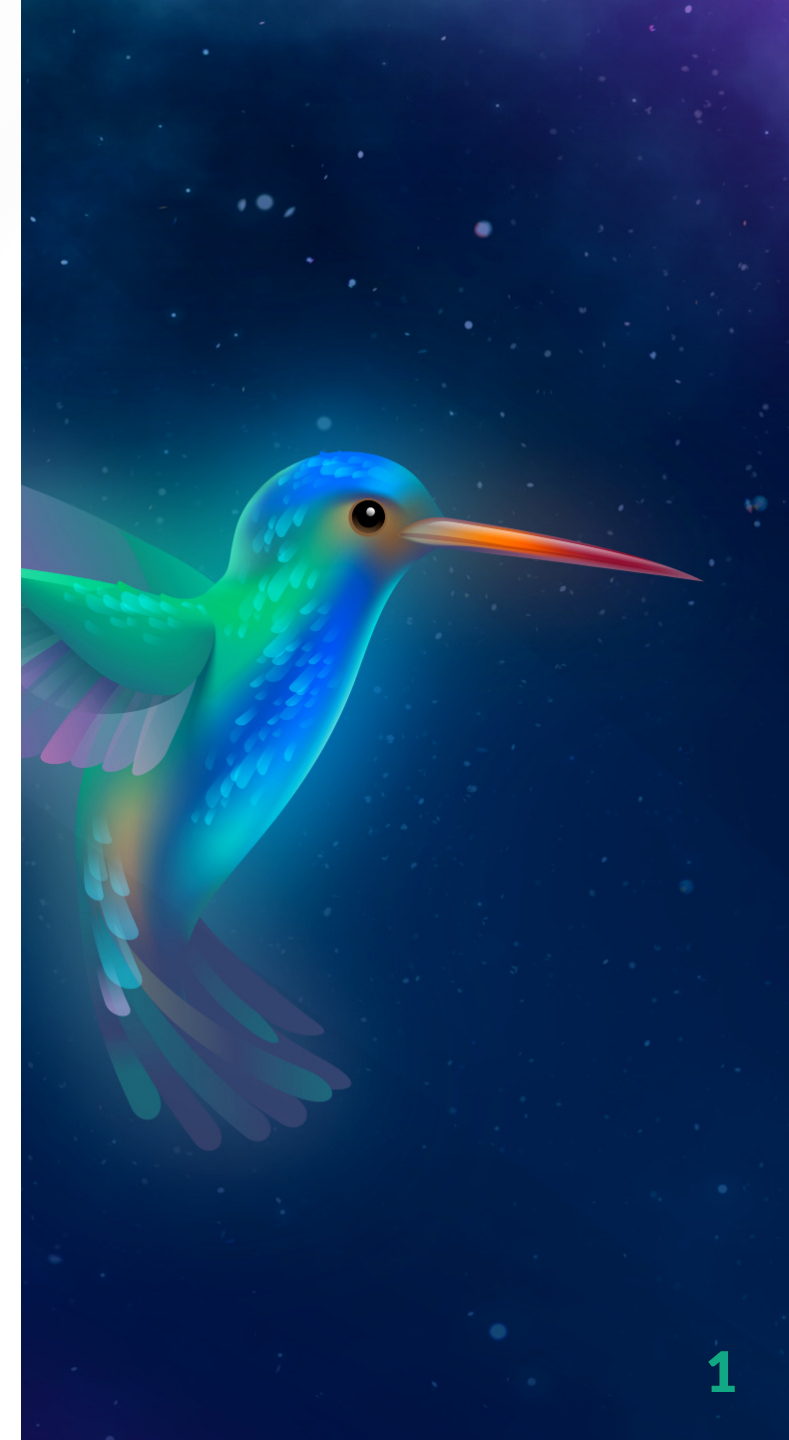
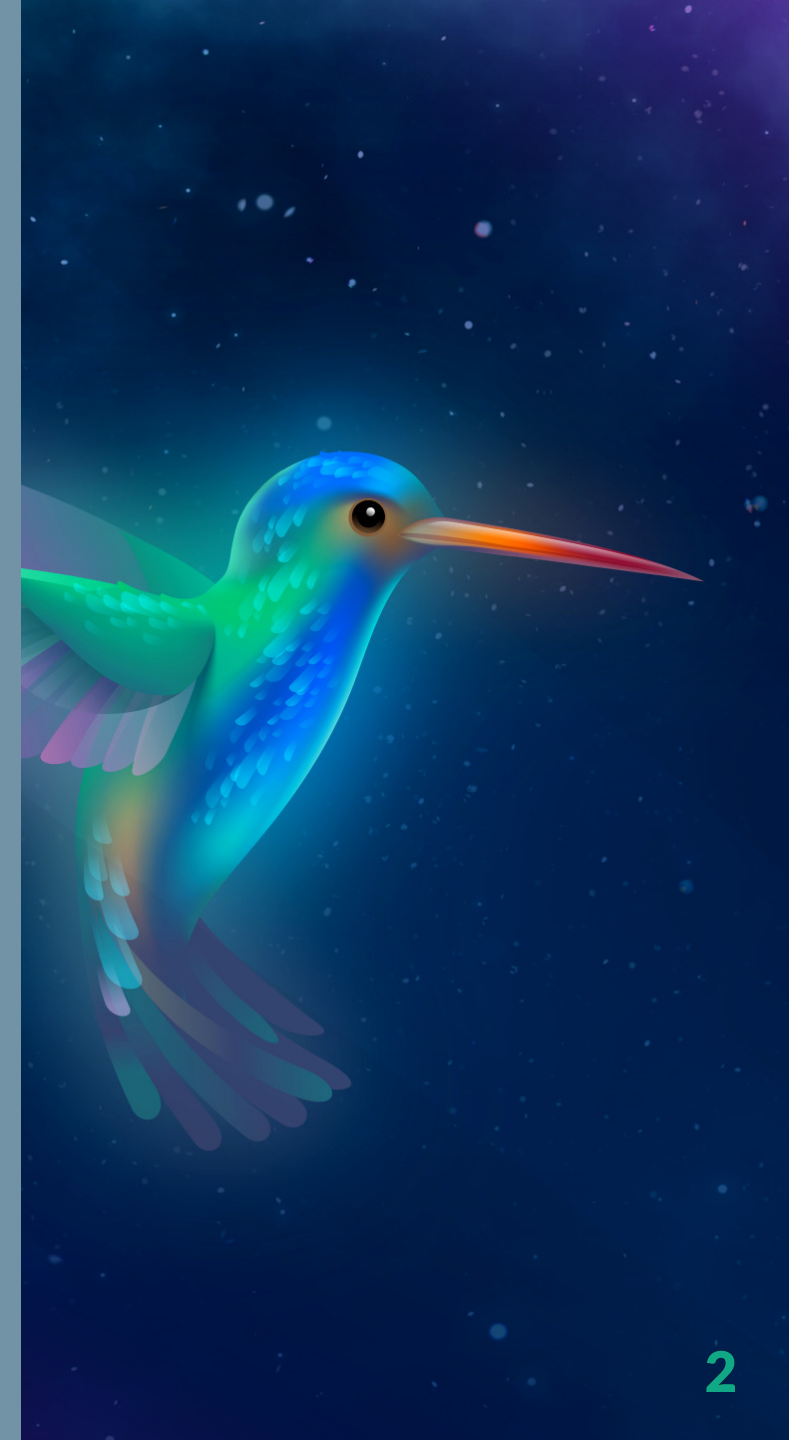


