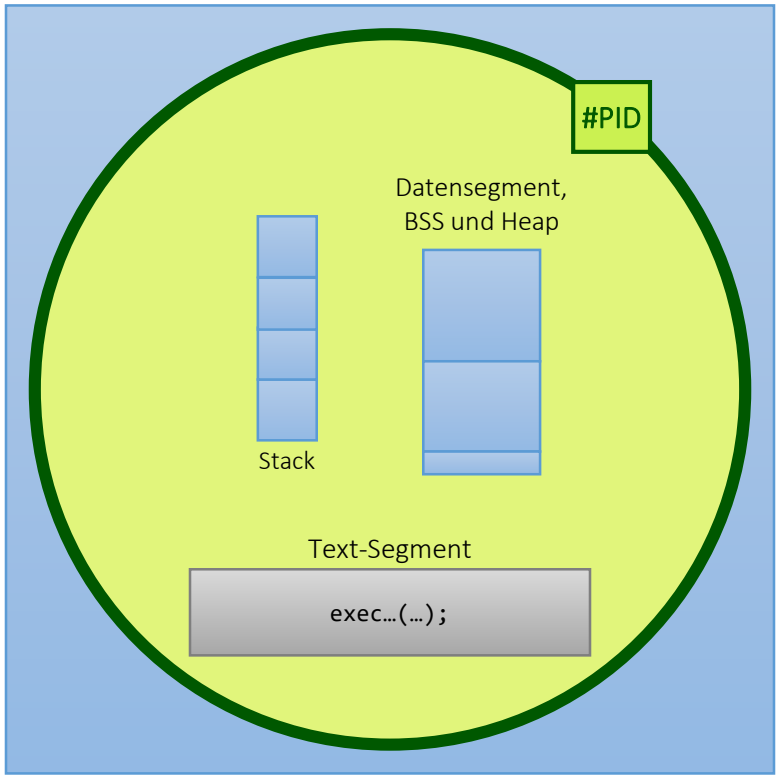
Fäden und Synchronisation

Emilio Pielsticker

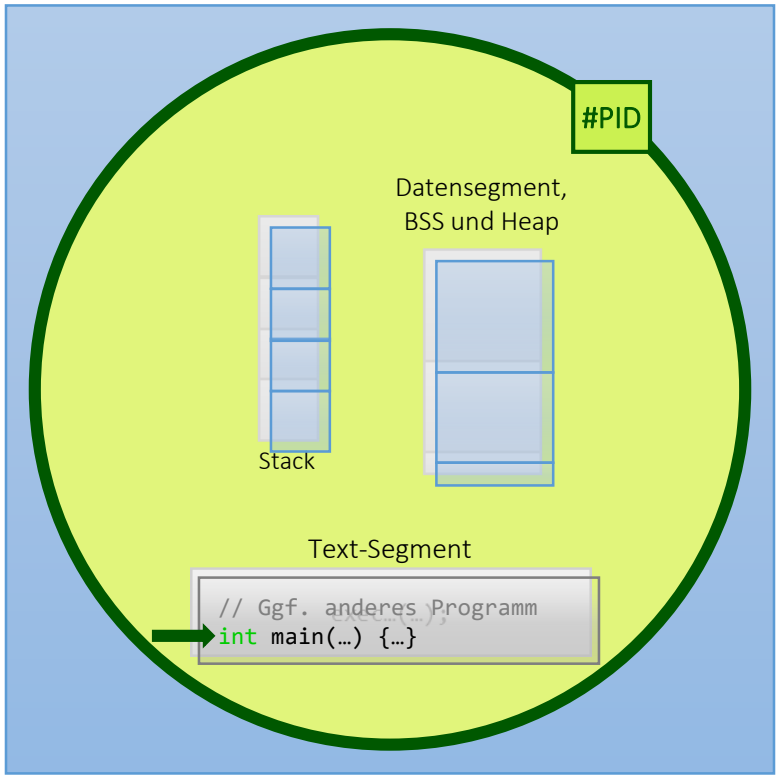
Prozesse und Fäden

- **Prozess** (**schwergewichtige Prozesse**): Haben eigenen Adressraum
  - **`pid_t`** als Datentyp für die Prozess-Identifikation
- **Faden** (engl. **thread**): Teilt sich Adressraum mit schwergew. Prozess
  - Ein Prozess kann mehrere Fäden haben
  - Jeder Faden hat seinen eigenen Stack
  - Heap-, BSS- und Daten-Segment werden geteilt
  - **`pthreads`** bietet die in POSIX standardisierte Schnittstelle
  - Jeder Faden hat eine eineindeutige Identifikation (**`pthread_t`**)
- **Kettfaden** (engl. **warp**): Nicht atomar-zerlegbarer Zusammenschluss von Fäden (→ GPU-Programmierung)

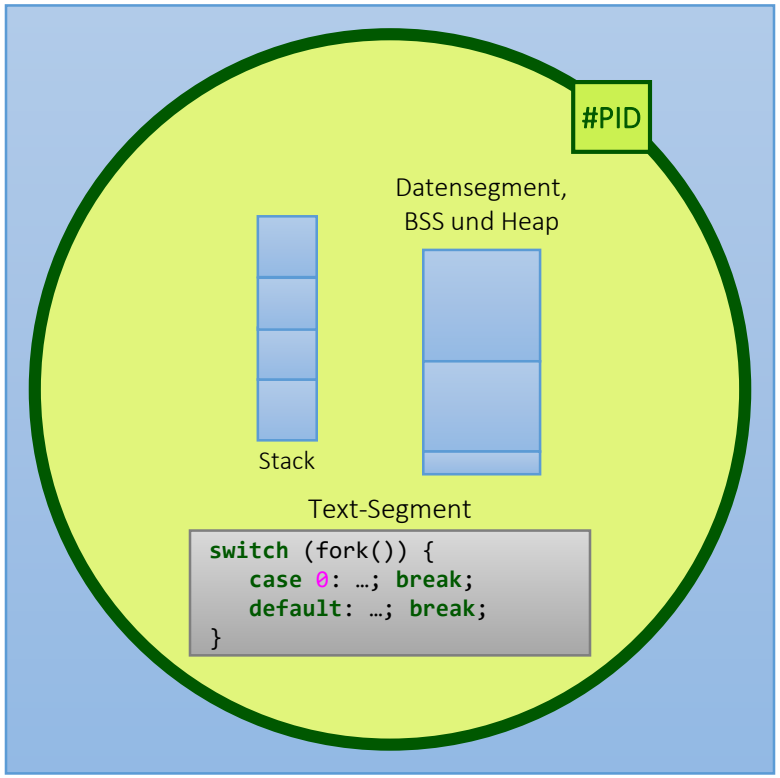
Schwergewichtiger Prozess (Skizze)



