Übungen Betriebssysteme (BS)

U2 – Thread-Synchronisation

Alwin Berger

TU Dortmund - AG Systemsoftware

technische universität dortmund

Arbeitsgruppe Systemsoftware

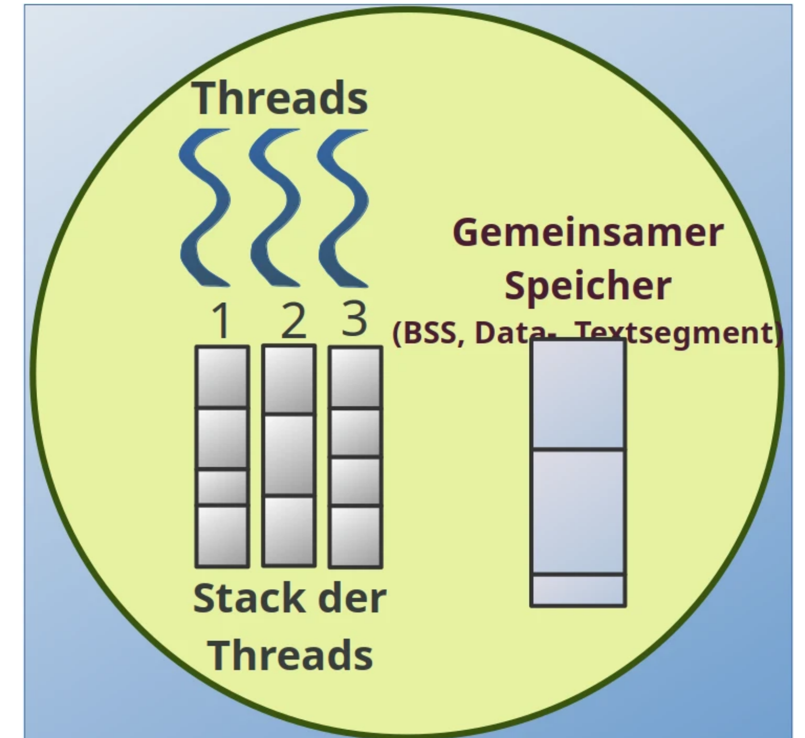
by Decker

Agenda

- Besprechung von „A1 - Prozesse verwalten“
- Fortsetzung der C-Einführung
- Aufgabe 2: Thread-Synchronisation
  - UNIX-Prozesse vs. POSIX-Threads
  - Vergleich: `exec*()`, `fork()` und `pthread_create`
  - Funktionen von `pthreads`
  - Mutex

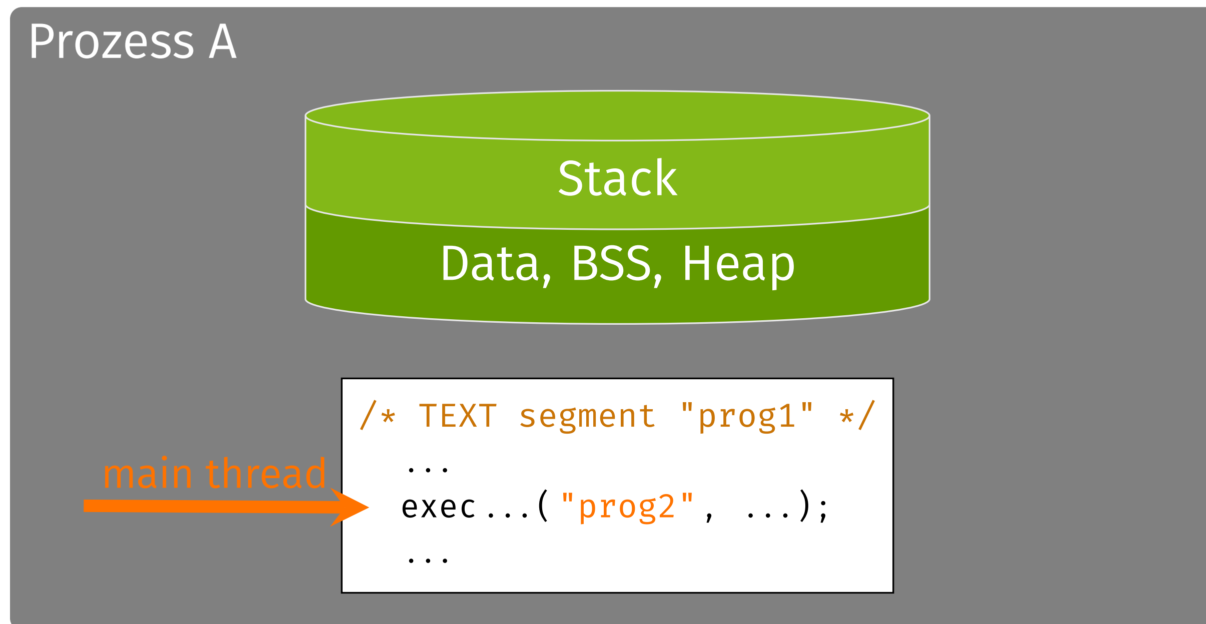
UNIX-Prozesse vs. POSIX-Threads

- **UNIX-Prozesse:** schwergewichtig, haben einen eigenen Adressraum
- **POSIX-Threads:** leichtgewichtig (**pthreads**)
  - ein Prozess kann mehrere Threads haben (teilen sich den gleichen Adressraum)
  - pthreads bieten standardisierte Schnittstelle
  - pthreads verwenden intern Systemaufrufe
  - jeder pthread hat eine eigene ID
  - (Typ: pthread: unsigned long int)



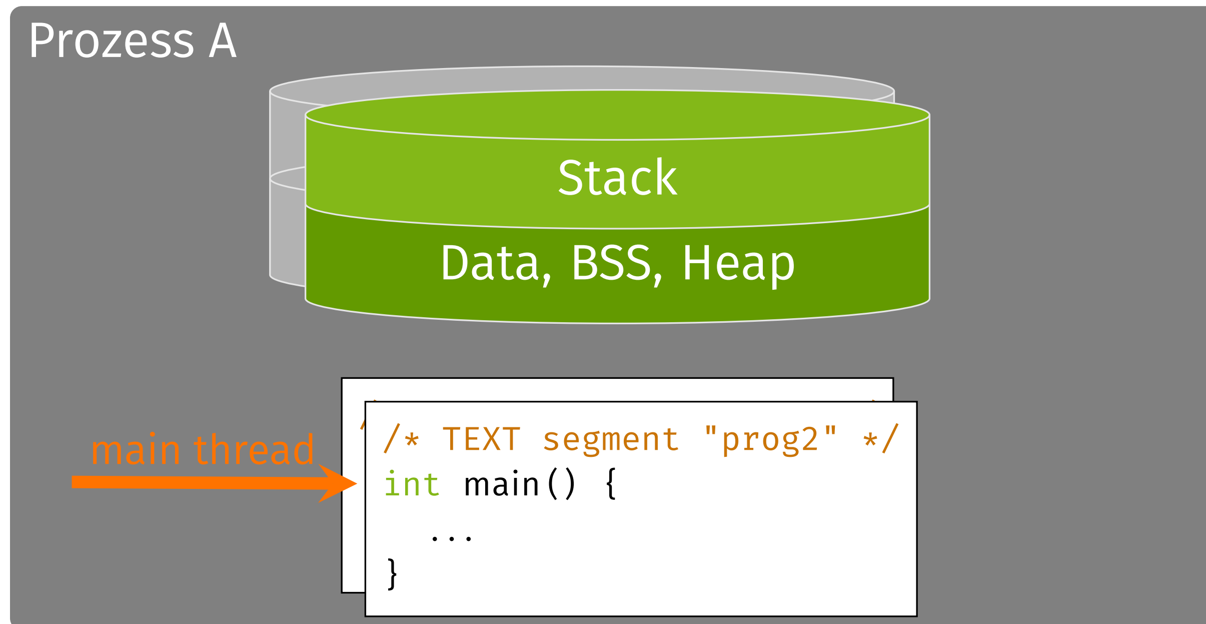
Vergleich: *exec*()*, fork() und pthread_create()

- Überlagerung eines Prozesses
- **Keine** gemeinsamen Daten
- schwergewichtig



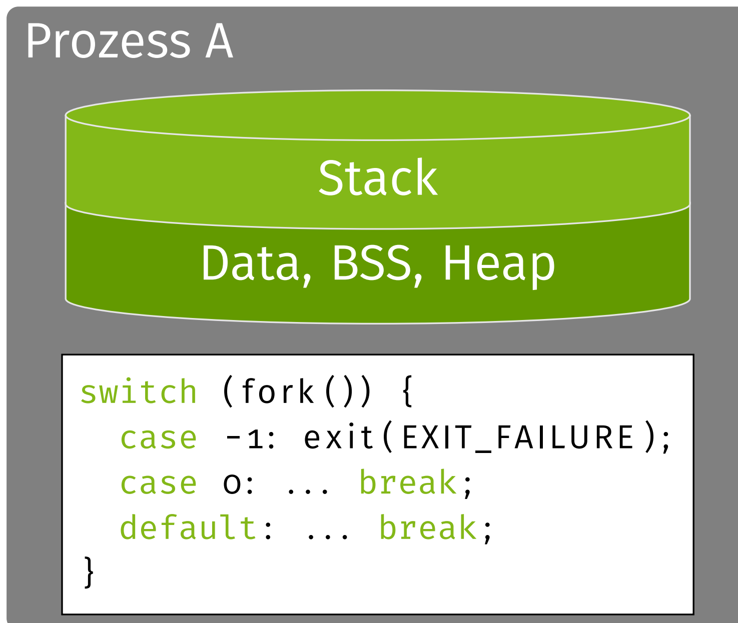
Vergleich: *exec*()*, fork() und pthread_create()