# Tutorat 11

## Mutexe und Semaphore



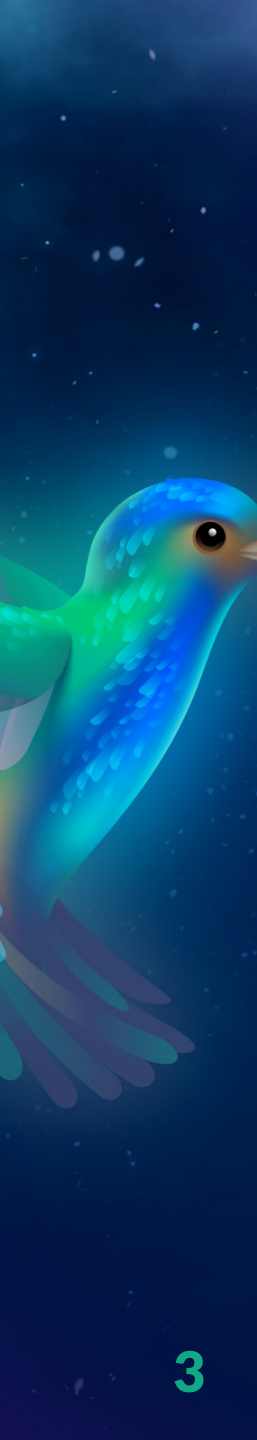
# Organisatorisches



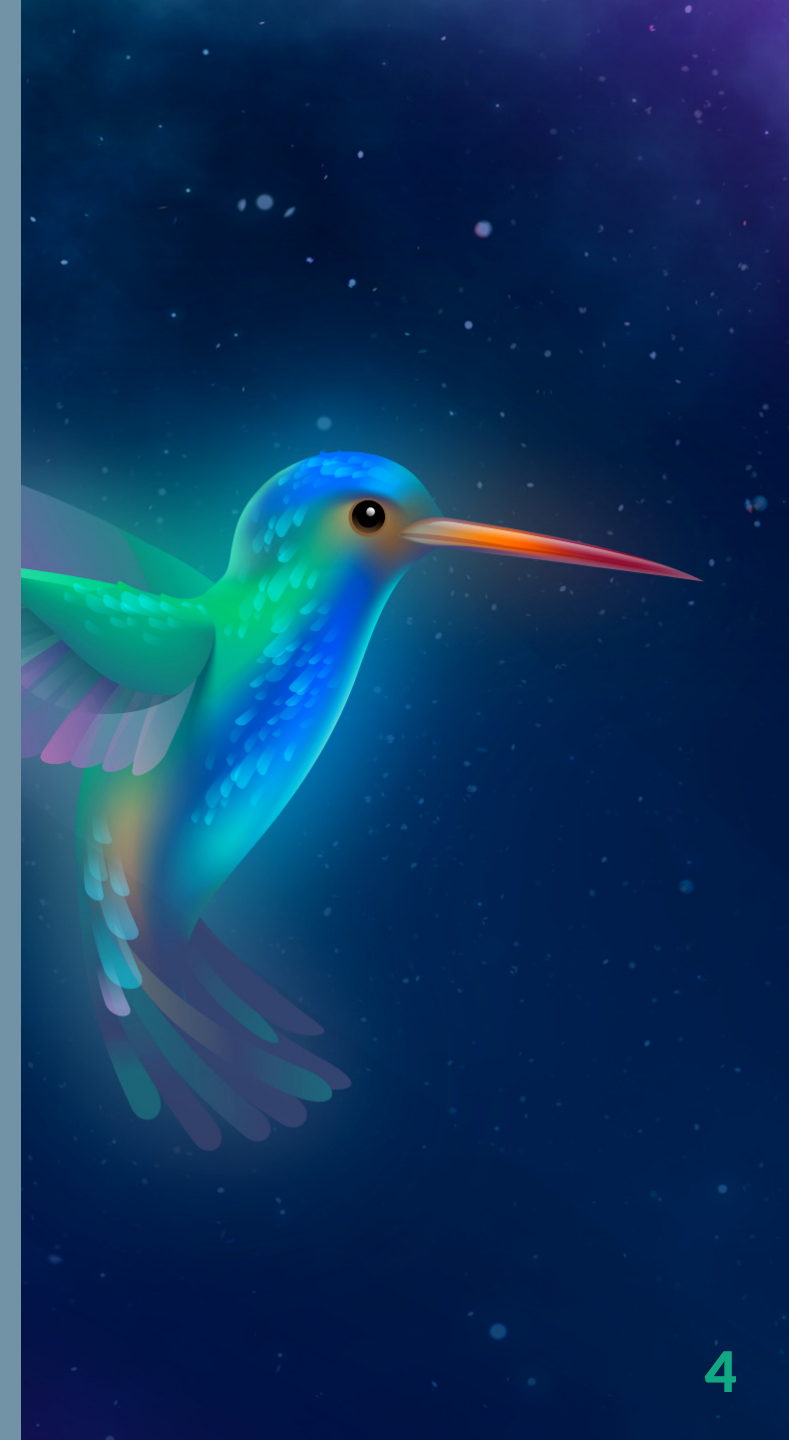
# Organisatorisches

## Verbesserte Lösungen

- `uebungsblatt_5_aufgabe_3.reti` (RETI-Code)
- `uebungsblatt_5_aufgabe_3.csv` (Symboltabelle)
- `uebungsblatt_5_aufgabe_3.ast` (Abstract Syntax → Klammern)
- `uebungsblatt_6_aufgabe_1.reti` (RETI-Code)
- `uebungsblatt_6_aufgabe_1.csv` (Symboltabelle)
- `uebungsblatt_6_aufgabe_1.ast` (Abstract Syntax → Klammern)