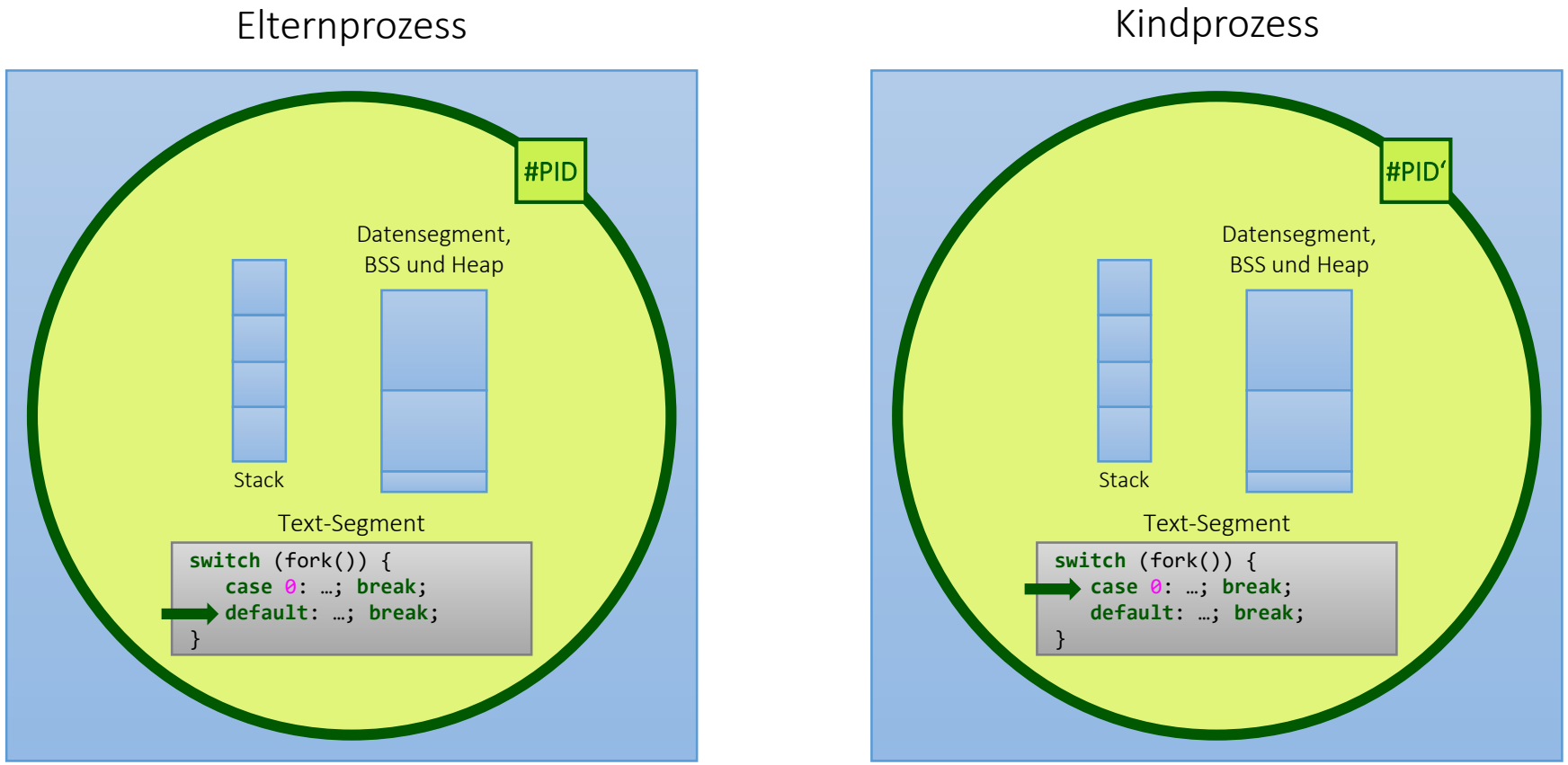
Überschreiben schwergewichtiger Prozesse



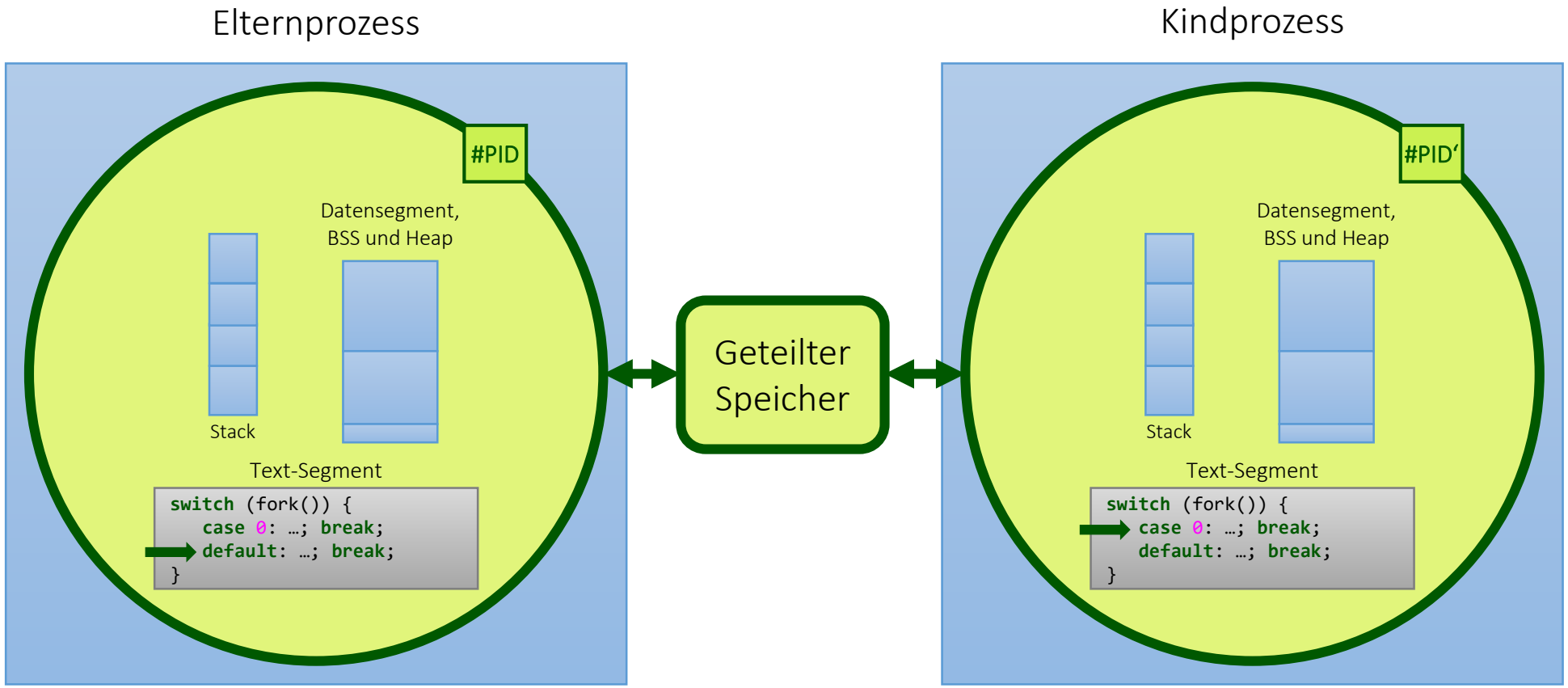
Schwergewichtiger Prozess vor `fork` (Skizze)