- Überlagerung eines Prozesses
- **Keine** gemeinsamen Daten
- schwergewichtig



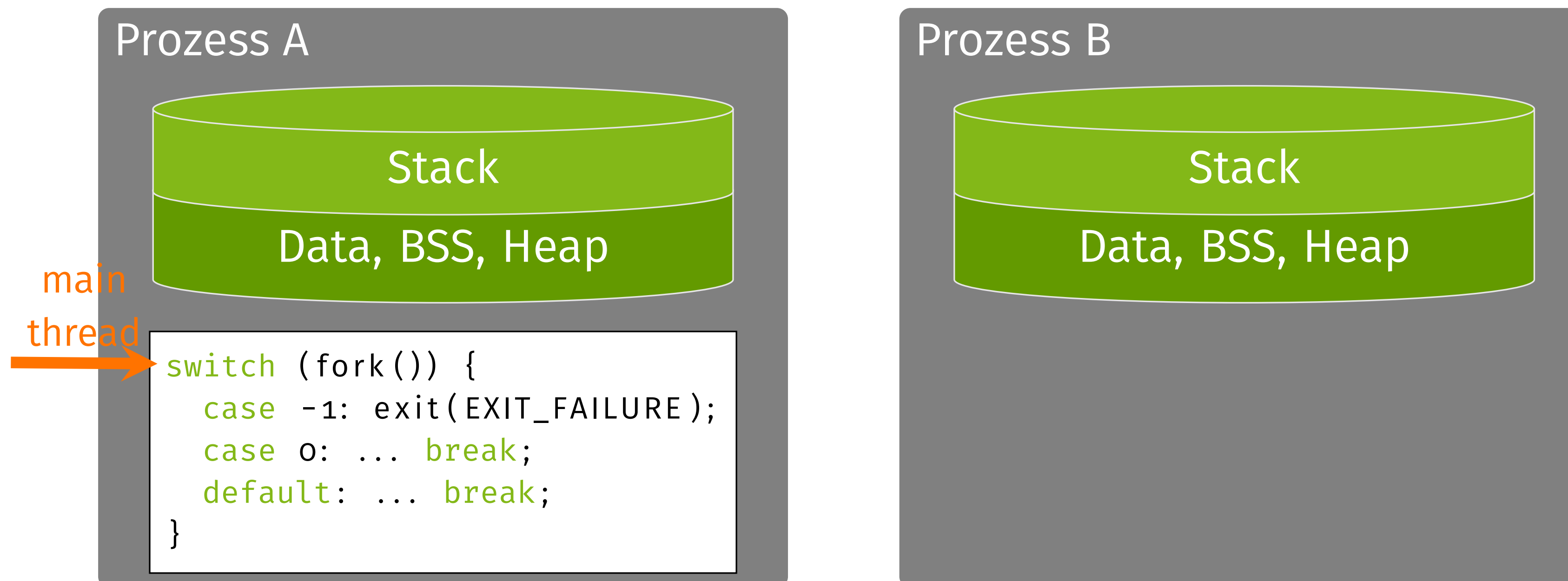
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



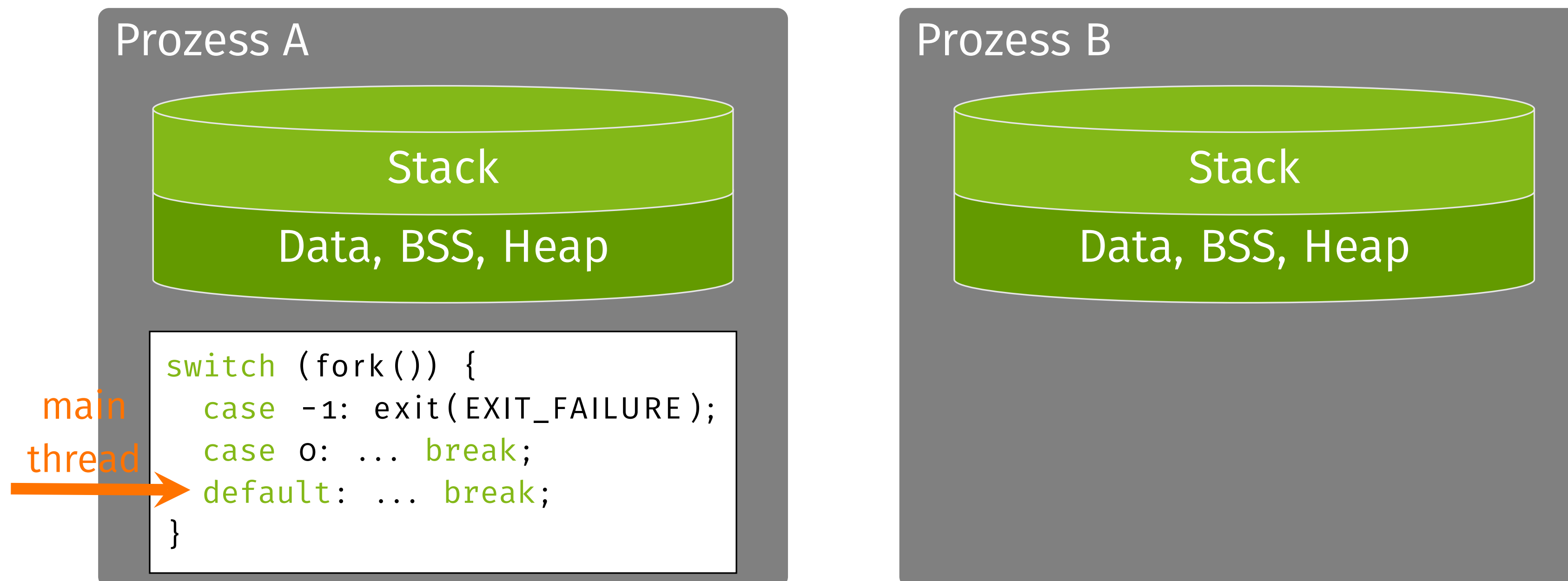
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



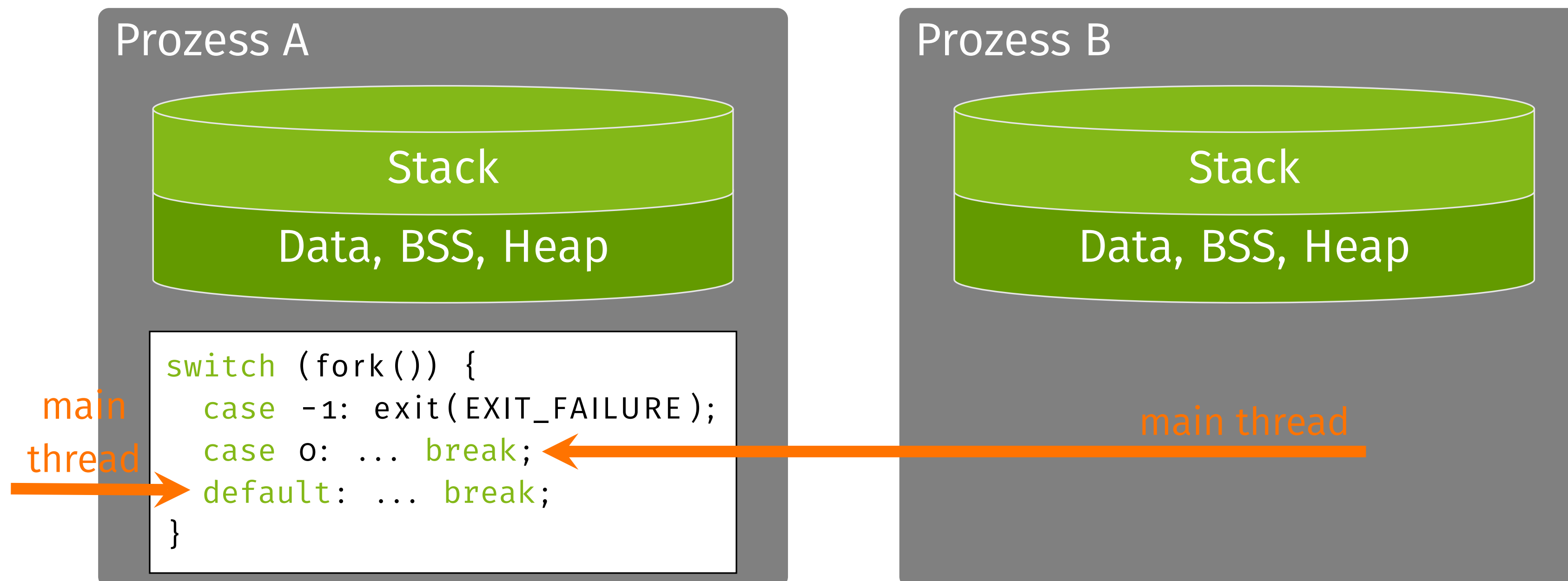
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



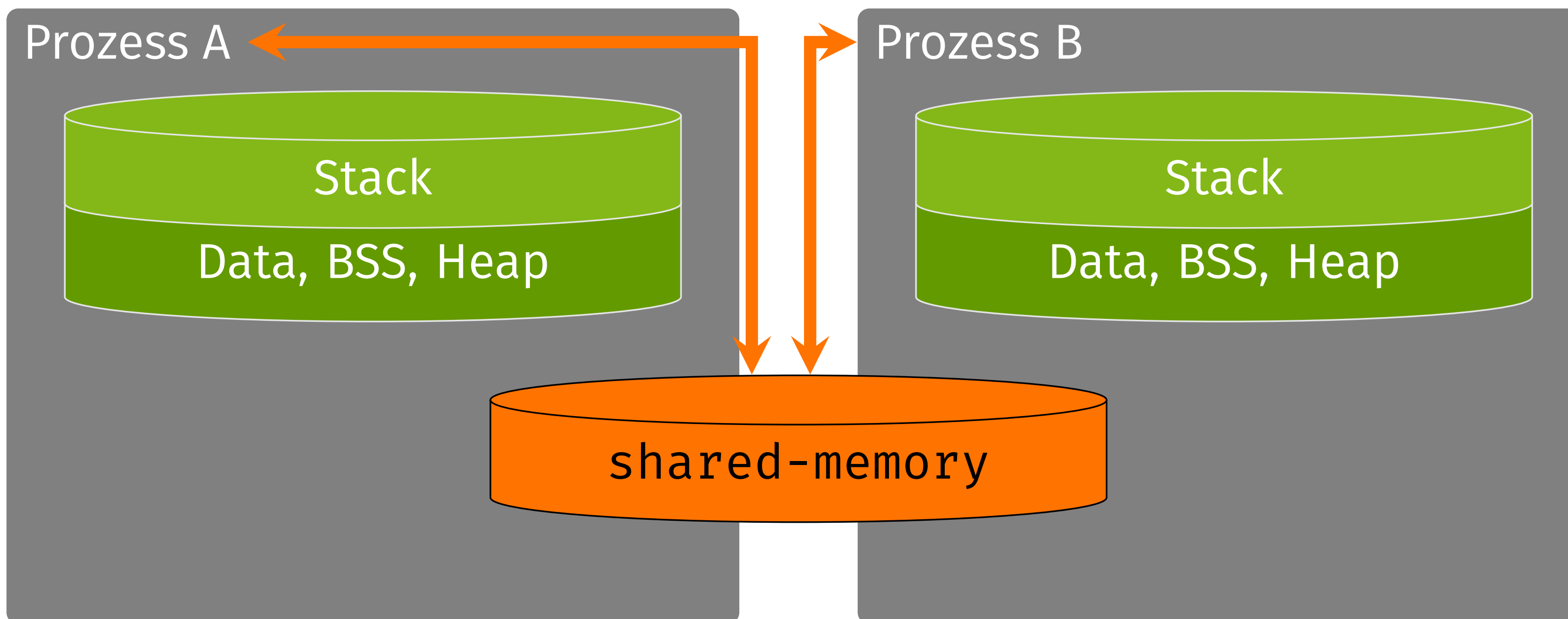
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