# Vorbereitungen

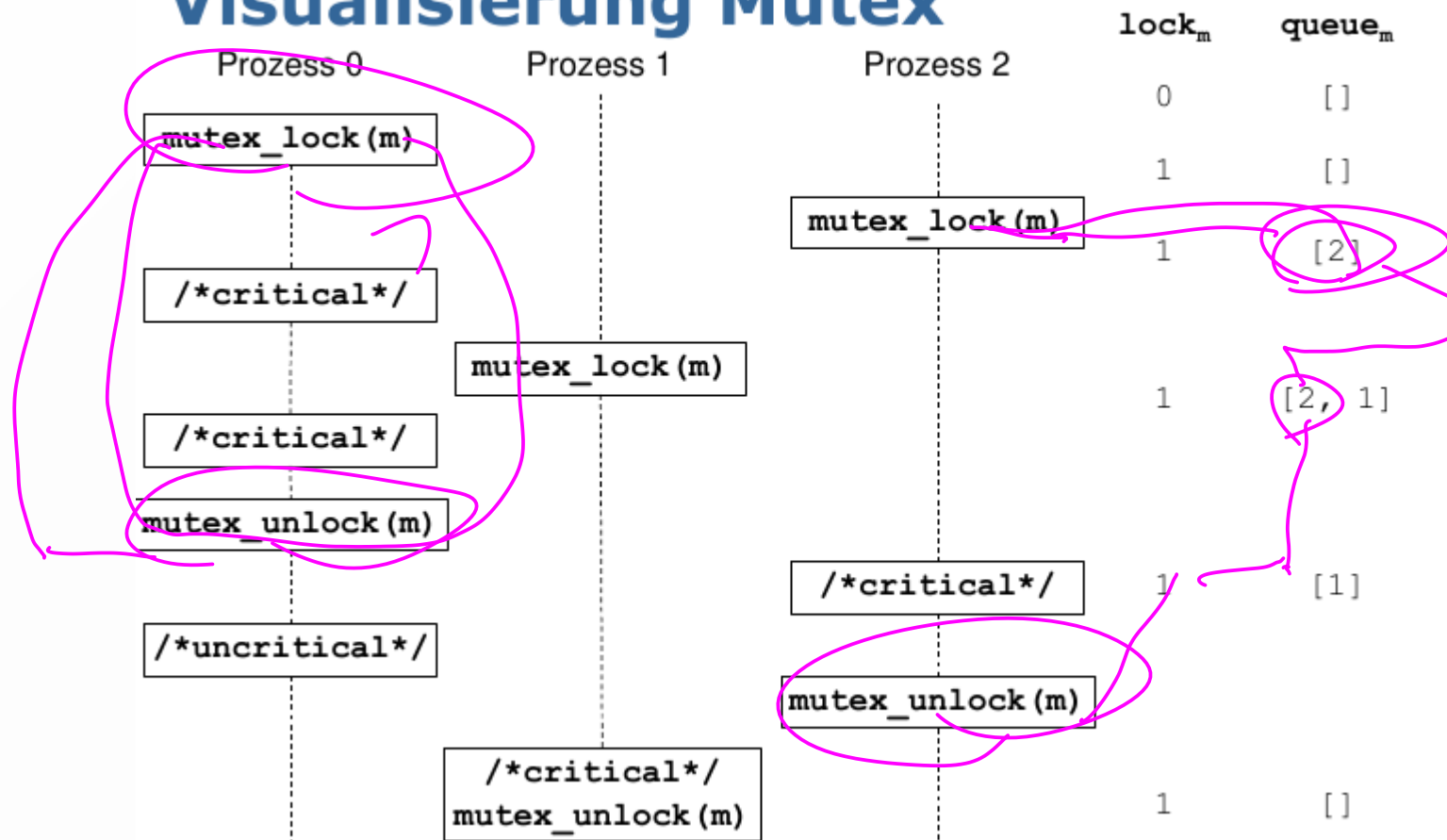


# Vorbereitungen

$S = \text{Semaphor (counter=1)}$

## Mutex

### Visualisierung Mutex



# Vorbereitungen

## Semaphor

### Semaphor: down/up Operationen

- Initialisiere Zähler des Semaphors

- **down-Operation:**

- Verringere den Wert von  $\text{count}_s$  um 1
- Wenn  $\text{count}_s < 0$ , blockiere den aufrufenden Prozess, sonst fahre fort

- **up-Operation:**

- Erhöhe den Wert von  $\text{count}_s$  um 1
- Wenn  $\text{count}_s \leq 0$ , wecke einen der blockierten Prozesse auf

$$\begin{aligned} \text{count} &\leq 0 \\ \text{count} &< 1 \end{aligned} \quad \begin{aligned} &+1 \\ &+1 \end{aligned}$$

$$\text{bzw. } \text{count} \leq 0$$

$$\text{count} = 0$$

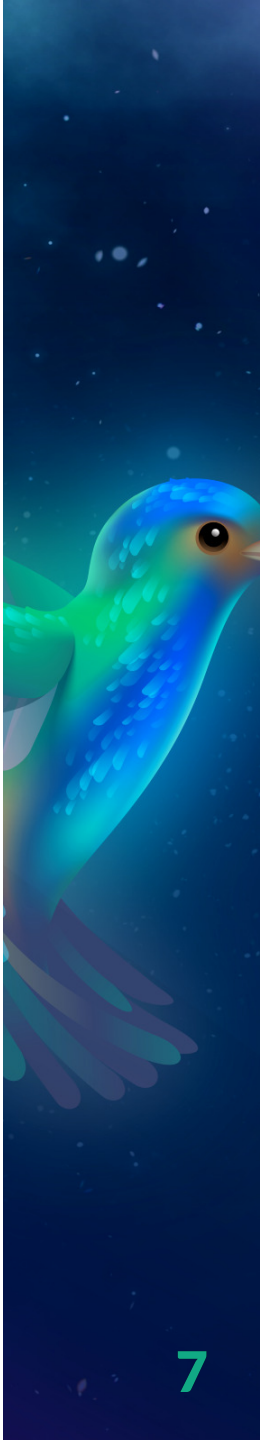
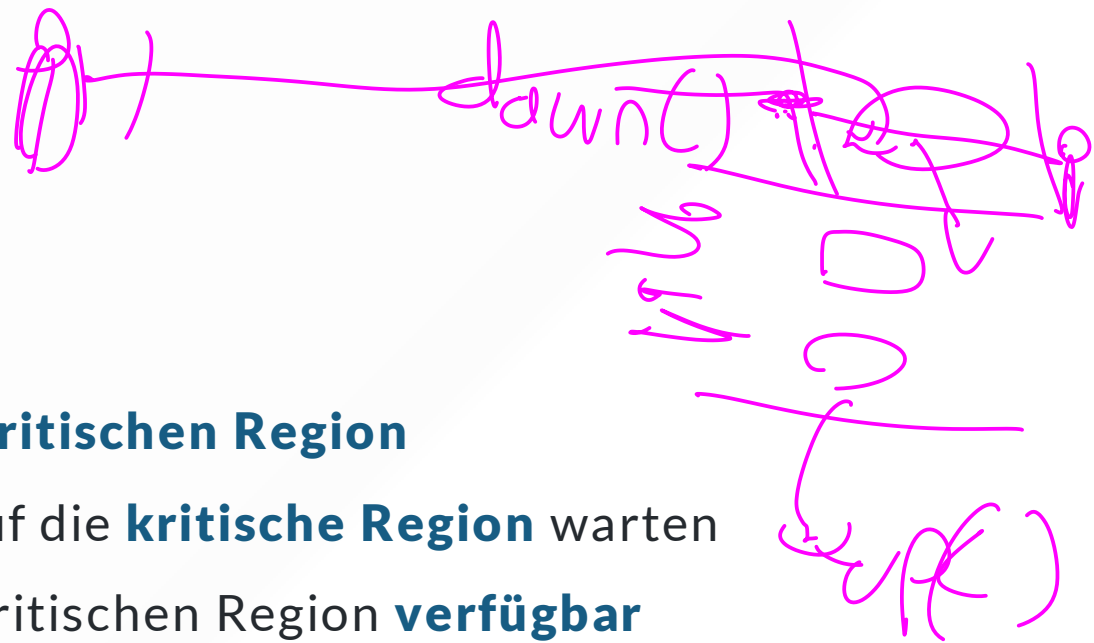
$$\text{up()}$$