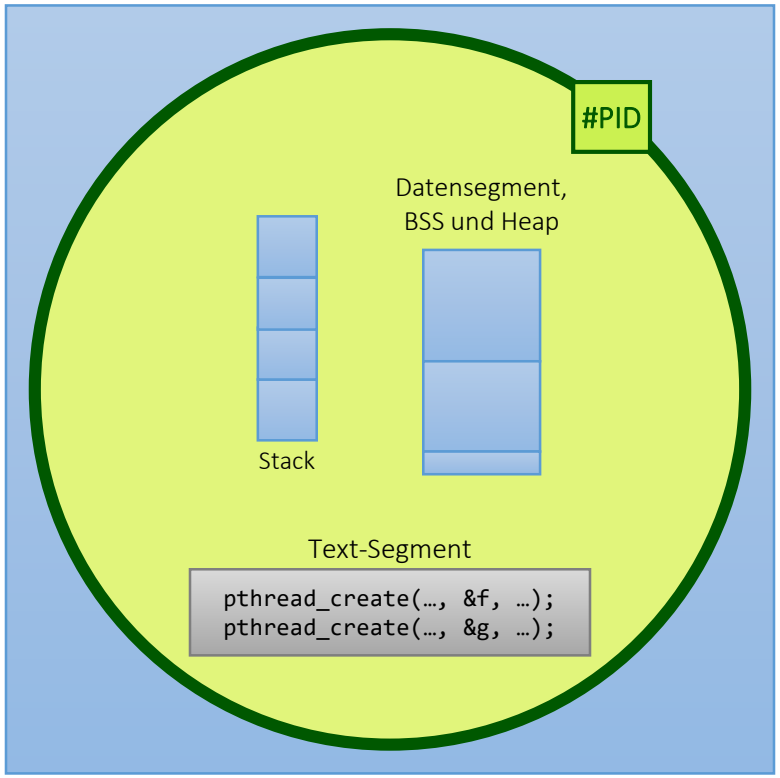
Schwergewichtiger Prozess nach **fork** (Skizze)



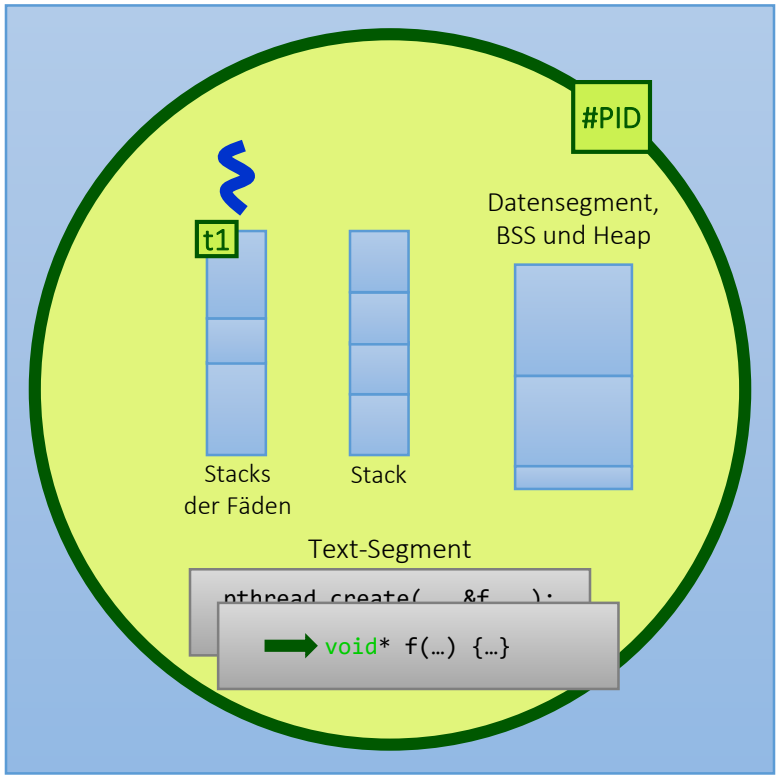
Geteilter Speicher (→ `shmget(2)`)



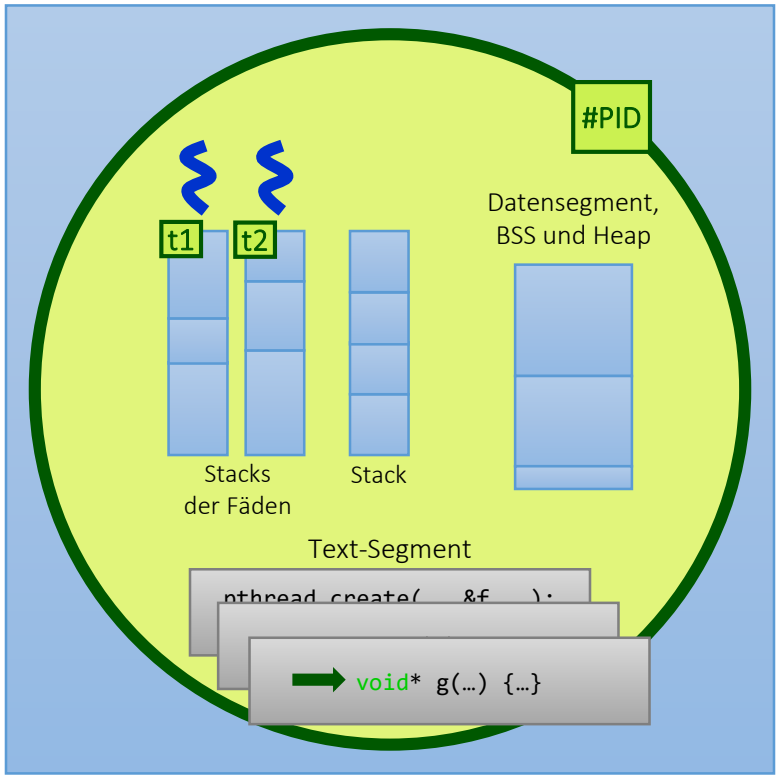
Prozess vor `pthread_create` (Skizze)



Prozess mit einem Faden (Skizze)



Prozess mit zwei Fäden (Skizze)