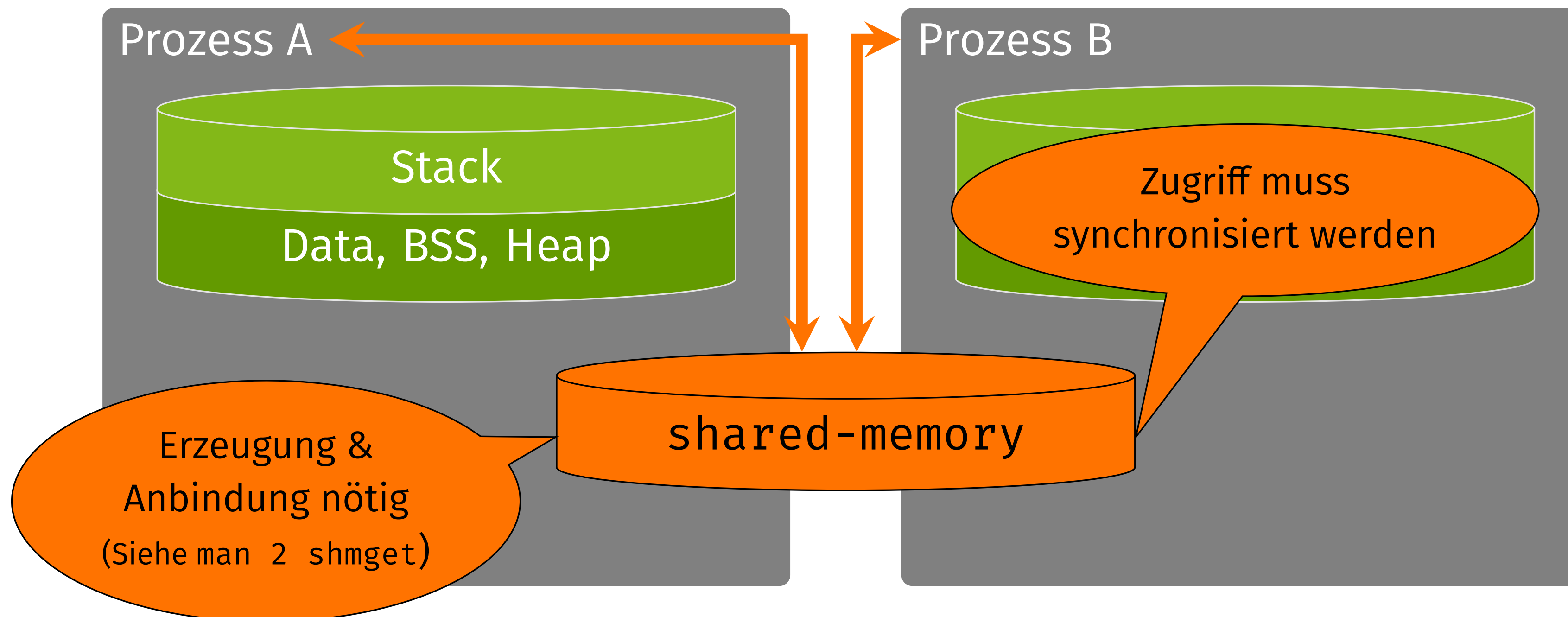
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



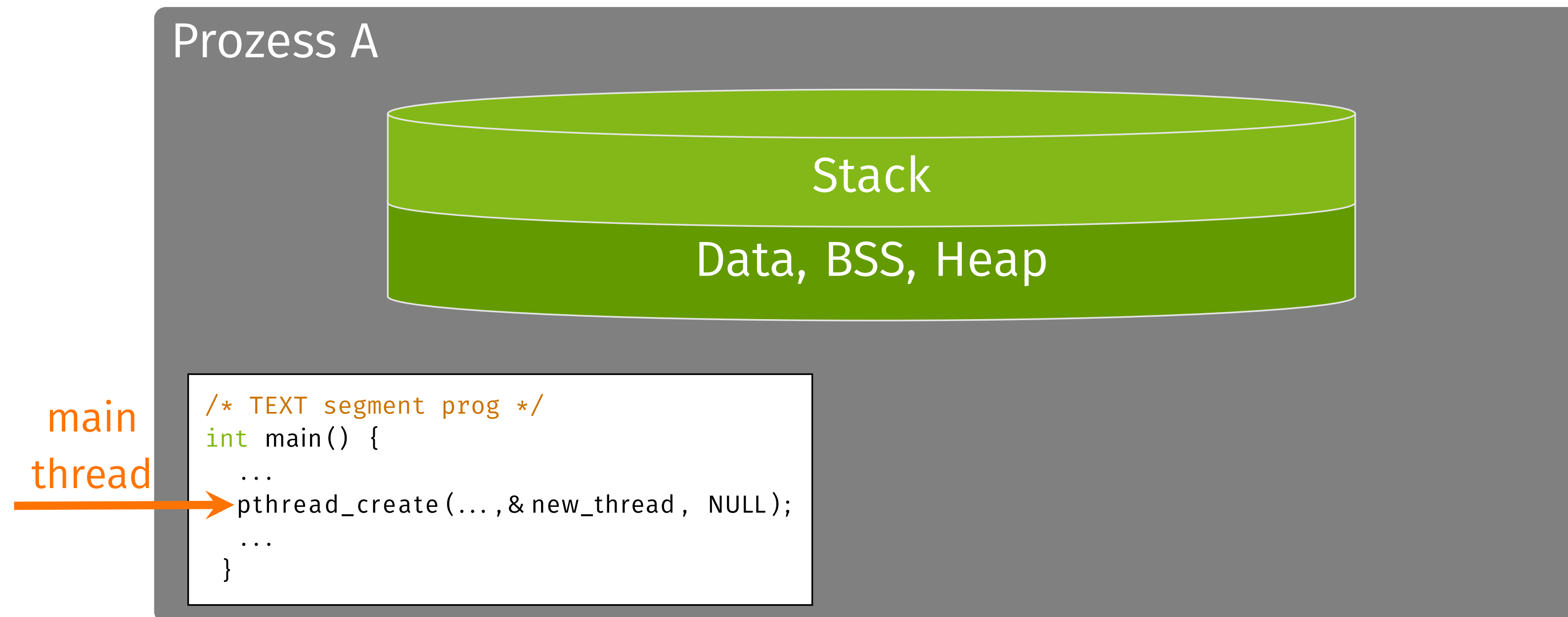
Vergleich: exec*(), *fork()* und pthread_create()

- Verzweigung eines Prozesses
- Gemeinsame Daten über `shared-memory`
- schwergewichtig



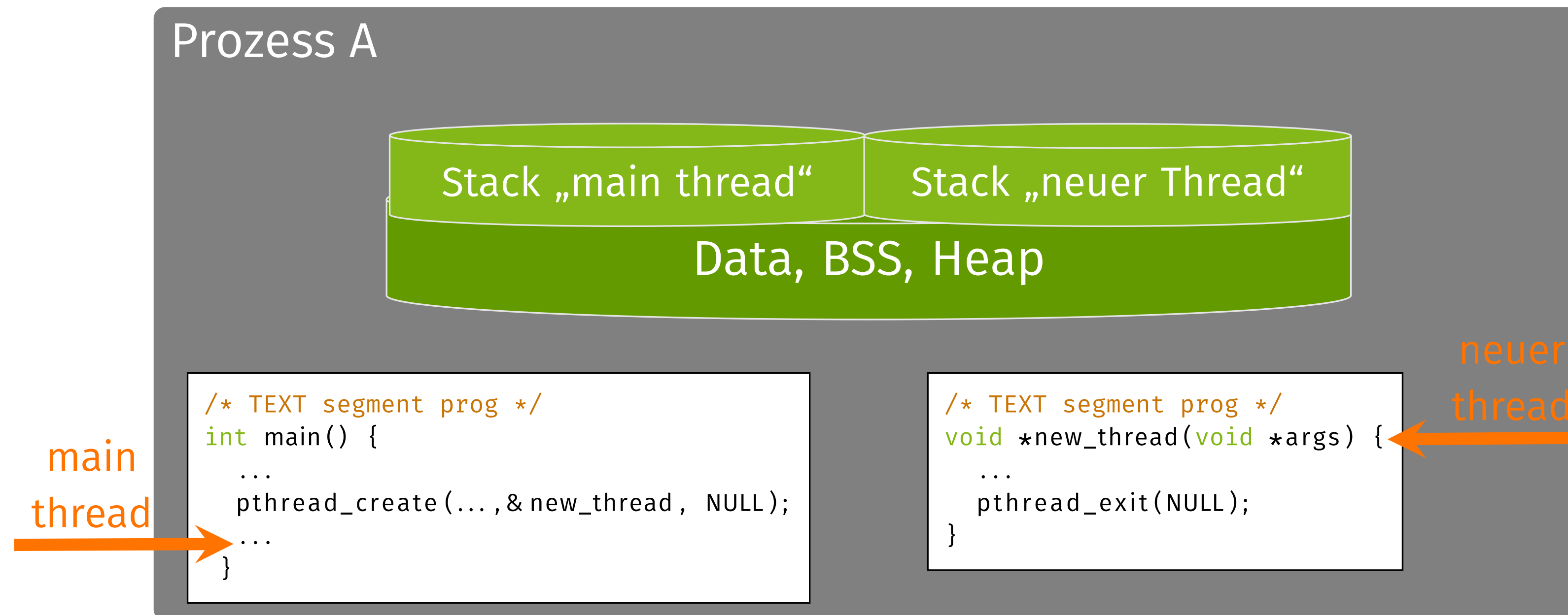
Vergleich: exec*(), fork() und *pthread_create()*

- Aufteilung eines Prozesses
- Gemeinsame Daten: `data`, `BSS`, `heap`, `shared-memory`
- leichtgewichtig



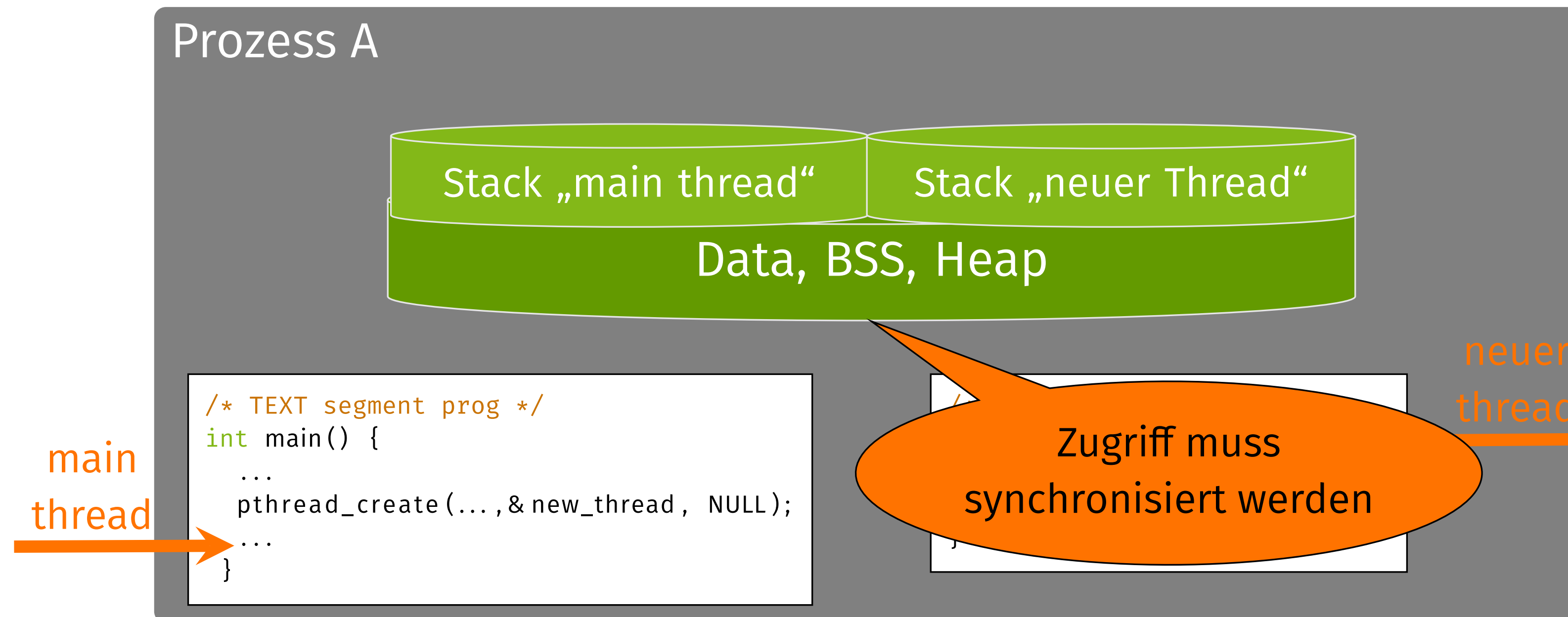
Vergleich: exec*(), fork() und *pthread_create()*

