Übung zu Betriebssysteme

Threadumschaltung

5. & 6. Dezember 2019

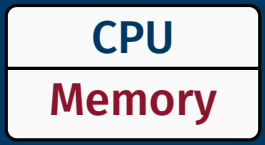
Andreas Ziegler, Bernhard Heinloth, Christian Eichler

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

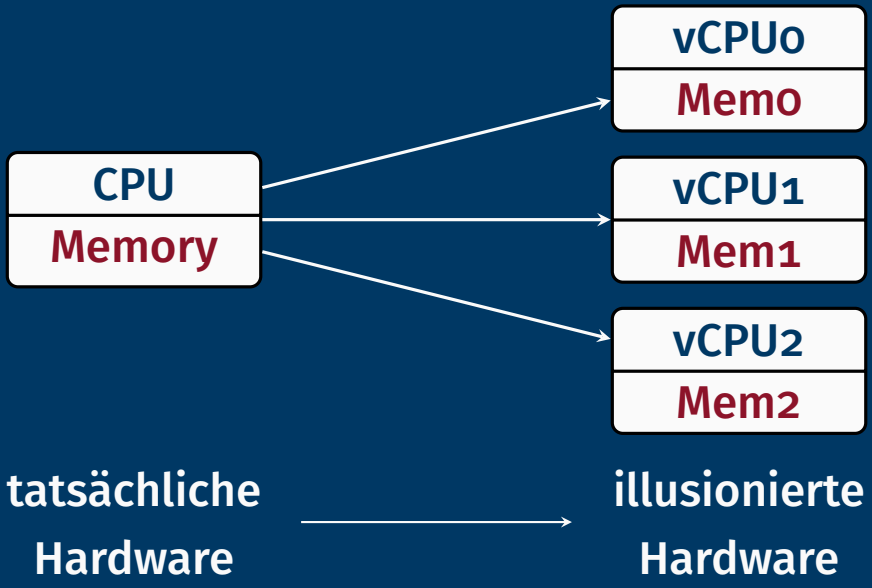
Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT



tatsächliche
Hardware


CPU
Memory
vCPU0
Mem0
vCPU1
Mem1
vCPU2
Mem2
tatsächliche Hardware
illusionierte Hardware

- Speicher virtualisieren
- CPU virtualiseren

- **Speicher virtualisieren ist einfach:**

```
#define MEMSIZE 100
char Mem0[MEMSIZE];
char Mem1[MEMSIZE];
char Mem2[MEMSIZE];
```

- **CPU virtualiseren**

- **Speicher virtualisieren ist einfach:**

```
#define MEMSIZE 100
char Mem0[MEMSIZE];
char Mem1[MEMSIZE];
char Mem2[MEMSIZE];
```

- **CPU virtualiseren**
  - nicht teilbar im Ort

- **Speicher virtualisieren ist einfach:**

```
#define MEMSIZE 100
char Mem0[MEMSIZE];
char Mem1[MEMSIZE];
char Mem2[MEMSIZE];
```

- **CPU virtualiseren**
  - nicht teilbar im Ort
  - aber teilbar in der Zeit

Koroutinen

Motivation: mehr Aktivitätsträger als CPUs

```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;

  }

}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;

  }

}
```

Einseitiger Aufruf

```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;

  }
  bar();
}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;

  }

}
```

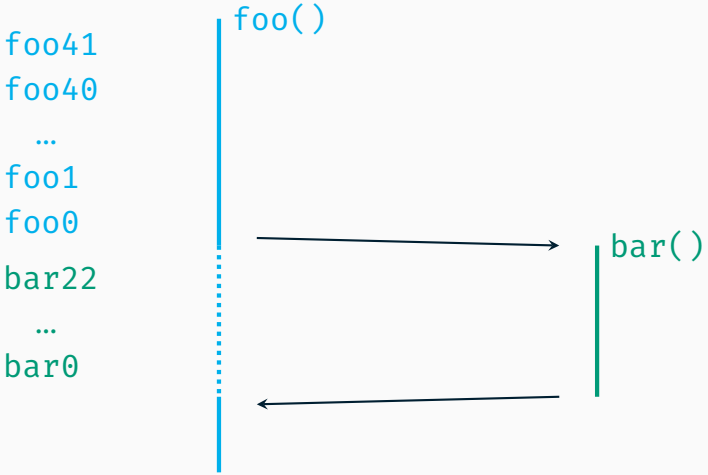
Einseitiger Aufruf

```
foo41
foo40
  …
foo1
foo0
```