POSIX-Fäden erzeugen und beenden

- `pthread_create(&thread, attr, &f, arg)` erzeugt Faden
  - `thread` ist die Faden-Variable vom Typ `pthread_t`
  - `attr` ist ein Zeiger auf den Arbeitsbereich des Fadens
  - `f` ist die auszuführende Funktion
  - `arg` ist ein Zeiger auf die Argumente mit der `f` ausgeführt werden soll
- `0 bei Erfolg`
- `≠0 bei Misserfolg`
- In `f`: `pthread_exit(p)`: Beendet Faden und liefert Zeiger `p` zurück

Beispiel eines POSIX-Fadens

```c
void* hello(void* arg) {
   printf("Hallo!\n");
   pthread_exit(NULL);
}
```

```c
#include <pthread.h>
#include <stdio.h>

int main(void) {
   int status, s;
   pthread_t thread;
   s = pthread_create(&thread, NULL, &hello, NULL);
   if (s) { /* Fehlerbehandlung */ }
   status = pthread_join(thread, NULL);
   if (status) { /* Fehlerbehandlung */ }
   return 0;
}
```

Faden mit Hauptprozess zusammenführen

- `pthread_join(&thread, &q)` legt den aufrufenden Faden/Prozess schlafen, bis der Faden `thread` terminiert
- `q` ist eine Zeigervariable, die mit dem durch `pthread_exit(p)` zurückgegebenen Zeiger `p` überschrieben werden soll

POSIX-Fäden vergleichen & Identität feststellen

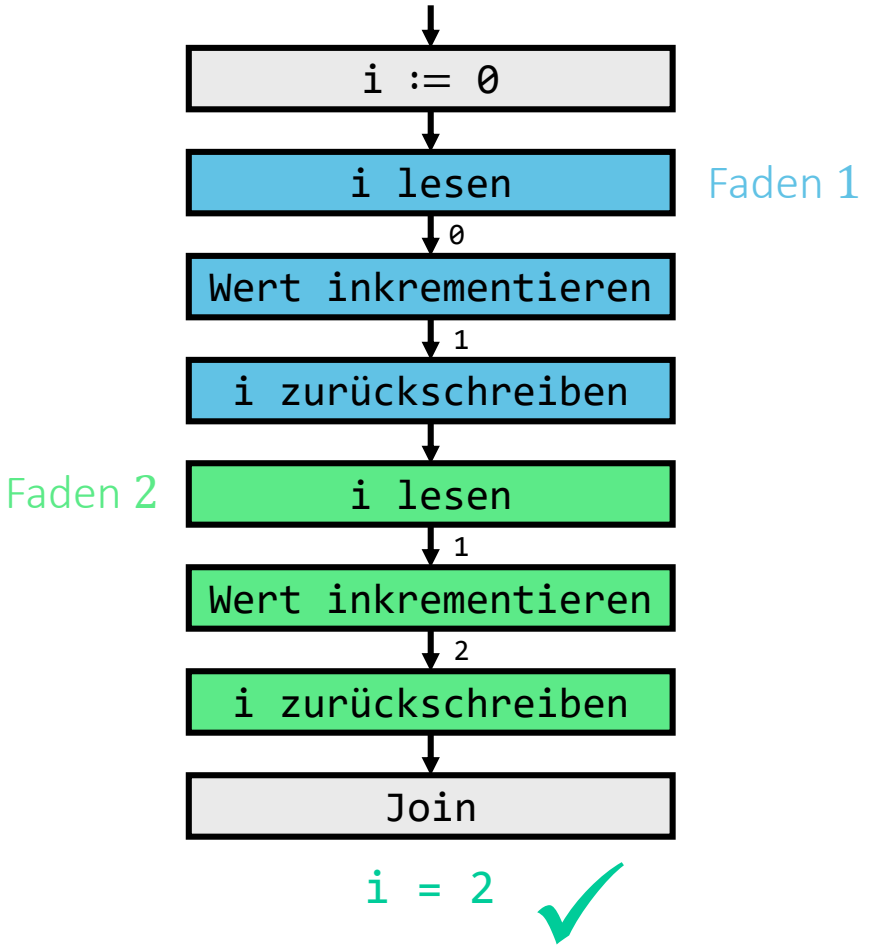
- `pthread_equal(&thread1, &thread2)`
  - liefert einen Wert ungleich 0 zurück, falls es sich um den selben Faden handelt
- `pthread_self()` liefert die ID eines Fadens als Rückgabewert

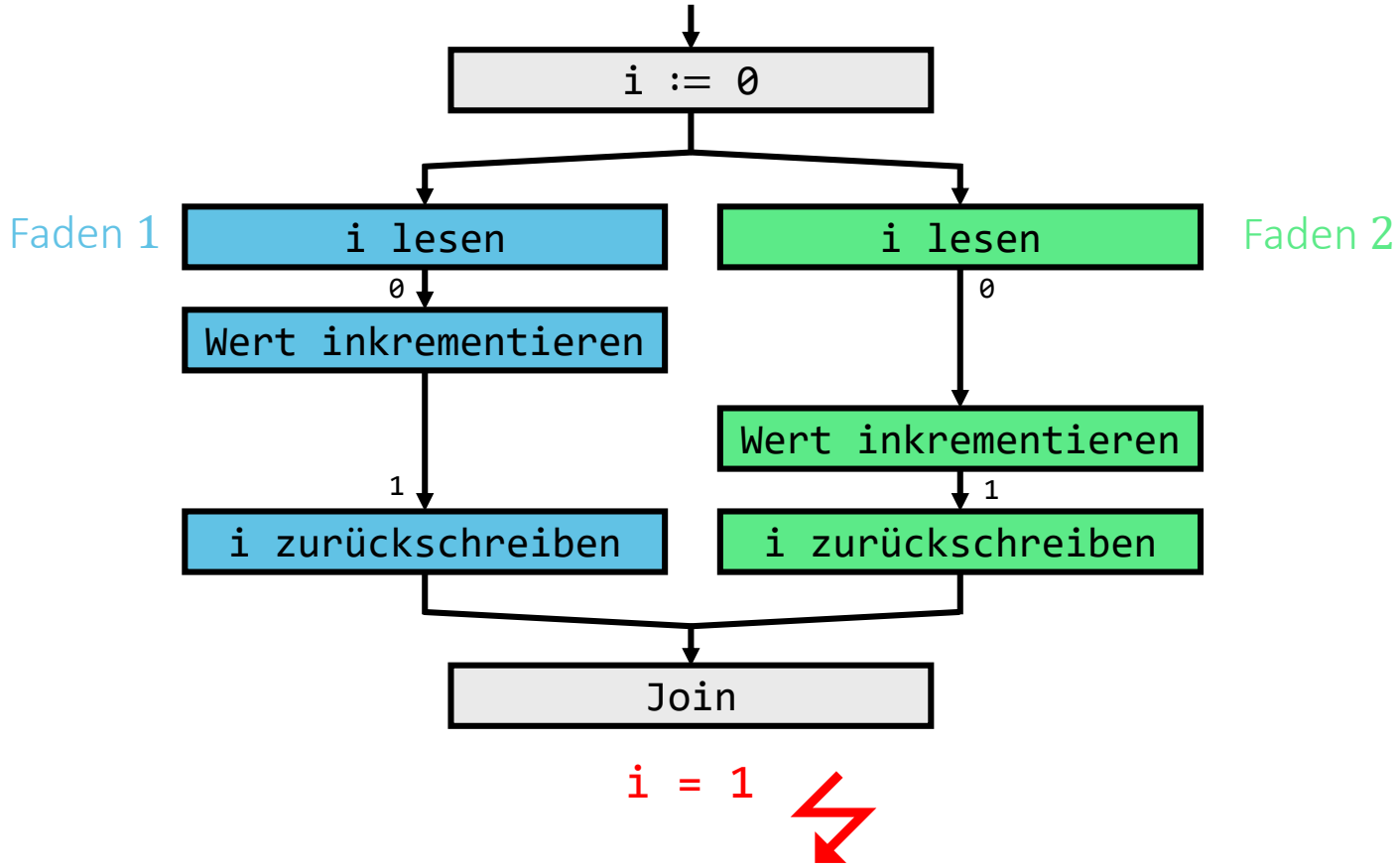
Ein Beispiel mit zwei Fäden

```c
void* f1(void* arg) {
   i++;
   return NULL;
}

void* f2(void* arg) {
   i++;
   return NULL;
}
```

```c
#include <pthread.h>
#include <stdio.h>

int i = 0;

int main(void) {
   int status, s, t;
   pthread_t thread1;
   pthread_t thread2;
   s = pthread_create(&thread1, NULL, &f1, NULL);
   t = pthread_create(&thread2, NULL, &f2, NULL);
   …
   return 0;
}
```