- Aufteilung eines Prozesses
- Gemeinsame Daten: `data`, BSS, `heap`, `shared-memory`
- leichtgewichtig



Vergleich: exec*(), fork() und *pthread_create()*

- Aufteilung eines Prozesses
- Gemeinsame Daten: `data`, BSS, `heap`, `shared-memory`
- leichtgewichtig



pthread-Beispiel

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

void* Hello(void *arg) {
  printf("Hello! Its me, thread!");
  pthread_exit(NULL); // oder: return NULL;
}

int main(void) {
  int status;
  pthread_t thread;

  status = pthread_create(&thread, NULL, &Hello, NULL);
  if (status) { /* Fehlerbehandlung */ }

  status = pthread_join(thread, NULL);
  if (status) { /* Fehlerbehandlung */ }

  pthread_exit(NULL); // Da wegen pthread_join() nur noch
}                     // ein Thread läuft, genügt return 0;
```

pthread_create(3)

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
      void *(*start_routine)(void *), void * arg);
```

- Erstellt einen neuen Thread
- Argumente
  - **thread:** Hier wird eine ID des Threads abgelegt
  - **attr:** Optionale Attribute (hier: NULL)
  - **start_routine:** Zeiger auf die auszuführende Funktion mit Signatur `void *func(void *arg)`
  - **arg:** Zeiger auf das Argument, welches start_routine übergeben wird
- Rückgabewerte
  - =0, wenn erfolgreich
  - !=0, wenn ein Fehler aufgetreten ist

pthread_exit(3)

```
void pthread_exit(void *retval);
```

- Beendet den aufrufenden Thread
- Argumente
  - **retval:** Gibt den exit-Status an, der zurückgeliefert werden soll. Darf NULL sein.
- Anmerkungen
  - `void*` stellvertretend für alle möglichen Zeiger. NULL ist gleichbedeutend mit `(void*)0`

pthread_join(3)

```
int pthread_join(pthread_t thread, void **retval);
```

- Legt den aufrufenden Thread schlafen, bis der Thread mit ID thread terminiert (ähnlich zu wait(2))
- Argumente
  - **thread:** Thread-ID, auf dessen Terminierung gewarten werden soll
  - **reval:** Ein Zeiger auf eine Variable, in die der exit-Status des Threads abgelegt werden soll. (Für uns nicht relevant. Darf daher NULL sein.)
- Rückgabewerte
  - =0, wenn erfolgreich
  - !=0, wenn ein Fehler aufgetreten ist

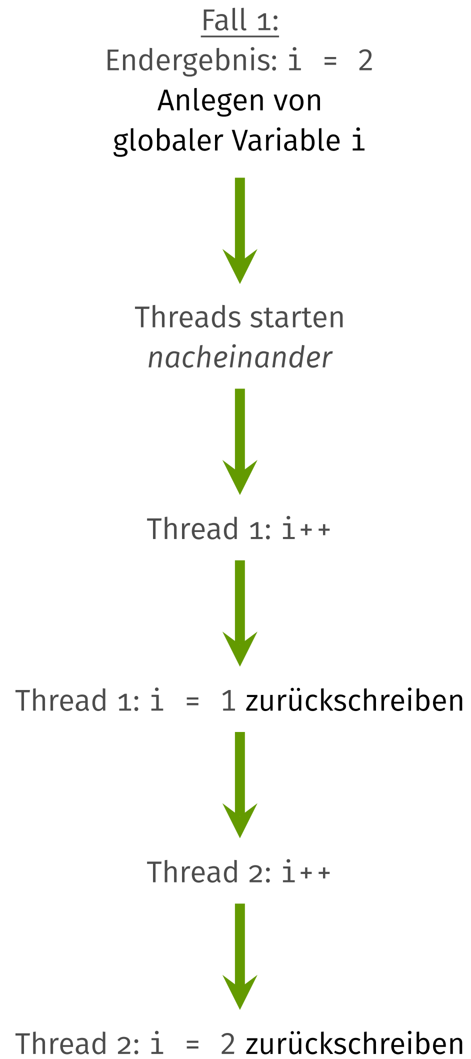
pthread_self(3)

- Liefert die ID des aufrufenden Threads zurück
- Der Rückgabewert entspricht dem Inhalt der Variable thread, auf die ein Zeiger bei `pthread_create(&thread, ...);` übergeben wurde.
- Anmerkungen
  - Vergleich zweier Thread-IDs mittels
    `int pthread_equal(pthread_t t1, pthread_t t2);` möglich.
  - Gibt einen Werte verschieden von 0 bei Gleichheit zurück.

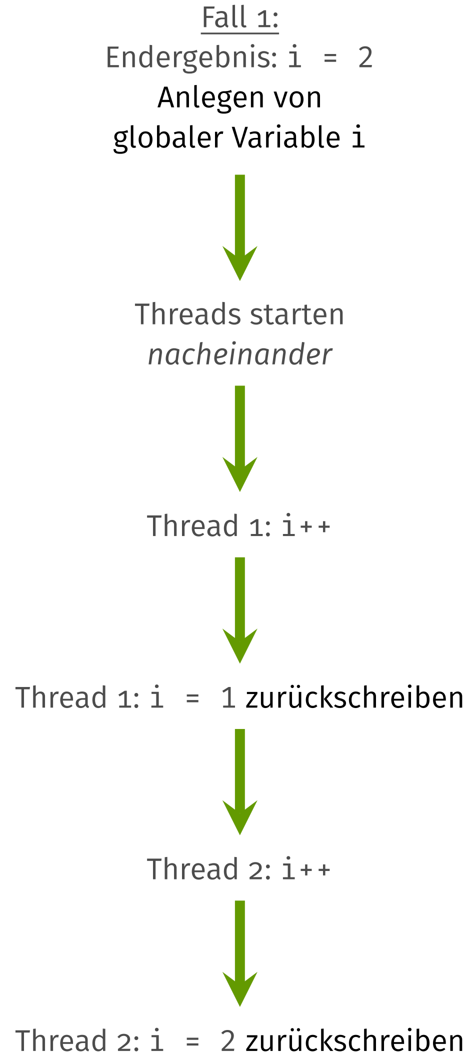
Was wird passieren?

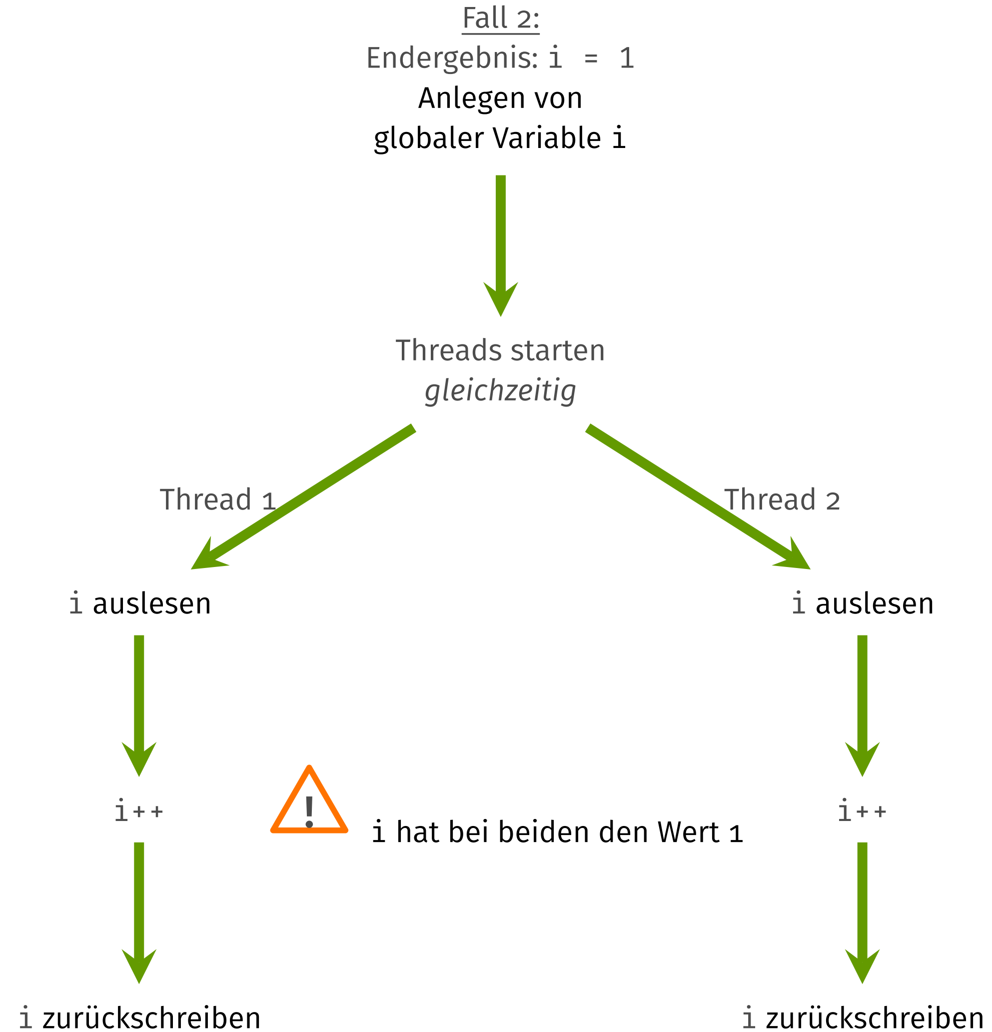
```c
#include <stdio.h>

/* globale Variable */
int i = 0;

/* Funktion von Thread 1 */
void *f1(void *arg) {
  i++;
  return NULL;
}

/* Funktion von Thread 2 */
void *f2(void *arg) {
  i++;
  return NULL;
}
```