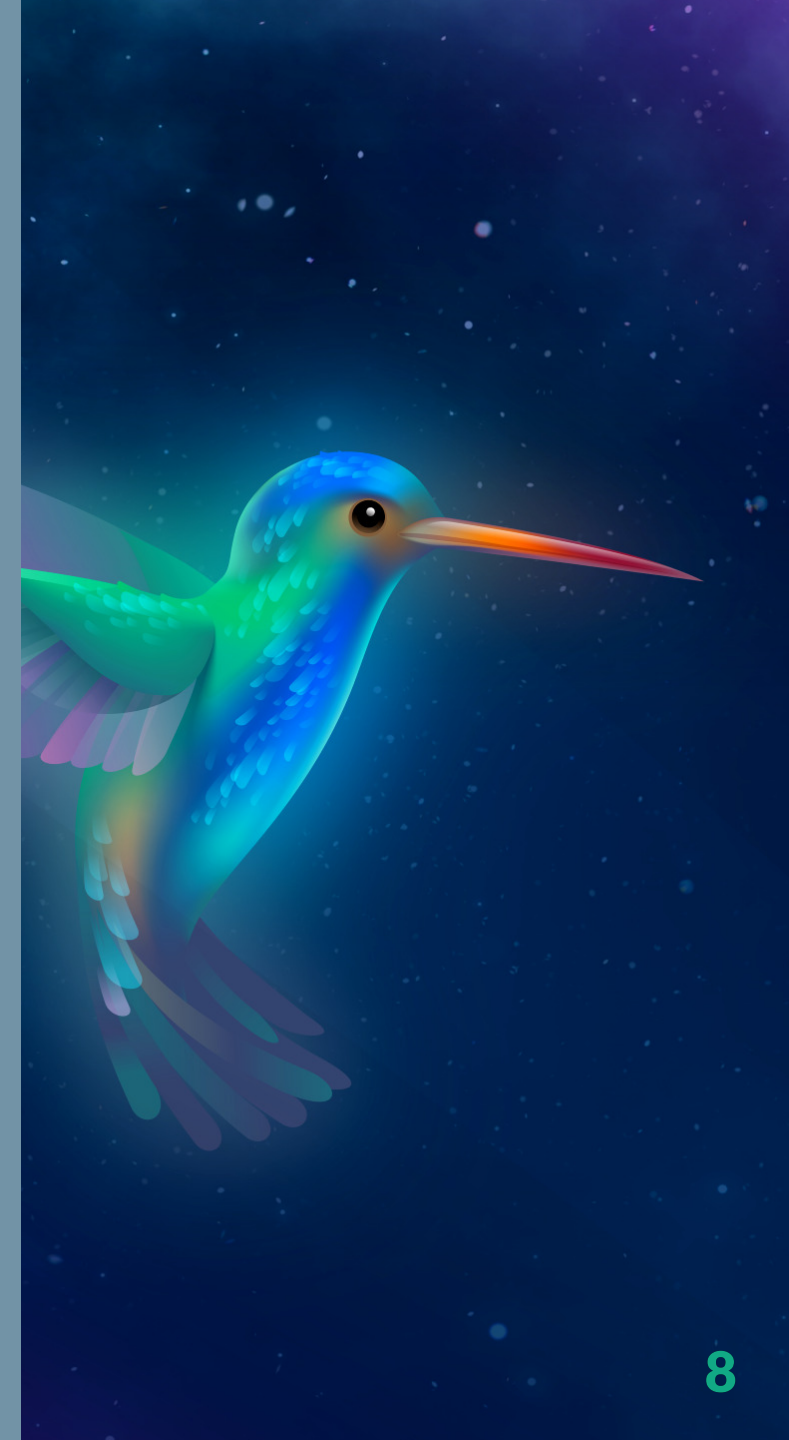
# Vorbereitungen

## Semaphor

- $count_S \geq 0$ : Menge an **Platz** in der **kritischen Region**
- $count_S \leq 0$ : **Anzahl Prozesse**, die auf die **kritische Region** warten
- `up()`: mitteilen, dass **neuer Platz** in kritischen Region **verfügbar**
- `down()`: **Platz** in kritischer Region **reservieren**



# Übungsblatt





# Übungsblatt

## Aufgabe 1a)

```
solange (i < n) {  
  mein_i := i;  
  i := i + 1;  
  if (mein_i < n) {  
    a := vektor_a[mein_i];  
    b := vektor_b[mein_i];  
    s := komplizierte_funktion(a, b);  
    ergebnis[mein_i] := s;  
  }  
}
```

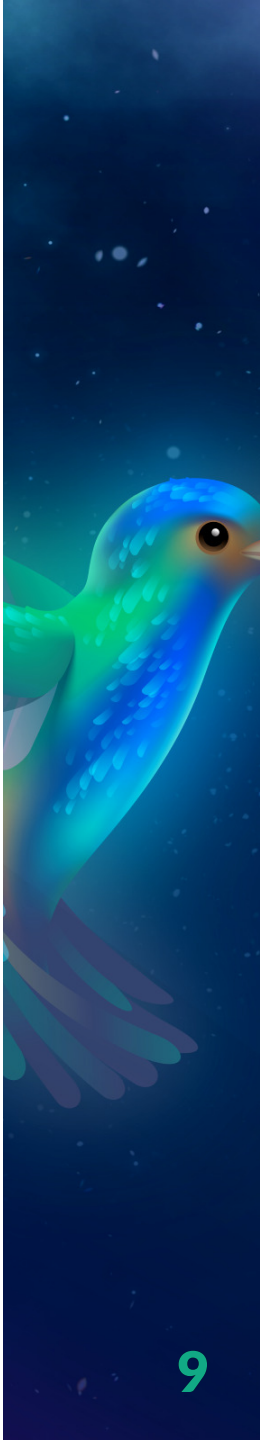


Sema (count=1)