Wettlaufsituation





Beim gleichzeitigen starten der Fäden entsteht eine **Wettlaufsituation**. Durch Synchronisationsmechanismen wie Semaphore oder Schlossvariablen können solche Situationen vermieden werden.

Semaphore

- Ein **Semaphor** (σῆμα dt. Zeichen, φέρειν dt. tragen) ist eine Datenstruktur zur Verwaltung **beschränkter Ressourcen** in digitalen Systemen → Analogie: Eisenbahnsignal
- Die Semaphor wird als Zähler mit der maximalen Anzahl n der entsprechenden Ressource initialisiert: $s := n$
- P( ): **Prolaag** = probeer te verlagen (dt. versuche zu **senken**)
  - $s > 0$: Ressource wird beansprucht und $s := s - 1$
  - $s = 0$: Warten bis Ressource wieder verfügbar ist: $s > 0$
- V( ): **Verhogen** bzw. **vrijgeven** dt. **freigeben**: $s := s + 1$

Bei den Begriffen 'Prolaag' (probeer te verlagen), 'verhogen' bzw. 'vrijgeven' handelt es sich niederländische Worte.

POSIX-Semaphore

- `sem_init(&s, 0, x)` initialisiert ein Semaphor `s` mit dem Wert `x`
- `sem_destroy(&s)` zerstört ein Semaphor `s`
- `sem_getvalue(&s, &v)` schreibt Wert des Semaphors `s` in `v`
- `P()`: `sem_wait(&s)`
- `V()`: `sem_post(&s)`

- `0` bei Erfolg
- $\neq 0$ bei Misserfolg

Binäre Semaphore und Schlossvariablen

- Nimmt ein Semaphor ausschließlich die Werte 0 und 1 an, spricht man von einem **binären Semaphor**
- Wird ein binärer Semaphor ausschließlich von genau einem Aktivitätsträger gesenkt und freigegeben und von anderen Aktivitätsträgern ausschließlich gelesen, spricht man von einem **gegenseitigen Ausschluss** (engl. **mutual exclusion** bzw. **Mutex**) oder von einer **Schlossvariablen** (**Lock**)
  - `P()` ≡ `pthread_mutex_lock()`
  - `V()` ≡ `pthread_mutex_unlock()`

Initialisieren und zerstören von POSIX-Schlössern

- `pthread_mutex_init(&s, attr)` initialisiert Schloss `s` (ungesperrt)
- `pthread_mutex_destroy(&s)` zerstört ein Schloss `s`

- `0 bei Erfolg`
- `≠0 bei Misserfolg`

- `const pthread_mutexattr_t* attr`
  ist ein Zeiger auf weitere Attribute und wird meist auf `NULL` gesetzt
- Ein POSIX-Schloss `s` ist vom Typ `pthread_mutex_t`

Öffnen und schließen von POSIX-Schlössern

- `pthread_mutex_lock(&s)` schließt Schloss `s`
- `pthread_mutex_unlock(&s)` öffnet ein Schloss `s`

- `0` bei Erfolg
- $\neq 0$ bei Misserfolg

Ein Beispiel mit zwei Fäden und einem Schloss

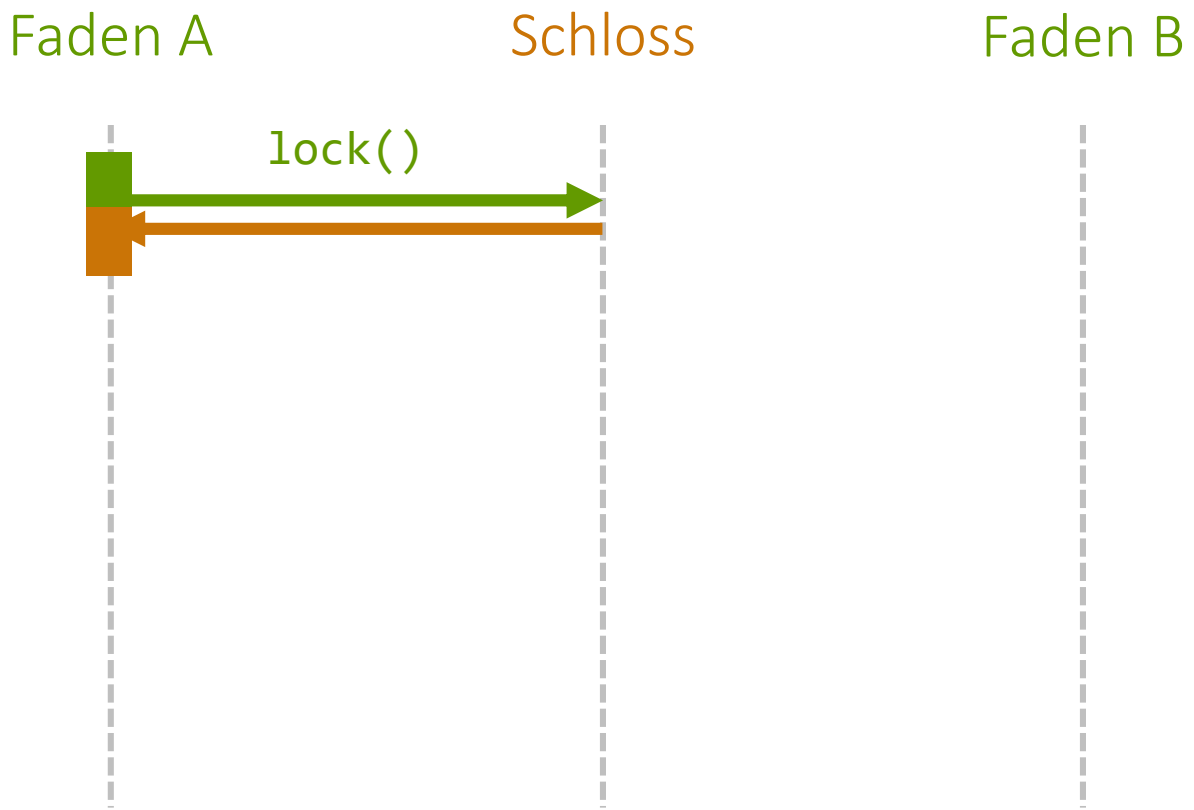
```c
void* f1(void* arg) {
 …
 pthread_mutex_lock(&l);
 i++;
 pthread_mutex_unlock(&l);
 …
 return NULL;
}

void* f2(void* arg) {
 …
 pthread_mutex_lock(&l);
 i++;
 pthread_mutex_unlock(&l);
 …
 return NULL;
}
```

```c
#include <pthread.h>
#include <stdio.h>

int i = 0;
pthread_mutex_t l;

int main(void) {
   int status, s, t;
   pthread_t threadA;
   pthread_t threadB;
   pthread_mutex_init(&l, NULL);
   s = pthread_create(&threadA, NULL, &f1, NULL);
   t = pthread_create(&threadB, NULL, &f2, NULL);
   …
   pthread_mutex_destroy(&l);
   return 0;
}
```