Wettlaufsituation (Race Condition)

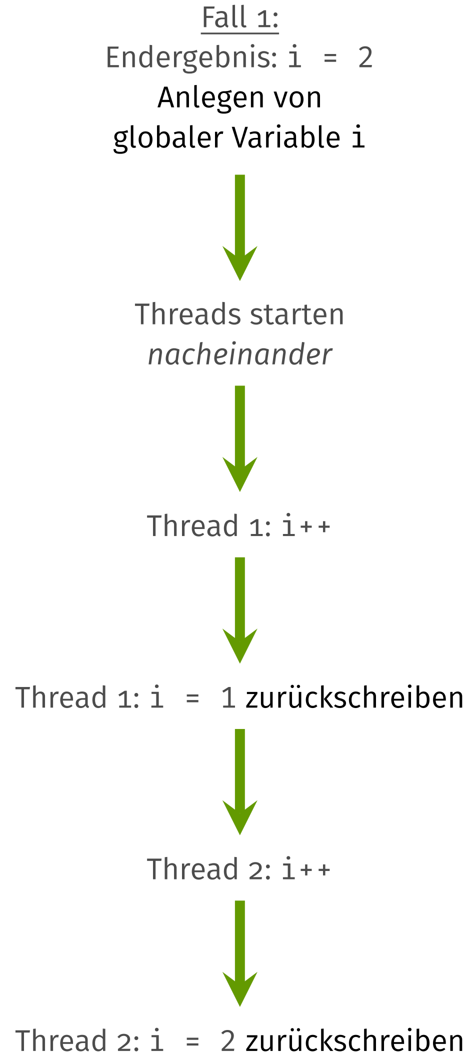


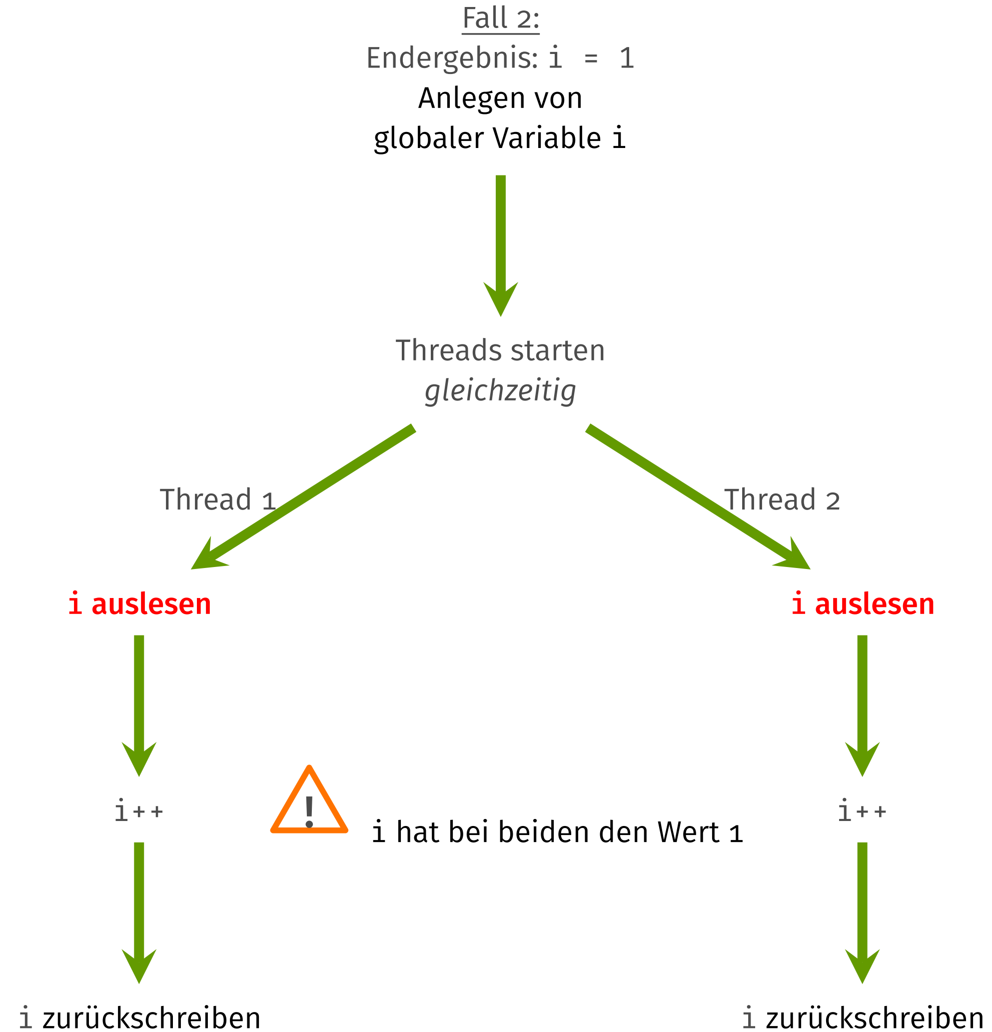
Wettlaufsituation (Race Condition)



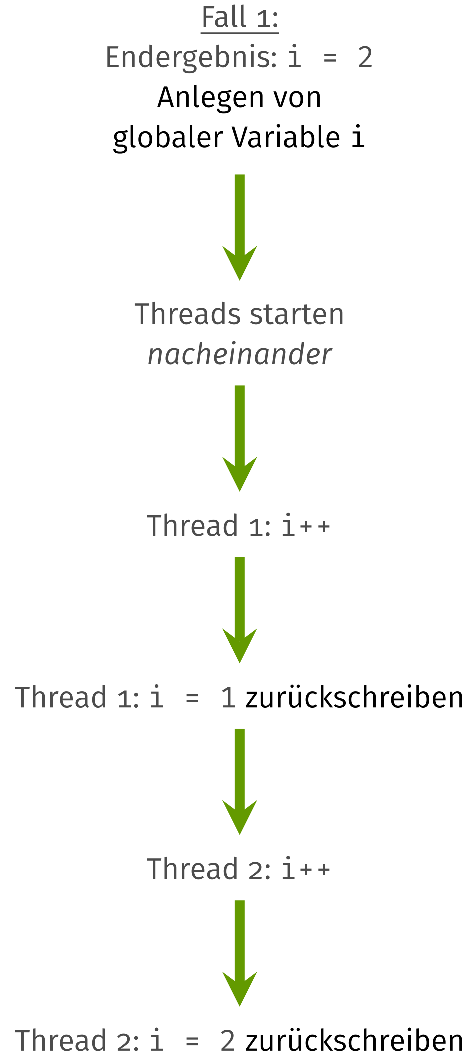


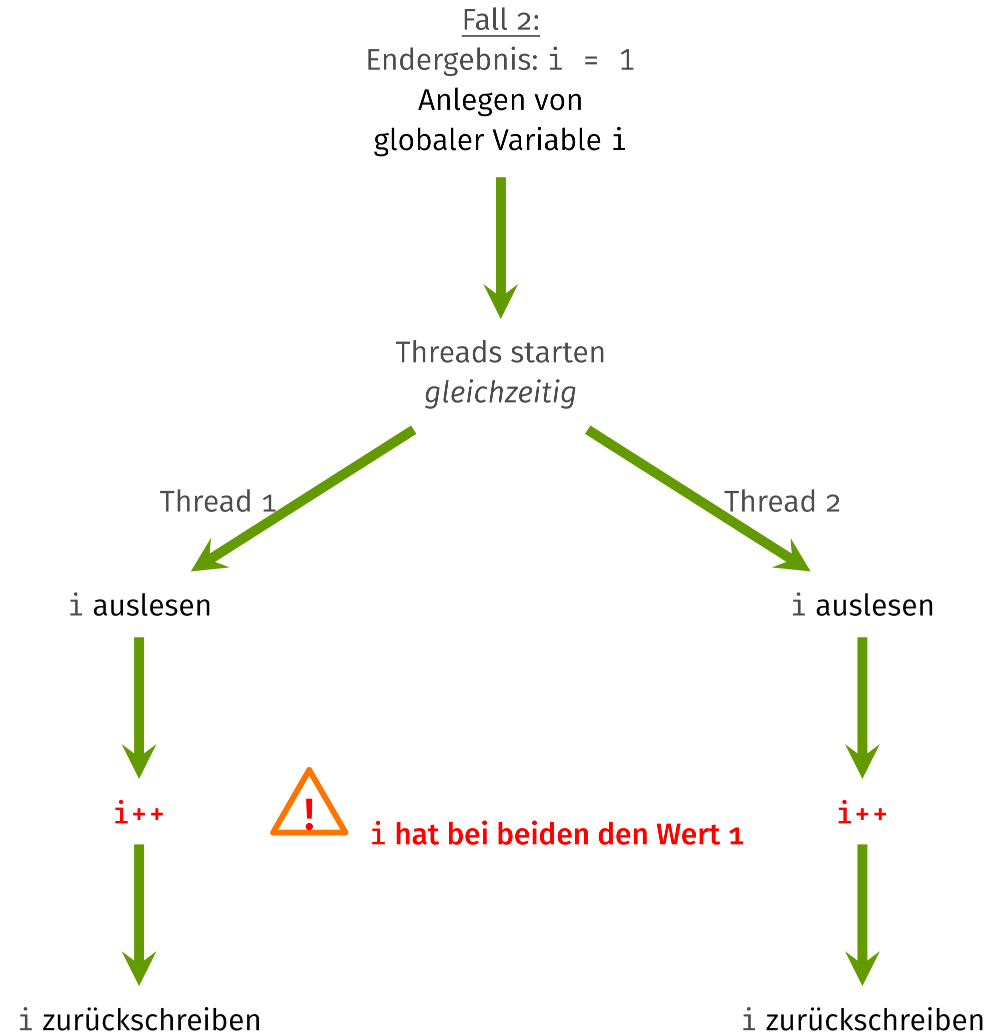
Wettlaufsituation (Race Condition)