center=1

a == 0 ->

count ->

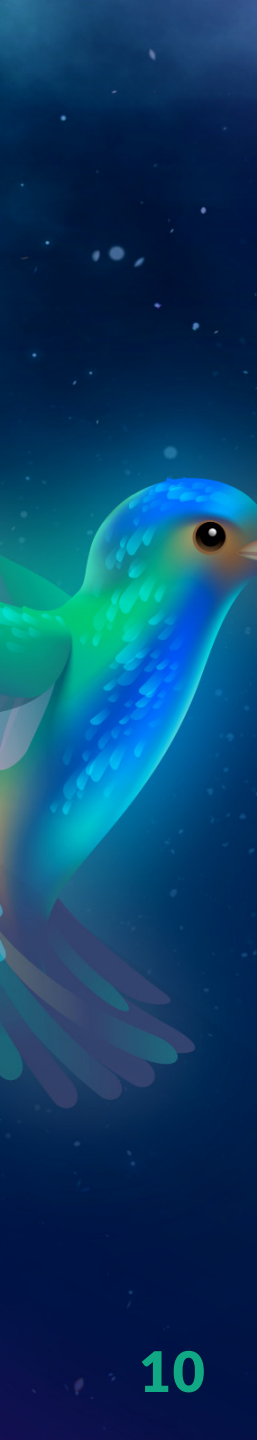


# Übungsblatt

## Aufgabe 1a)

- **Prozess p1** speichert das aktuelle `i` lokal in `mein_i`
- **Prozess p2** speichert das aktuelle `i` lokal in `mein_i`
- **beide** inkrementieren `i`
- jetzt speichert **p3** das aktuelle `i` lokal in `mein_i`
- **p1** und **p2** haben den gleichen Index `old_i`
- **p3** berechnet den Index `old_i + 2`
- `old_i + 1` ausgelassen

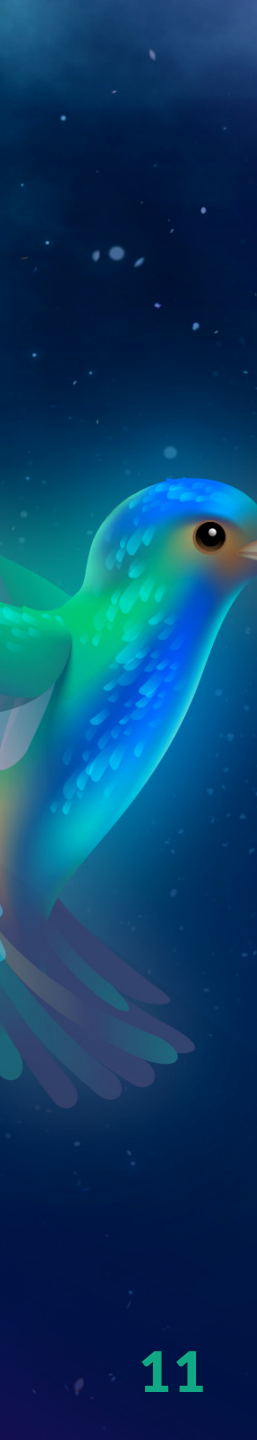
$i+2$



# Übungsblatt

## Aufgabe 1 b)

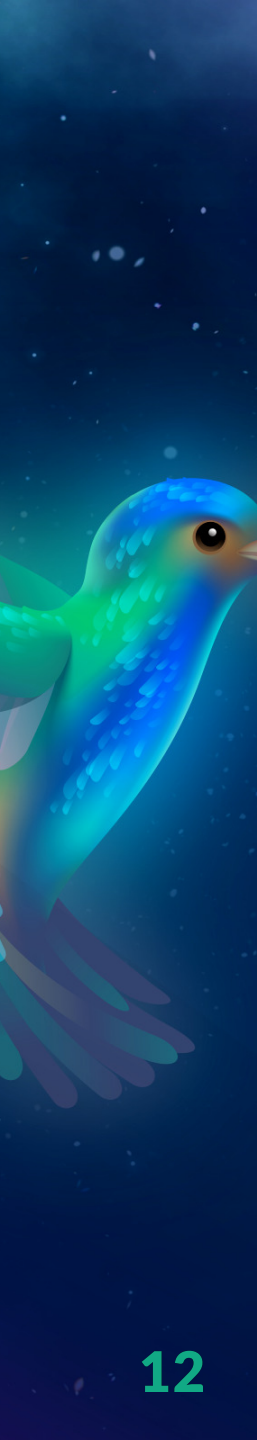
```
solange (i < n) {  
    mutex_lock(m);  
    mein_i := i;  
    i := i + 1;  
    mutex_unlock(m);  
    if (mein_i < n) {  
        a := vektor_a[mein_i];  
        b := vektor_b[mein_i];  
        s := komplizierte_funktion(a, b);  
        ergebnis[mein_i] := s;  
    }  
}
```



# Übungsblatt

## Aufgabe 1b)

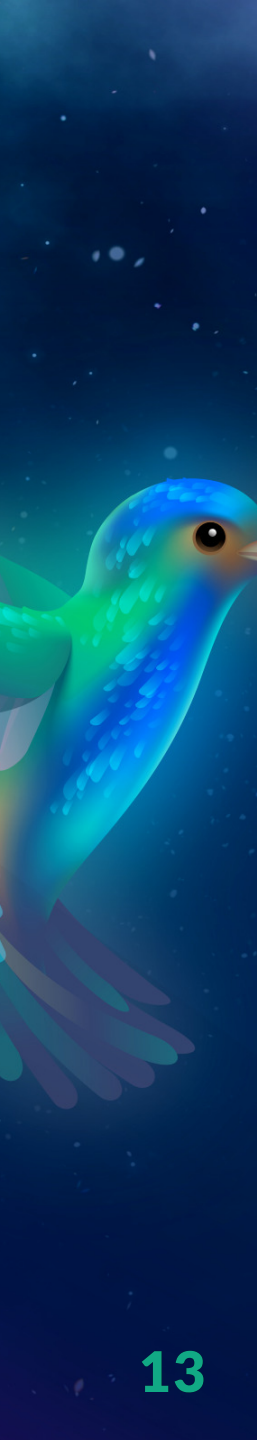
- `mein_i := i;` und `i := i + 1;` müssen **atomar** in einem Zug bearbeitet werden, ohne, dass ein andere Prozess dazwischenfunken kann → daher mit einem **Mutex** sicherstellen, dass da auch **nur einer rein kann** und raus kommt, bevor ein andere rein kann