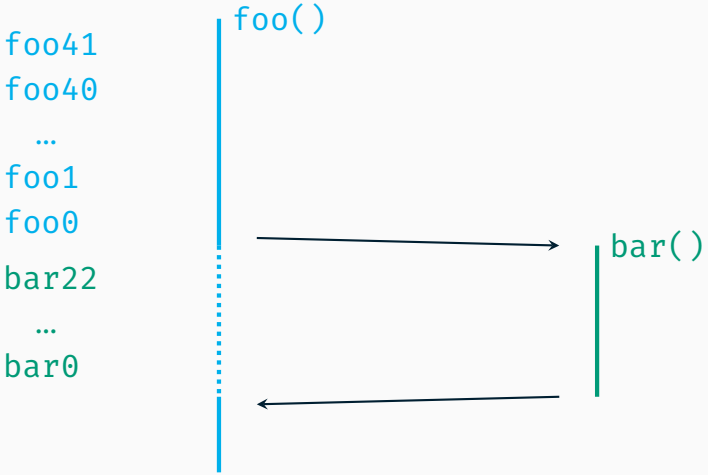
Einseitiger Aufruf

```
foo41
foo40
  …
foo1
foo0

bar22
  …
bar0
```

Einseitiger Aufruf



Einseitiger Aufruf



nicht parallel

Gegenseitiger Aufruf

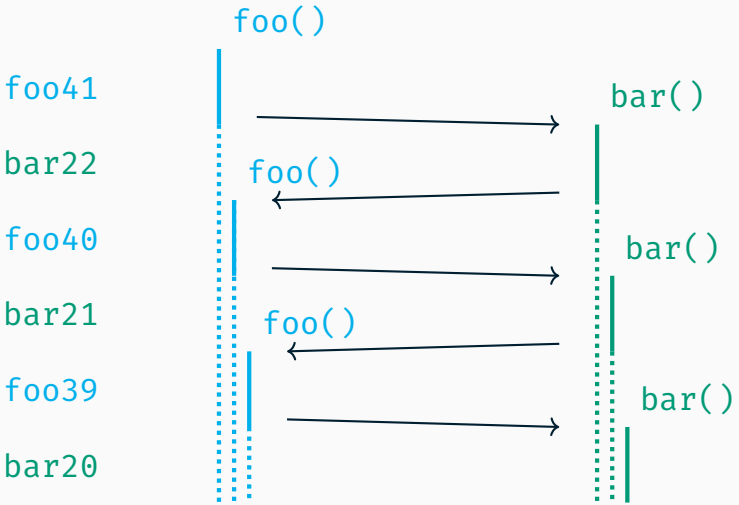
```
void foo(){
  static int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;
    bar();
  }

}
```

```
void bar(){
  static int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;
    foo();
  }

}
```

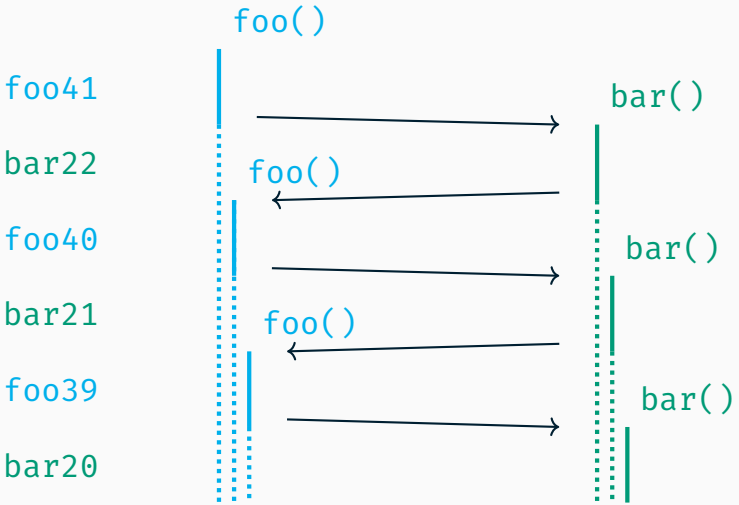
Gegenseitiger Aufruf

```
foo41

bar22

foo40

bar21

foo39

bar20
```