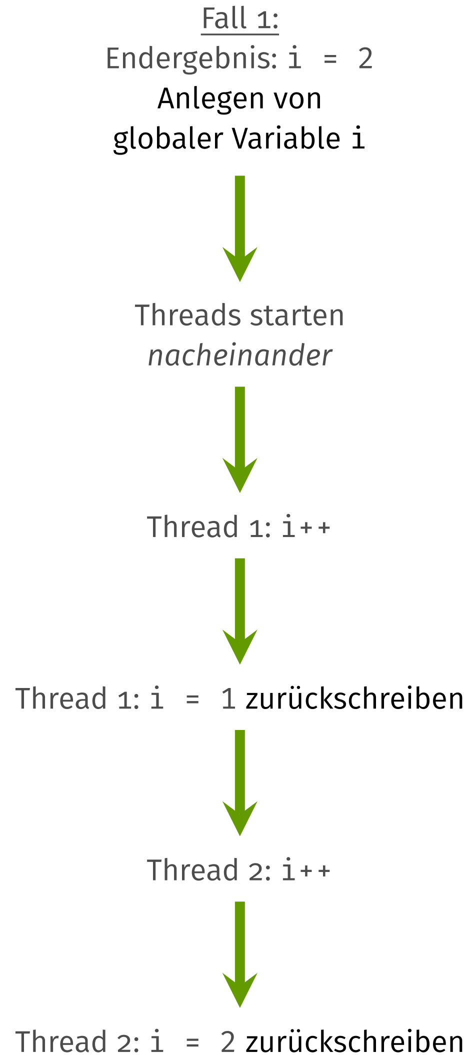


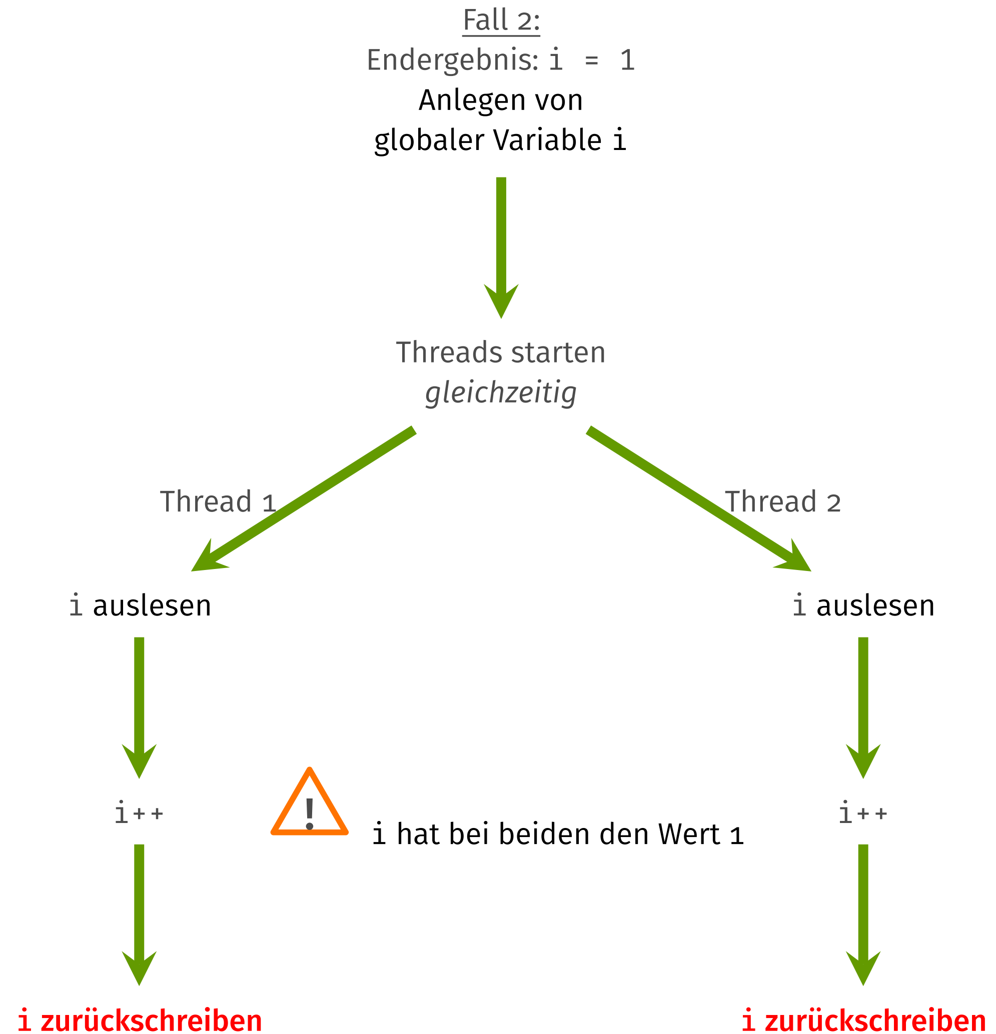
Wettlaufsituation (Race Condition)



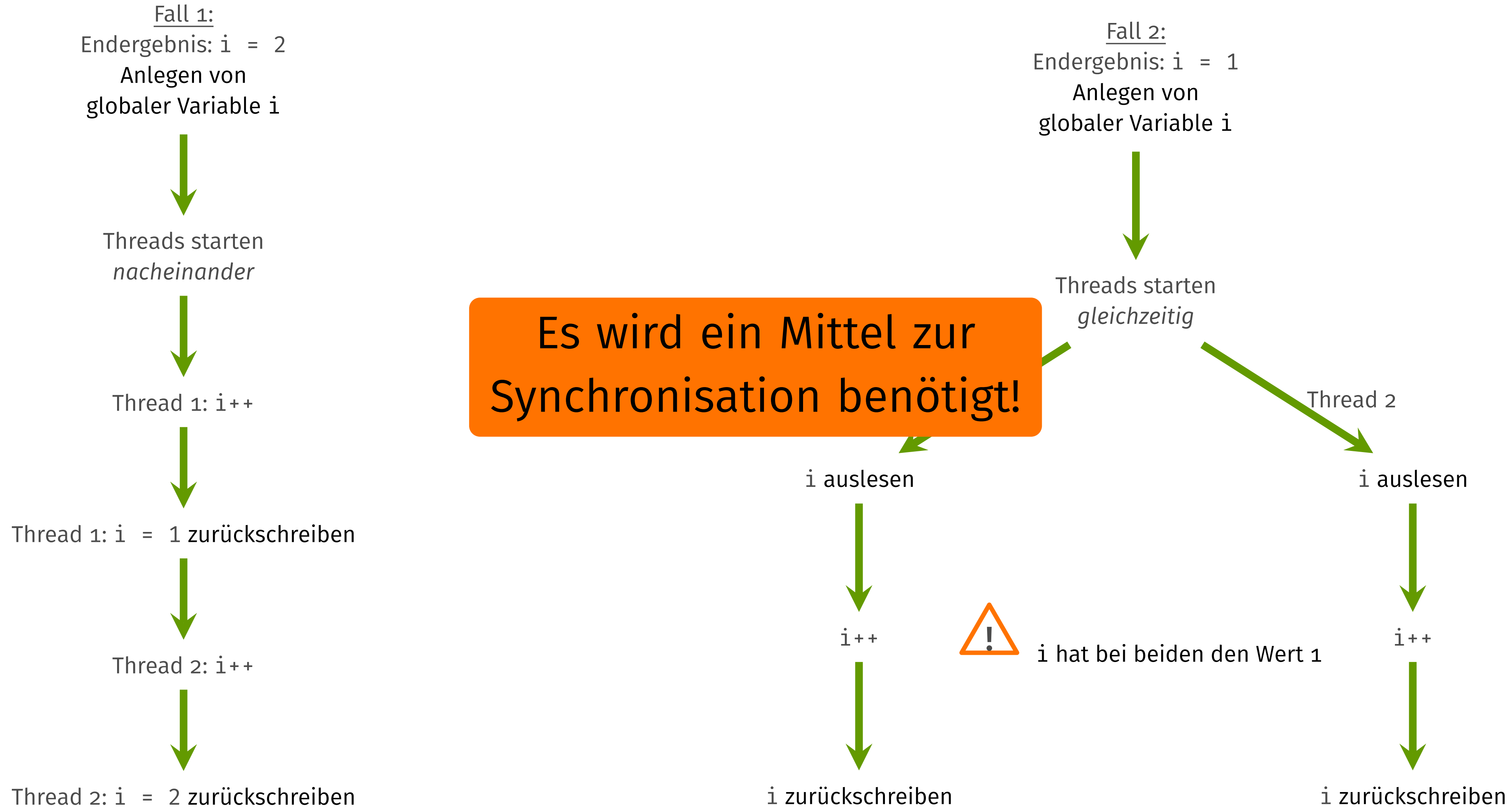


Wettlaufsituation (Race Condition)





Wettlaufsituation (Race Condition)



Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.