# Übungsblatt

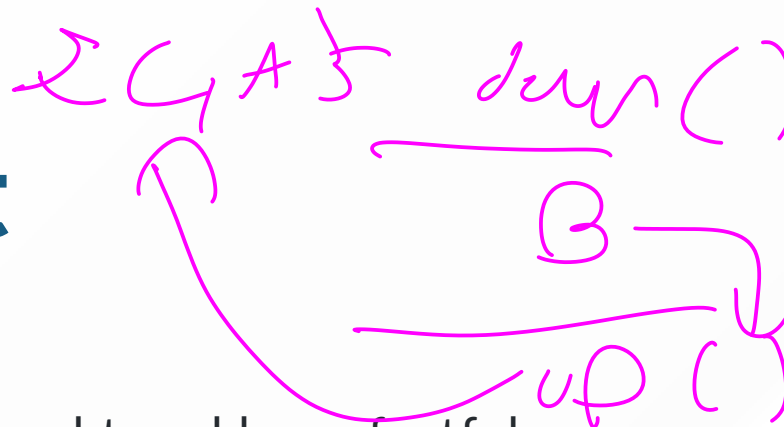
## Aufgabe 2a)

- **Prozess A** wird **schlafen gelegt**, wenn
  - Prozess A die Methode `S.down()` aufruft
  - der Zähler **vor** dem Aufruf  $\leq 0$  bzw.  $< 1$  war und der Zähler **nach** dem Aufruf  $\leq -1$  bzw.  $< 0$  war
- **Prozess A** wird **aufgeweckt**, wenn
  - sich Prozess A (an erster Stelle) in der **Warteschlange** des Semaphors befindet → Zähler **vor** dem Aufruf  $\leq -1$  bzw.  $< 0$  und **nach** dem Aufruf  $\leq 0$  bzw.  $< 1$
  - **Prozess B** die Methode `S.up()` aufruft
  - **Prozess A** wird sich nicht selber mit `up()` aufwecken können, da er ja in der Queue ein Nickerchen macht



# Übungsblatt

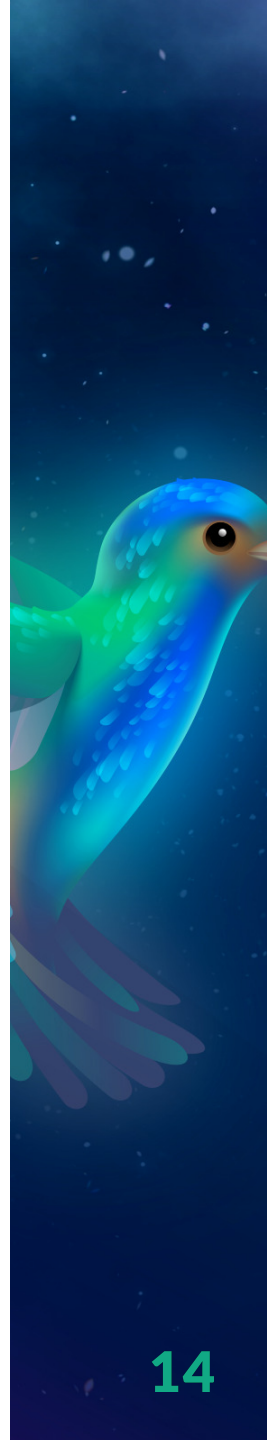
## Aufgabe 2b)



FIFO  
= first in,  
first out

- Ja, **Prozess C** wird aufgeweckt und kann fortfahren:
  - Wird `up()` aufgerufen und ist die Warteschlange **nicht leer**, dann wird immer ein **Prozess** aus der Warteschlange **geholt** und aufgeweckt
  - Der Betrag des Zählerstands vor dem Inkrementieren entspricht der **Länge der Warteschlange**. Wie in der Vorlesung:
    - "Wenn der Zählerstand nach dem Inkrementieren  $\leq 0$  ist, dann wird ein **Prozess** aus der Warteschlange **geholt** und aufgeweckt"
- **Queue** laut **Wikipedia**:
  - Eine Queue kann [...] eine beliebige Menge von Objekten aufnehmen und gibt diese **in der Reihenfolge ihres Einfügens** wieder zurück
  - aber später steht auch: "Es gibt Implementierungen, die **gar keinen prinzipiellen Unterschied zwischen Stacks und Queues machen.**"

FIFO  
= first in  
first out

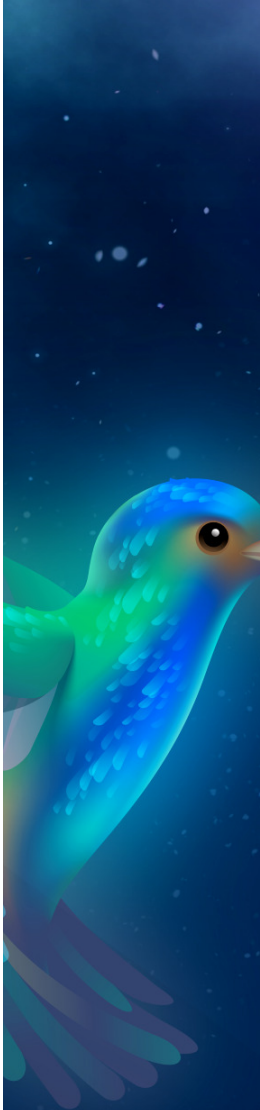




# Übungsblatt

## Aufgabe 2c)

Anweisung	Semaphor $S$		Prozess $A$	Prozess $B$	Prozess $C$	Prozess $D$
	Zähler	Warteschl.				
1. Initialisierung	2	leer	bereit	bereit	bereit	bereit
2. Prozess $A$ : $S.down()$	1	leer	bereit	bereit	bereit	bereit
3. Prozess $B$ : $S.down()$	0	leer	bereit	bereit	bereit	bereit
4. Prozess $C$ : $S.down()$	-1	[C]	bereit	bereit	blockiert	bereit
5. Prozess $D$ : $S.down()$	-2	[C,D]	bereit	bereit	blockiert	blockiert
6. Prozess $A$ : $S.up()$	-1	[D]	bereit	bereit	bereit	blockiert
7. Prozess $B$ : $S.up()$	0	leer	bereit	bereit	bereit	bereit
8. Prozess $C$ : $S.up()$	1	leer	bereit	bereit	bereit	bereit



# Übungsblatt

## Aufgabe 3a)

$< 0$

$count \neq -1$

Gemeinsame Initialisierung

```
1 a := 0
2 b := 0
3 summe := 0
4 // definieren und initialisieren Sie hier Ihre Semaphore
```