Gegenseitiger Aufruf



Gegenseitiger Aufruf



hoher Speicherverbrauch & (ggf.) Endlosrekursion

Umschaltung

Umschaltung

Kontrollflusszustand sichern & laden

Umschaltung

Kontrollflusszustand sichern & laden

- Stack
- Register

Umschaltung

Kontrollflusszustand sichern & laden

- Stack
- Register

} Kontext

Umschaltung

Kontrollflusszustand sichern & laden

- Stack
- Register

} Kontext

- Umschaltefunktion

Umschaltung

Kontrollflusszustand sichern & laden

- Stack
- Register

} Kontext

- Umschaltefunktion

`context_switch(stack * current, stack * next)`

Wechselt vom aktuellen Kontext `current` nach `next`

Umschaltung

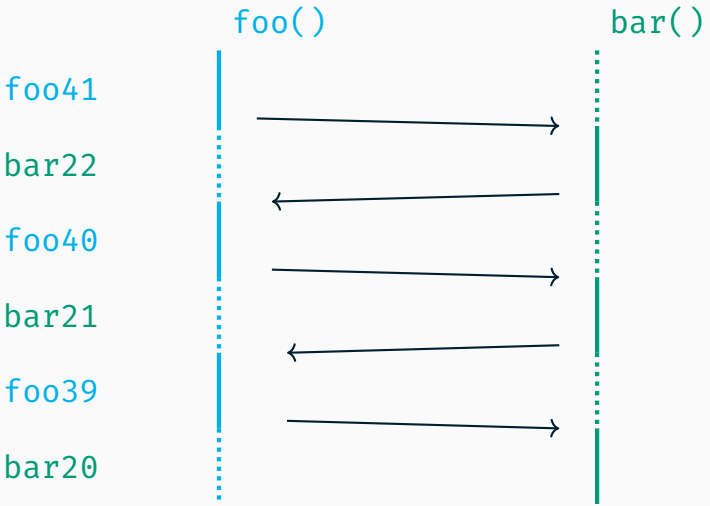
```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;
    context_switch(
        &stack_foo,
        &stack_bar
      );
  }

}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;
    context_switch(
        &stack_bar,
        &stack_foo
      );
  }

}
```

Umschaltung

```
foo41

bar22

foo40

bar21

foo39

bar20
```

Umschaltung



Genau was wir wollen.

Bild: twemoji (modifiziert)

Exkurs: Aufrufkonvention

Kontextsicherung gemäß Konvention

```
void baz(){
  ...

  func();

  ...
}
```

```
void func(){

  ...

  return;
}
```