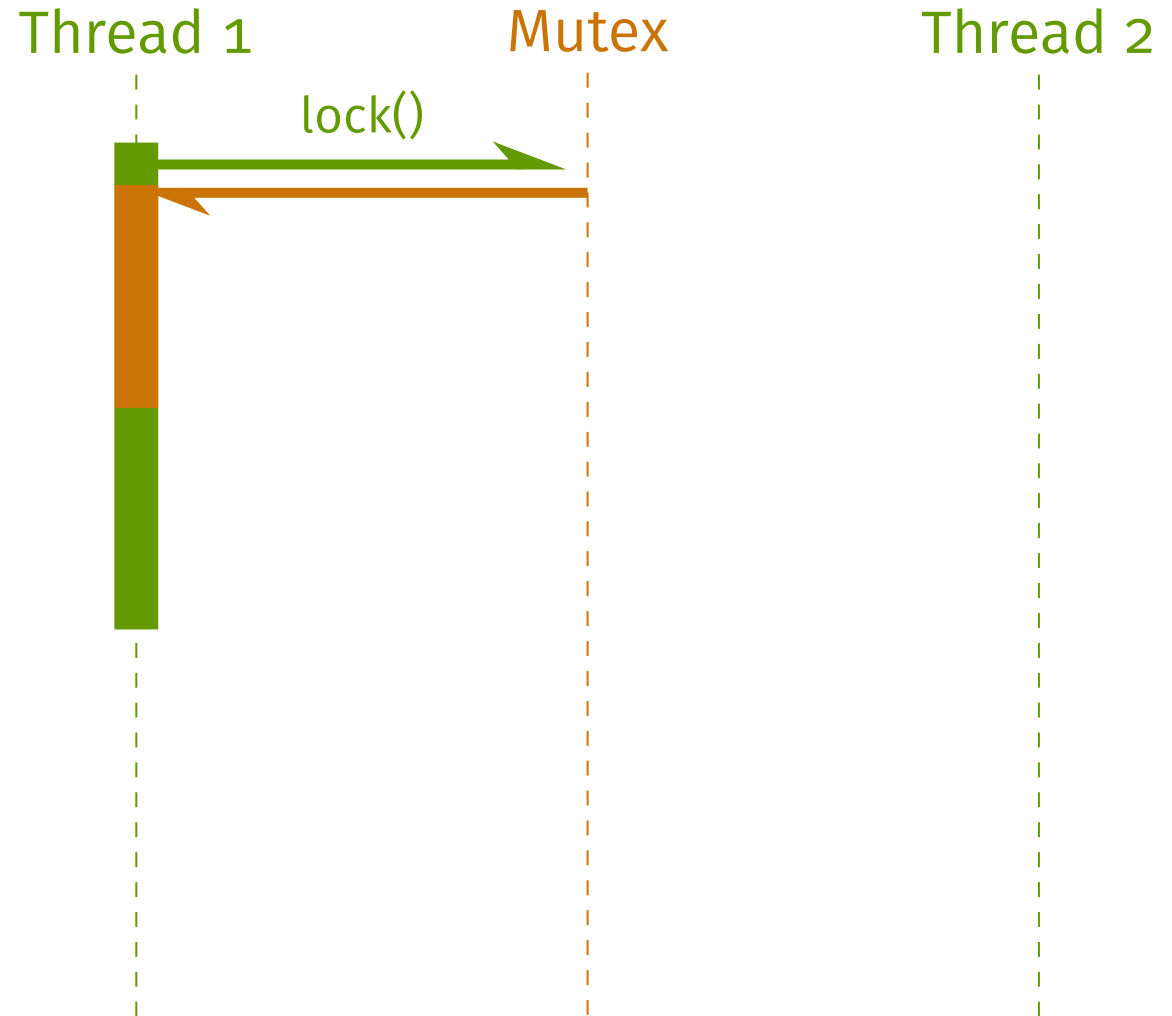
Thread 1

Mutex

Thread 2

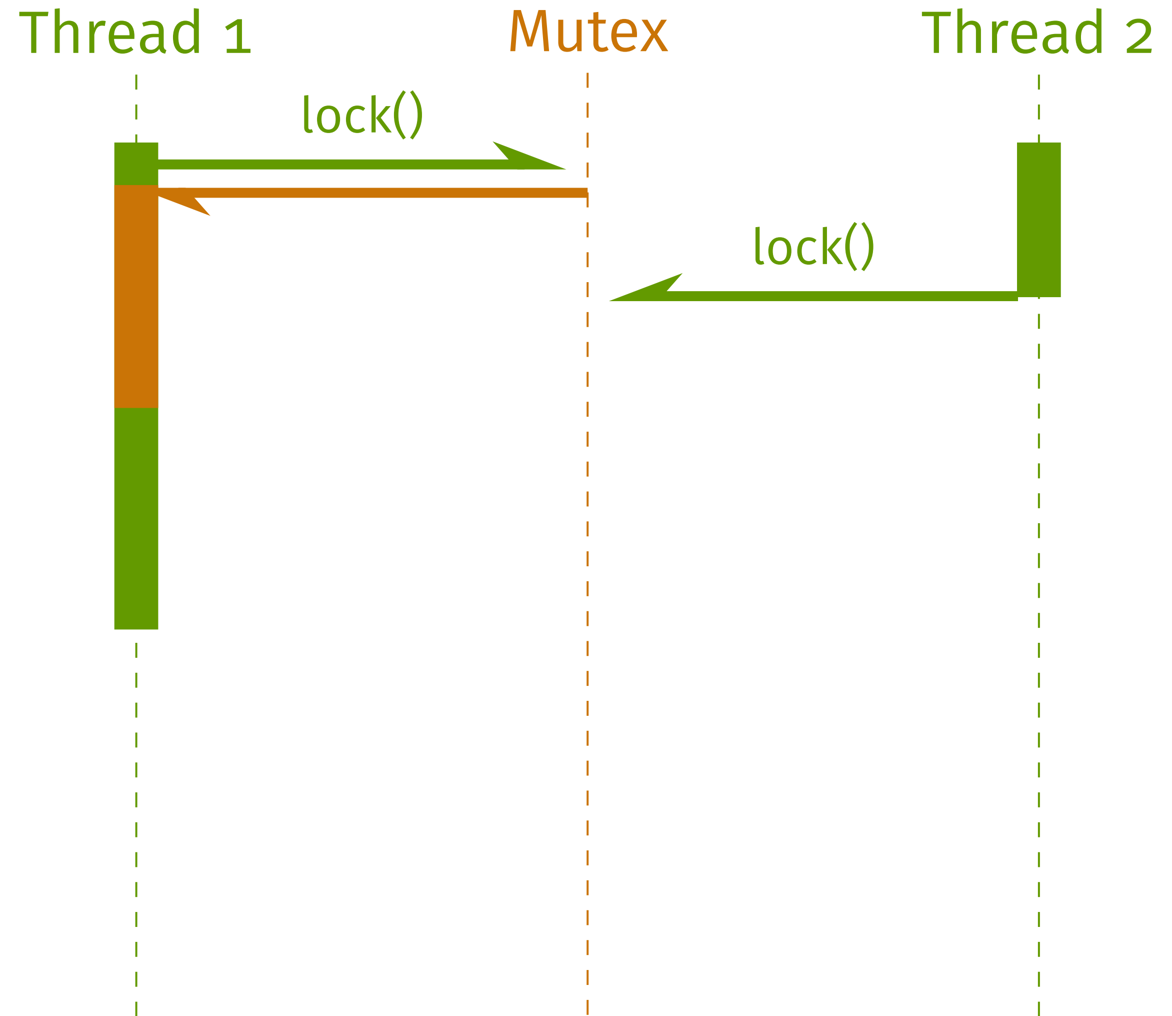
Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



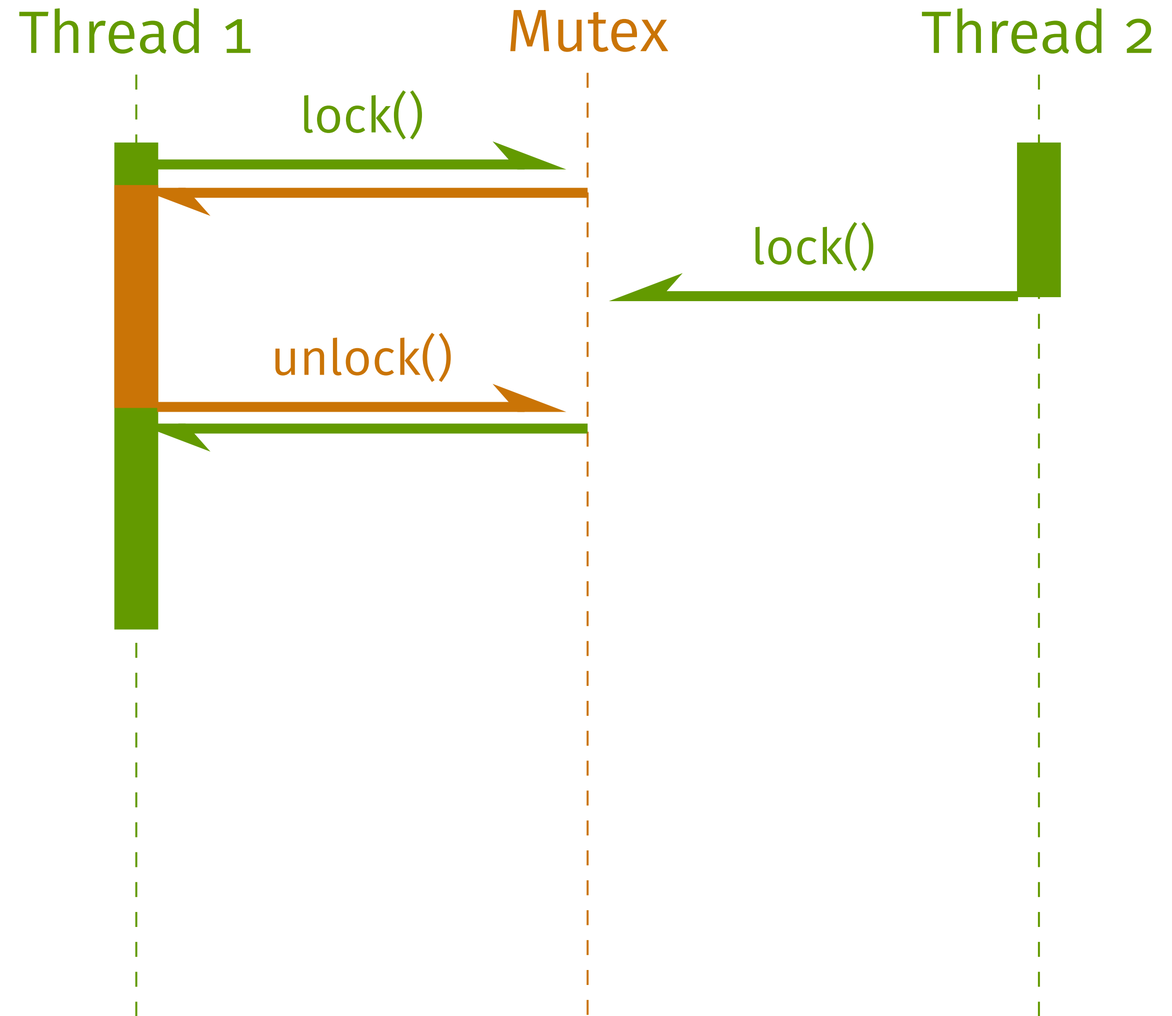
Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



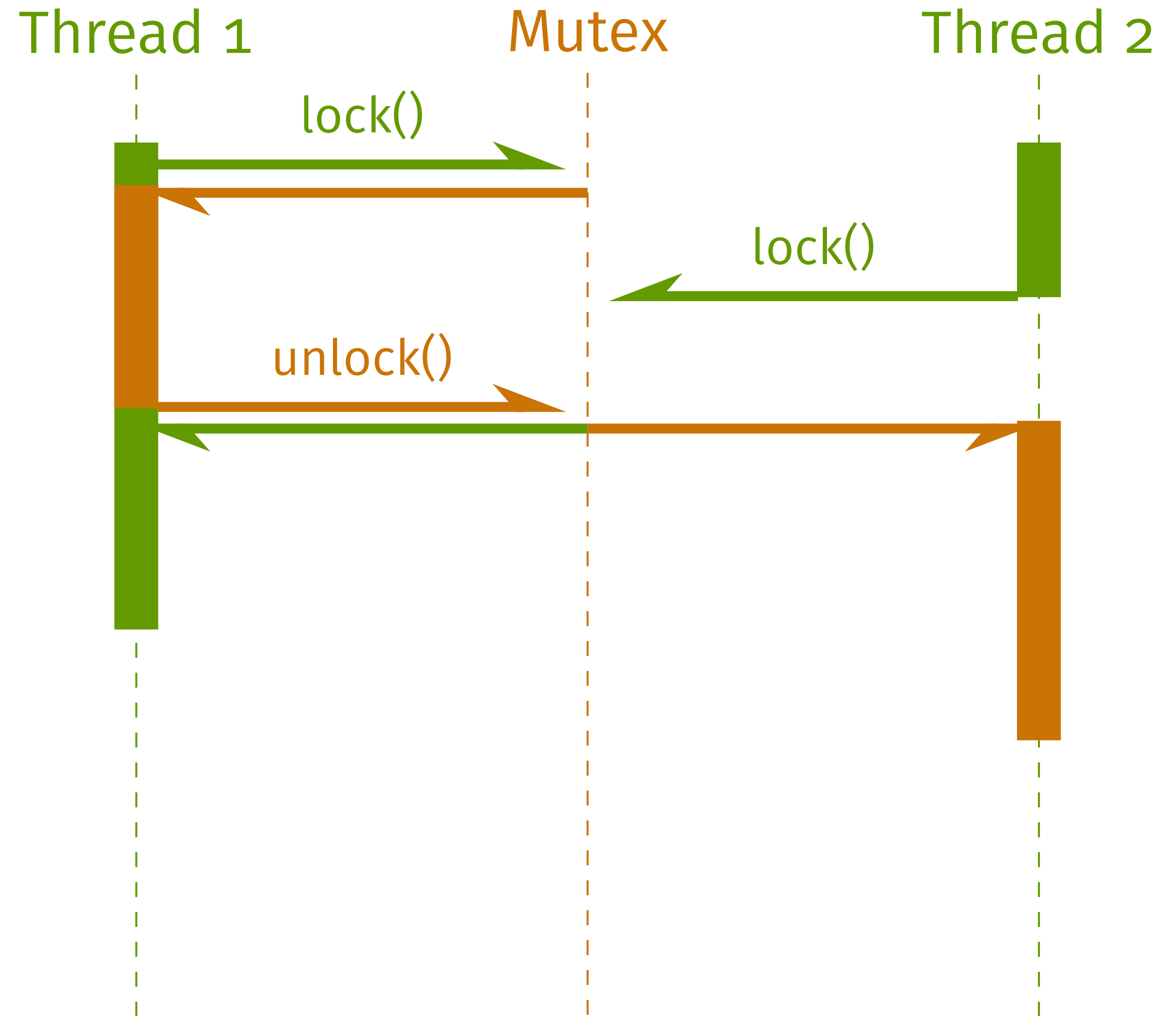
Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