$S = \text{Semaphore}(count = 0)$

Prozess A

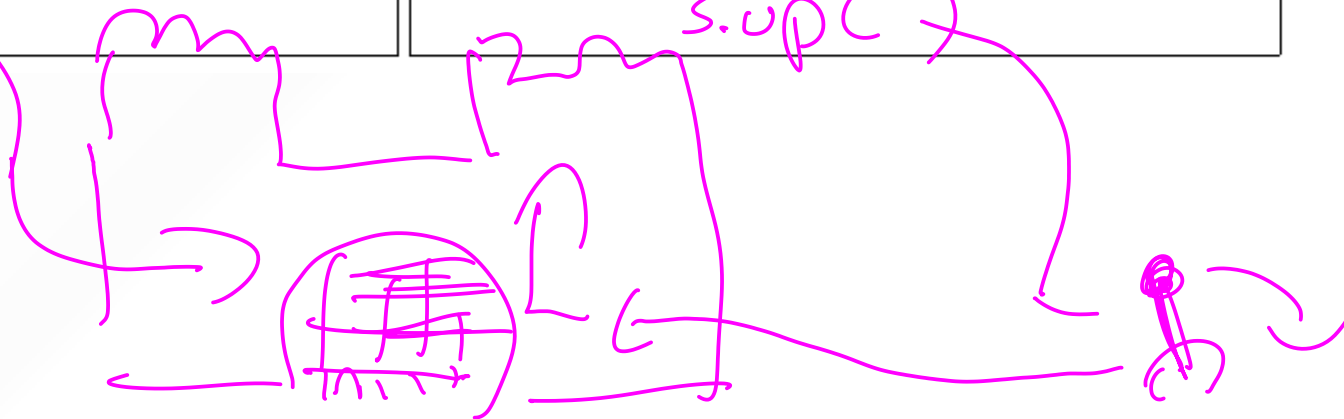
```
5 a := berechne_Teilergebnis_a()
6 summe := a + b
```

$S.down()$

Prozess B

```
b := berechne_Teilergebnis_b()
```

$S.up()$



# Übungsblatt

## Aufgabe 3a)

Gemeinsame Initialisierung

```
1 a := 0
2 b := 0
3 summe := 0
4 Semaphore warte_auf_b := new Semaphore(counter = 0);
```

Prozess A

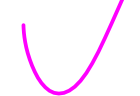
```
5 a := berechne_Teilergebnis_a()
6 warte_auf_b.down()
7 summe := a + b
```

Prozess B

```
b := berechne_Teilergebnis_b()
warte_auf_b.up()
```

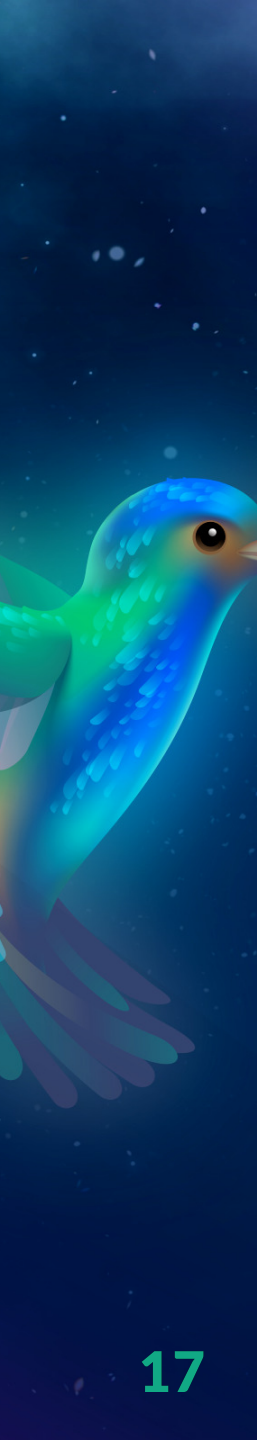
Fall 1:

A first on



Fall 2:

B first on



# Übungsblatt

## Aufgabe 3b)

$M = \text{Semaphor (count} = 0)$   
 $A = \text{Semaphor (count} = 0)$

Gemeinsame Initialisierung		
1	// definieren und initialisieren Sie hier Ihre Semaphoren	
2	Arbeiter A	Arbeiter B
3	Zahnrad_wechseln()	Schrauben_anziehen()
4		

- so schnell wie möglich
- nur arbeiten, wenn Maschine aus

# Übungsblatt

## Aufgabe 3b)

Gemeinsame Initialisierung

1 Semaphore warte\_bis\_Maschine\_aus := new Semaphore(counter = 0)  
2 Semaphore warte\_bis\_A\_fertig := new Semaphore(counter = 0)  
3 ~~Semaphore warte\_bis\_B\_fertig := new Semaphore(counter = 0)~~

Arbeiter A

4  
5 warte\_bis\_Maschine\_aus.down()  
6  
7 Zahnrad\_wechseln()  
8 warte\_bis\_A\_fertig.up()  
9  
10

Arbeiter B

warte\_bis\_Maschine\_aus.down()  
Schrauben\_anziehen()  
warte\_bis\_B\_fertig.up()

Arbeiter C

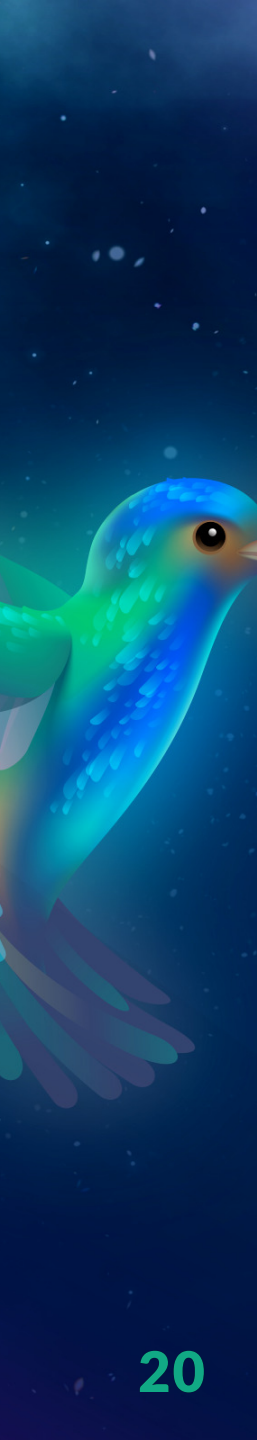
Maschine\_ausschalten()  
warte\_bis\_Maschine\_aus.up()  
warte\_bis\_Maschine\_aus.up()  
Material\_nachfuellen()  
~~warte\_bis\_A\_fertig.down()~~  
~~warte\_bis\_B\_fertig.down()~~  
Maschine\_einschalten()