Kontextsicherung gemäß Konvention

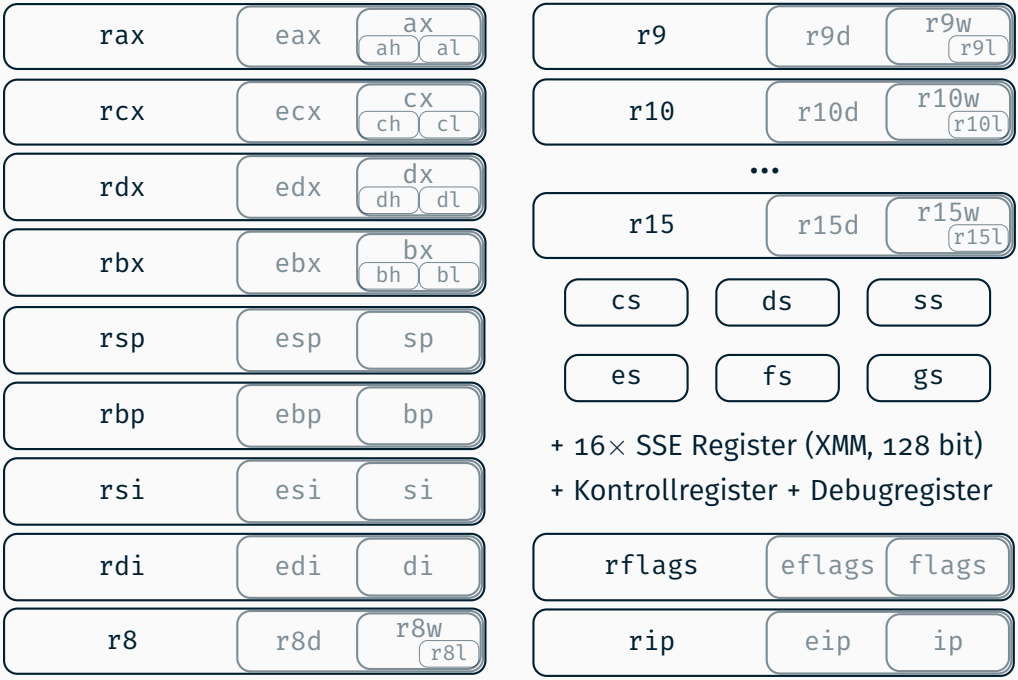
```
void baz(){
  ...

  // flüchtige Register
  // sichern

  func();

  // flüchtige Register
  // wiederherstellen

  ...
}
```

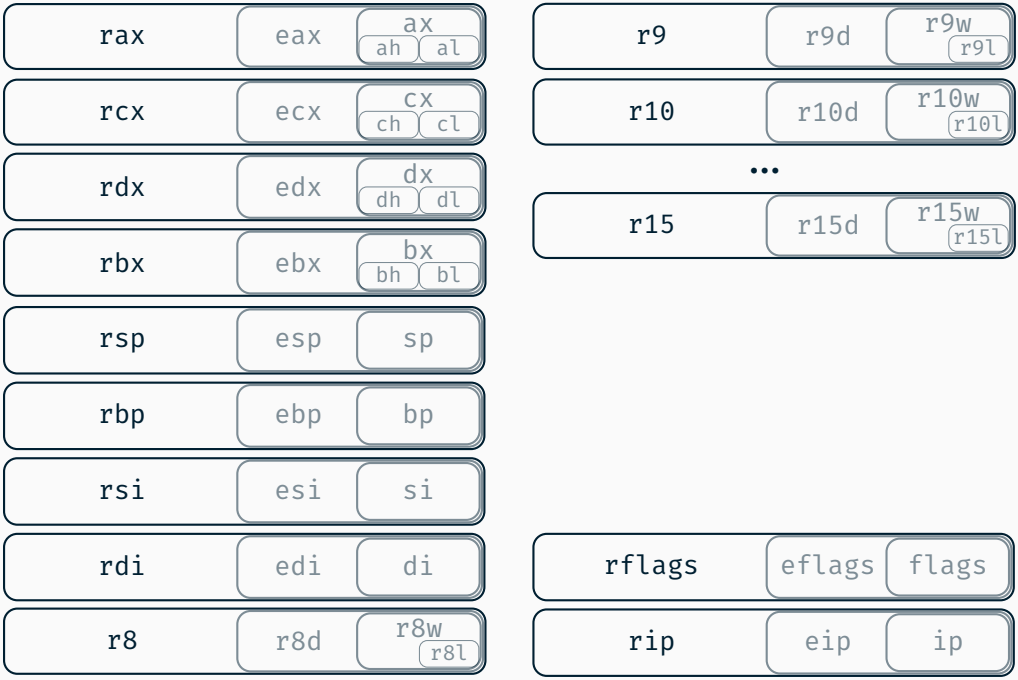
```
void func(){



  ...



  return;
}
```