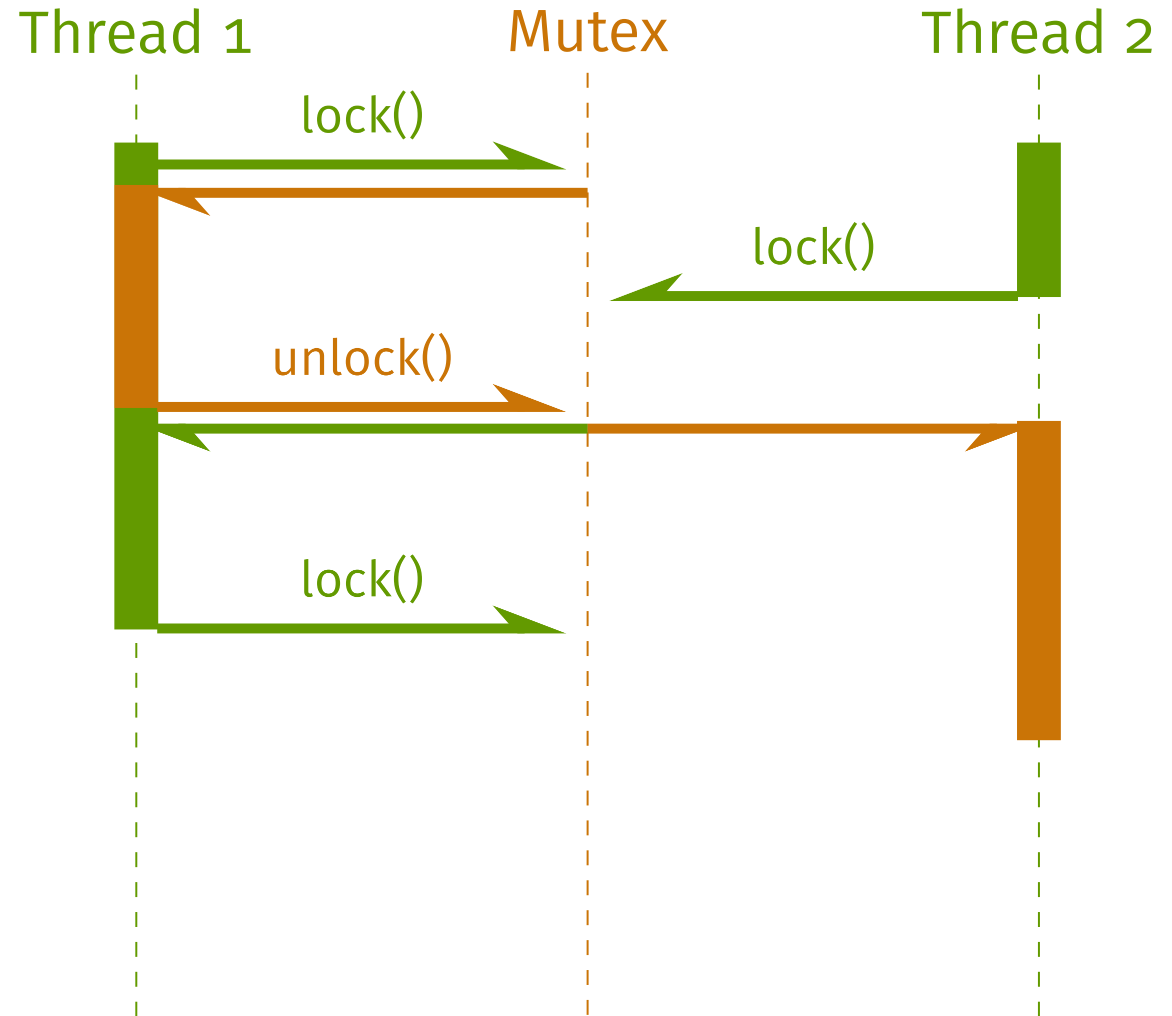
Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



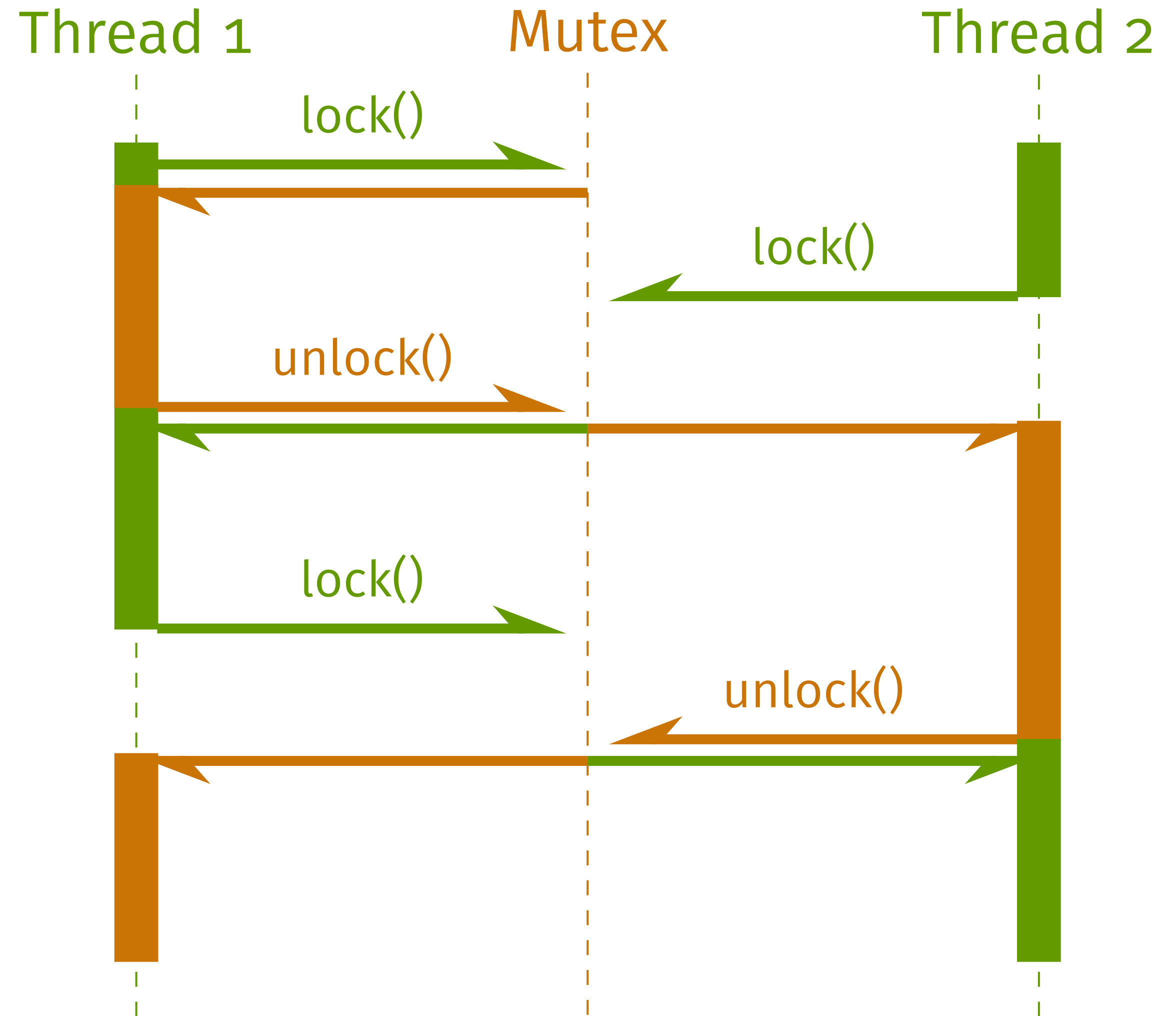
Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



Mutex

- Steht für **Mutual exclusion**
- Objekt zum Erzwingen von gegenseitigem Ausschluss
- Muss vom gleichen Thread freigegeben werden, der auch den Mutex gesperrt hat.



pthread_mutex_init(3)

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
```

- Initialisiert den Mutex mit dem Zusatnd *nicht gesperrt*
- Argumente
    - **mutex:** Ein Zeiger auf den zu initialisierenden Mutex
    - **attr:** Ein Zeiger auf zusätzliche Attribute (meistens NULL)
- Rückgabewerte
    - =0, wenn erfolgreich
    - !=0, wenn ein Fehler aufgetreten ist
- Gegenstück:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

pthread_mutex_lock(3)

```
int   pthread_mutex_lock(pthread_mutex_t *mutex);
```

- Betritt und sperrt den kritischen Abschnitt
- Legt den aufrufenden Thread schlafen, wenn bereits belegt
- Argumente
  - **mutex:** Ein Zeiger auf den zu verwendenden Mutex
- Rückgabewerte
  - =0, wenn erfolgreich
  - !=0, wenn ein Fehler aufgetreten ist
- Gegenstück:
  `int pthread_mutex_unlock(pthread_mutex_t *mutex);`

Mutex-Beispiel

```c
#include <stdio.h>

/* globale Variablen */
int i = 0 ;
pthread_mutex_t pferd;

int main ( . . . ) {
  pthread_mutex_init(&pferd, NULL);
  /* Erstelle zwei Threads */
  /* und warte auf deren Ende */
  pthread_mutex_destroy(&pferd);
}
```

```c
/* Funktion von Thread 1 */
void * f1(void * arg ) {
  pthread_mutex_lock(&pferd);
  i++;
  pthread_mutex_unlock(&pferd);
  return NULL ;
}

/* Funktion von Thread 2 */
void * f2(void * arg ) {
  pthread_mutex_lock(&pferd);
  i++;
  pthread_mutex_unlock(&pferd);
  return NULL ;
}
```

Was haben wir heute gelernt?

- Es gibt verschiedene Arten von Fäden: leicht- vs. schwergewichtige
- Das Erzeugen von leichtgewichtigen Fäden in Linux (POSIX-Thread)
- Das Zustandekommen einer Wettlaufsitaution (Race Condition)
- Die Verwendung von Synchronisationmittel zur Vermeidung selbiger – siehe Mutex

Bei Fragen und Problemen

- HelpDesk findet montags 14-16 Uhr sowie dienstags und mittwochs 10-12 Uhr statt
- Via Matrix #bs-helpdesk:fachschaften.org
- Via E-Mail bs-problems@ls12.cs.tu-dortmund.de