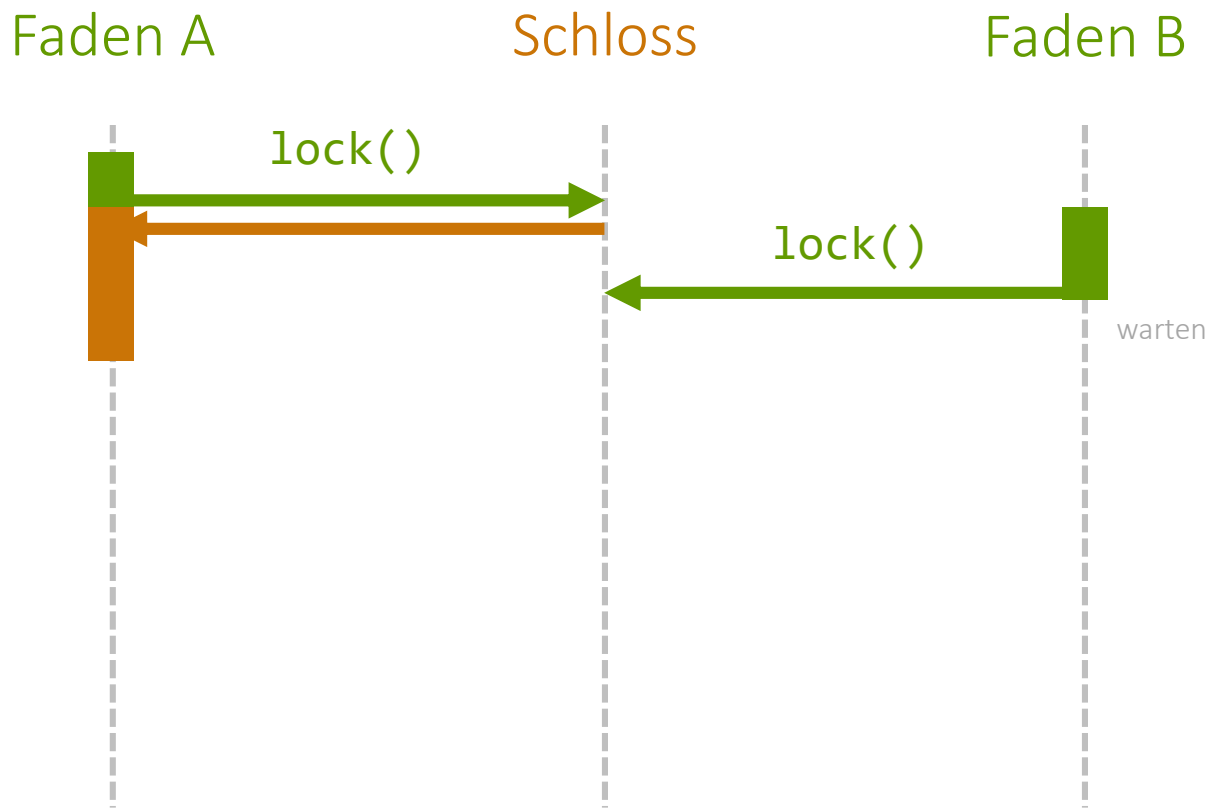
Achtung: Fehlerbehandlung fehlt!