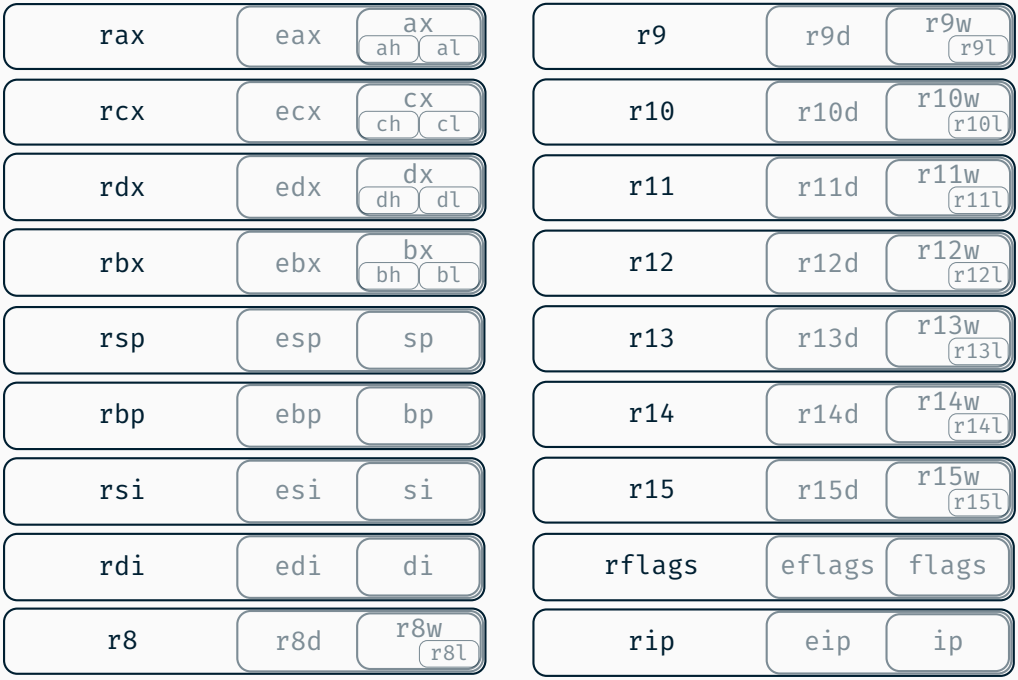
Kontextsicherung gemäß Konvention

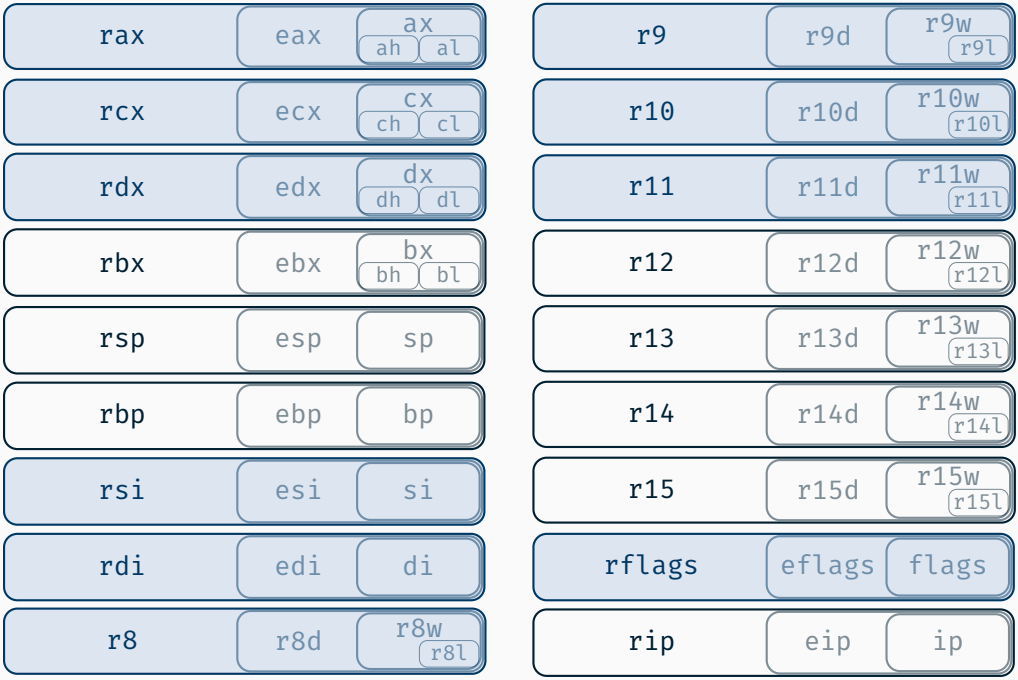
```
void baz(){
  ...

  // flüchtige Register
  // sichern

  func();

  // flüchtige Register
  // wiederherstellen

  ...
}
```

```
void func(){
  // nicht-flüchtige
  // Register
  // sichern

  ...

  // nicht-flüchtige
  // Register
  // wiederherstellen

  return;
}
```