# Übungsblatt

## Aufgabe 3b)

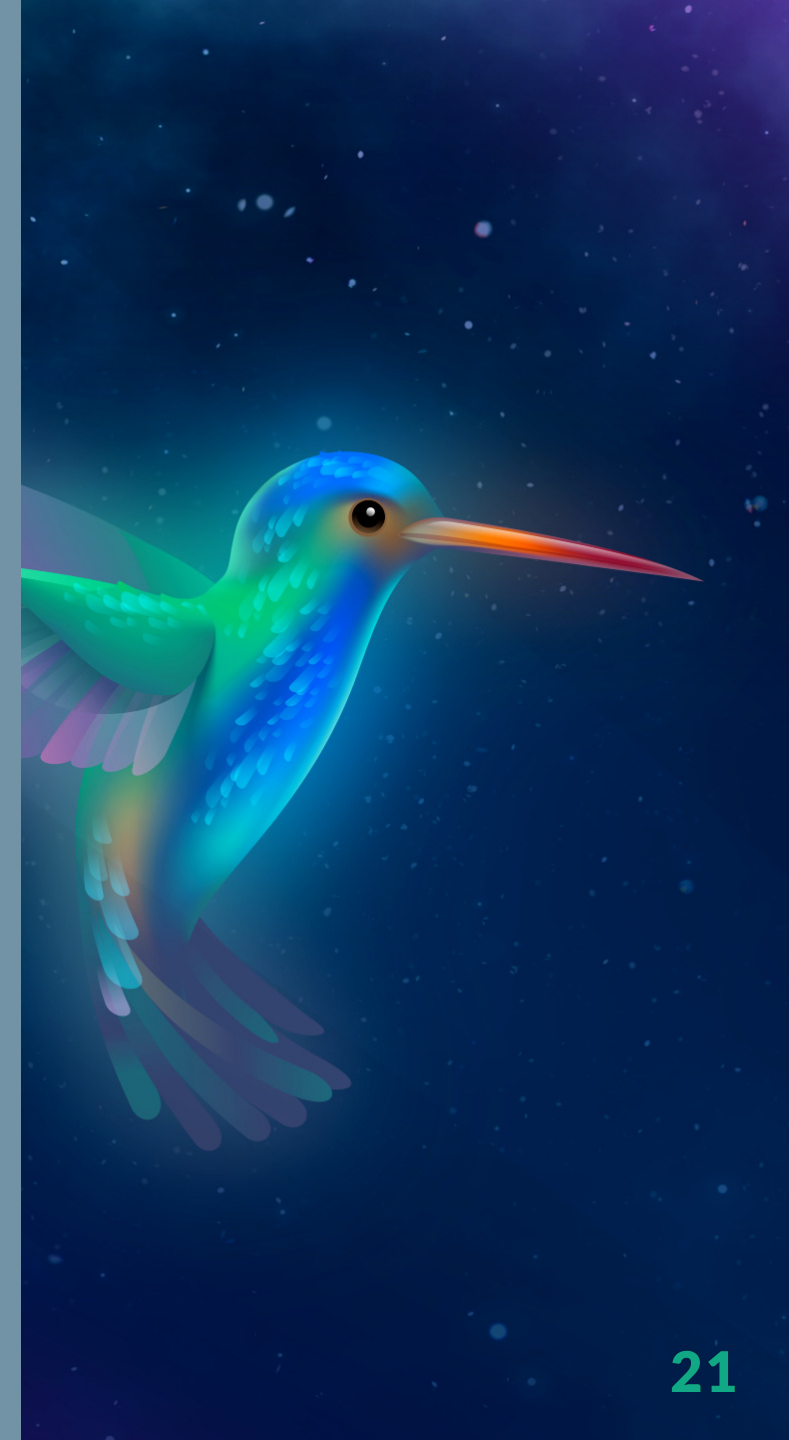
### Alternativlösung

- Semaphore `warte_bis_A_fertig` und `warte_bis_B_fertig` können zu einem Semaphor **zusammengefasst** werden
- Das Zusammenfassen mit dem Semaphor `warte_bis_Maschine_aus` funktioniert **nicht** ohne weitere Hilfsvariablen, da sonst **nicht sichergestellt** wäre, dass **vor** dem Einschalten der Maschine die **Arbeiten** der anderen Arbeiter **fertiggestellt** wurden






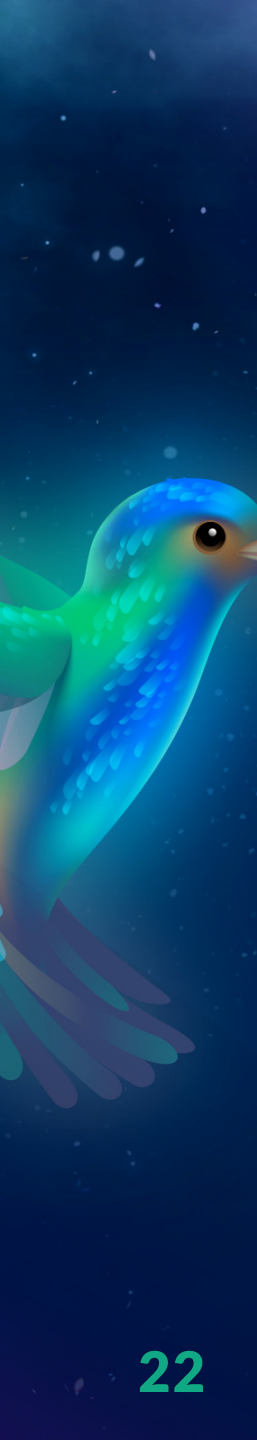
# Quellen



# Quellen


## Wissenquellen

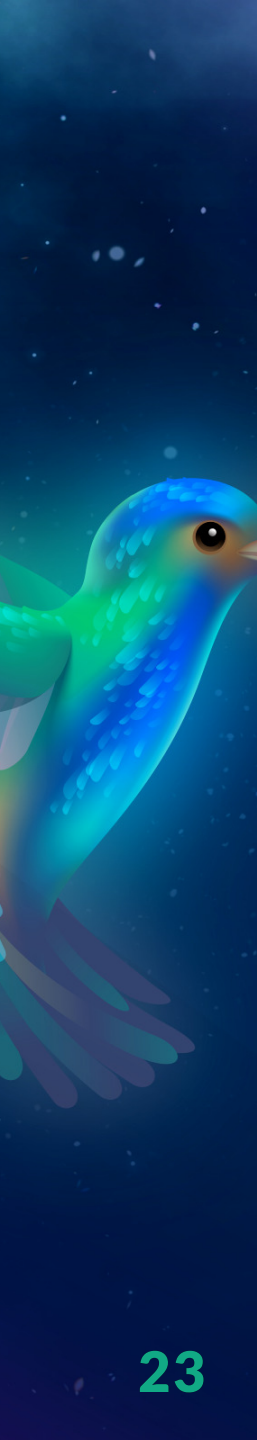
- 
- source



# Quellen

## Bildquellen

- 
- source



**Vielen Dank für  
eure  
Aufmerksamkeit!**