Gegenseitiger Ausschluss

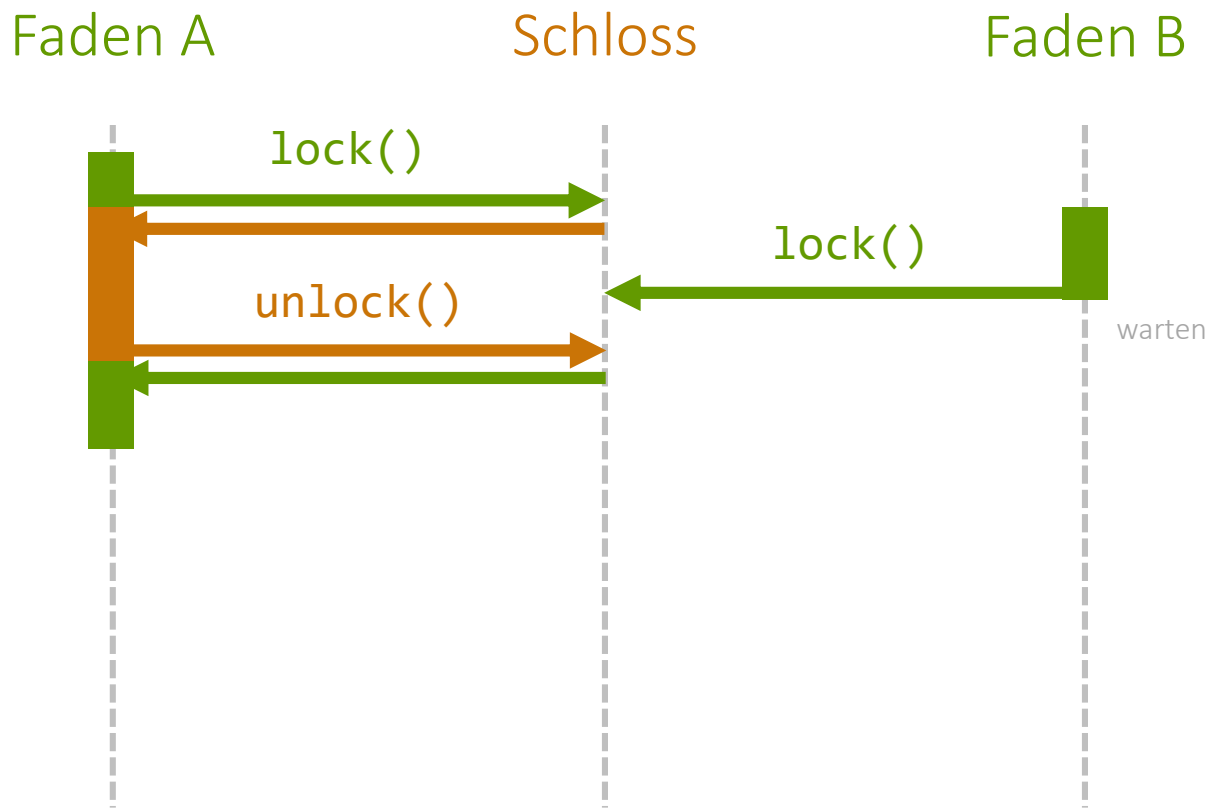
Gegenseitiger Ausschluss I



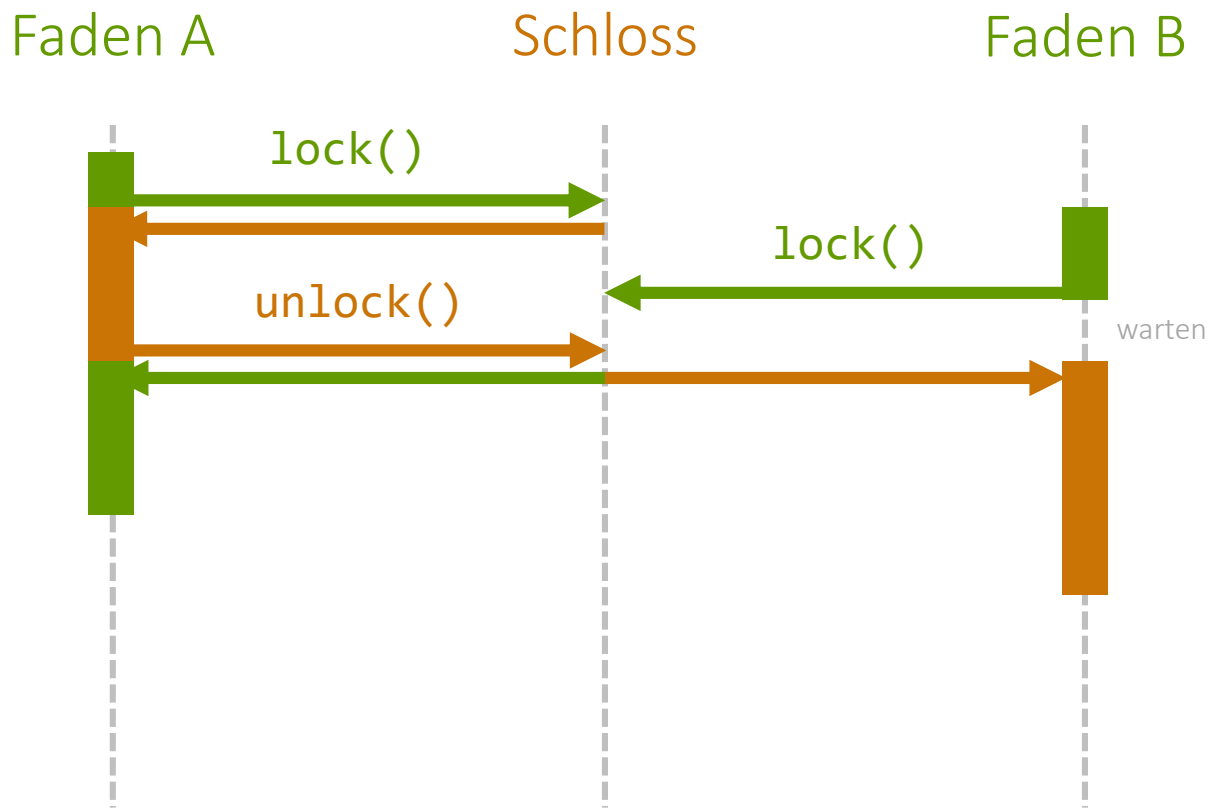
Gegenseitiger Ausschluss II



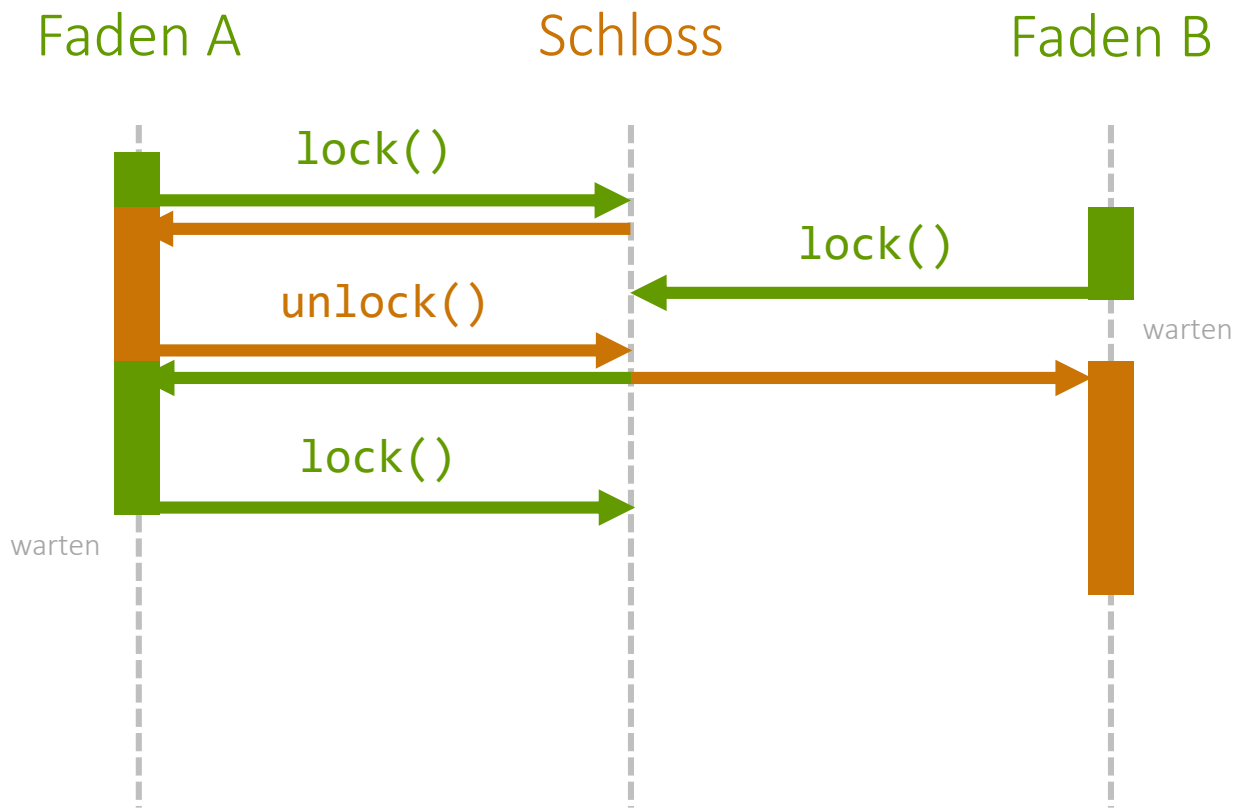
Gegenseitiger Ausschluss III



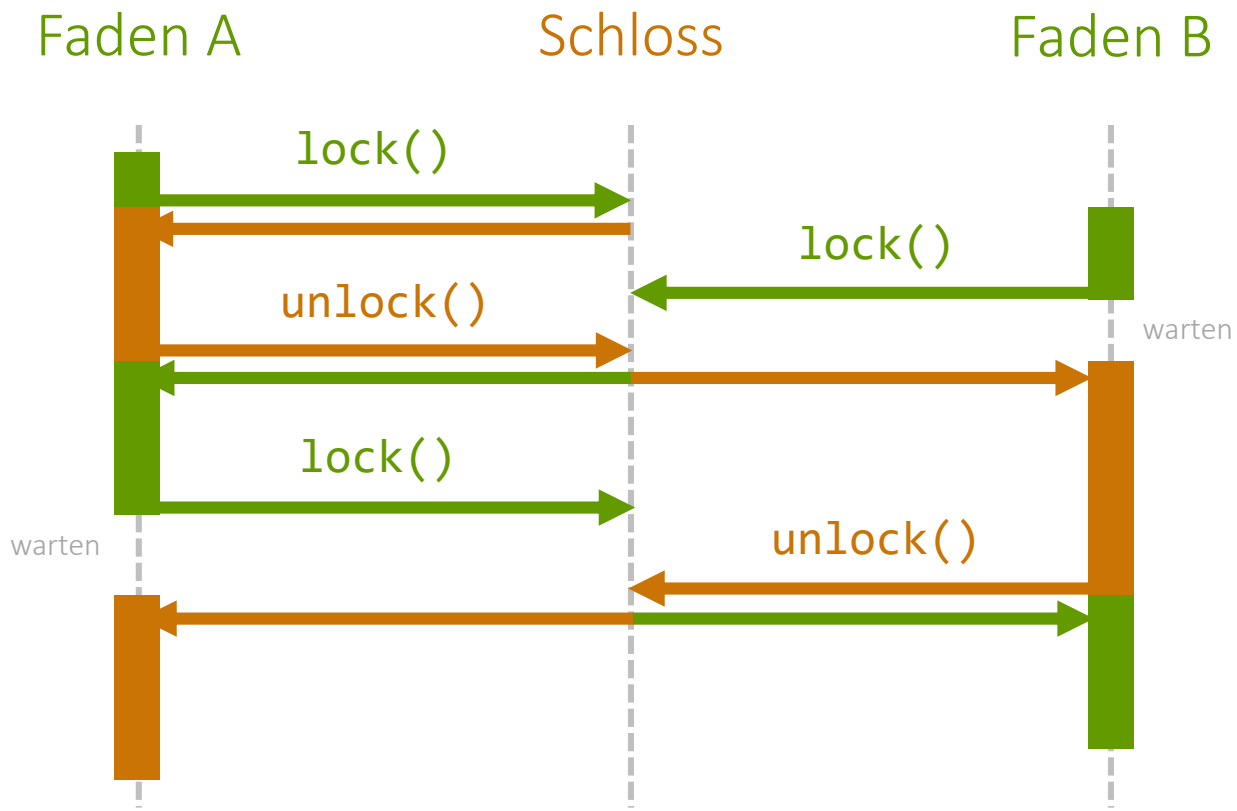
Gegenseitiger Ausschluss IV



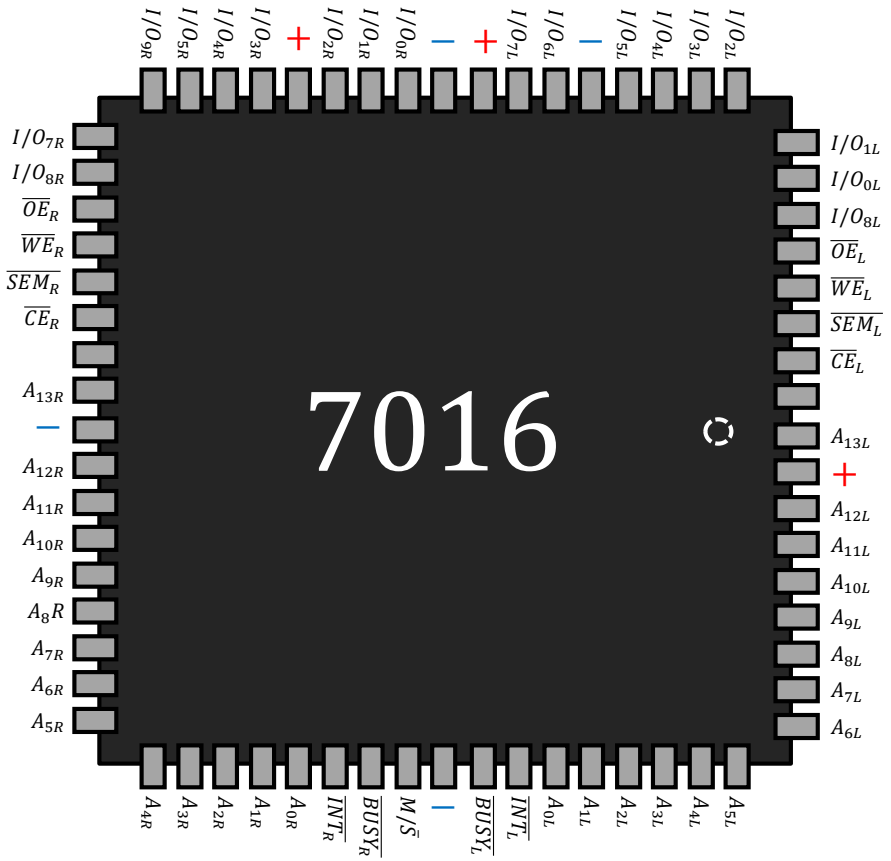
Gegenseitiger Ausschluss V



Gegenseitiger Ausschluss VI



Exkurs: Mehrkanalspeicher mit Semaphore



Exkurs: Semaphorelogik des 7016