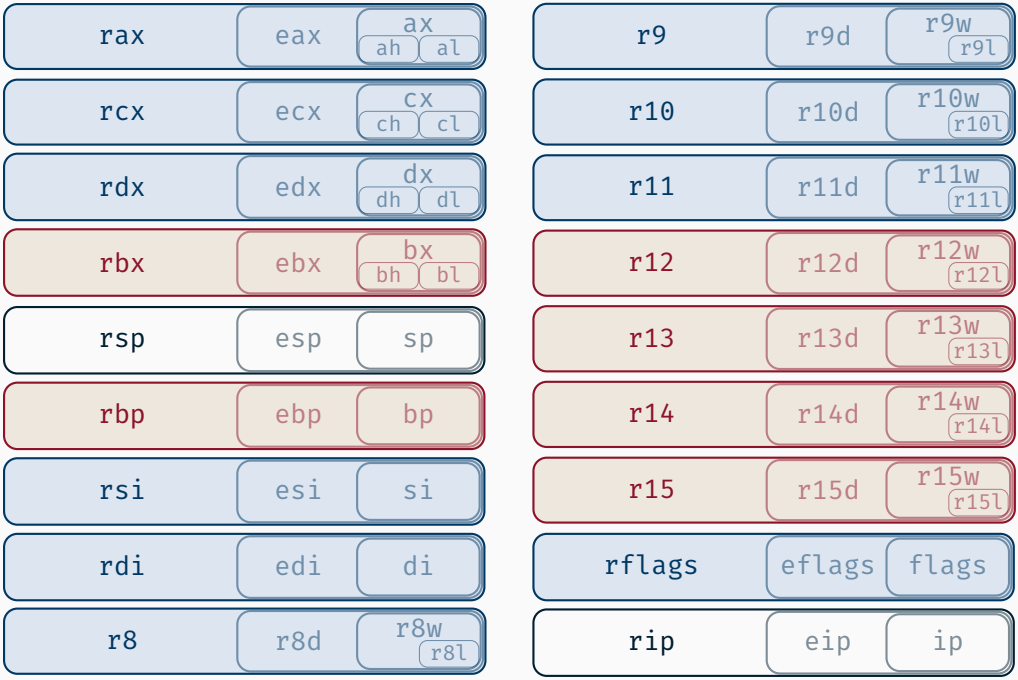

rax
eax
ax
ah
al
rcx
ecx
cx
ch
cl
rdx
edx
dx
dh
dl
rbx
ebx
bx
bh
bl
rsp
esp
sp
rbp
ebp
bp
rsi
esi
si
rdi
edi
di
r8
r8d
r8w
r8l
r9
r9d
r9w
r9l
r10
r10d
r10w
r10l
...
r15
r15d
r15w
r15l
cs
ds
ss
es
fs
gs
+ 16× SSE Register (XMM, 128 bit)
+ Kontrollregister + Debugregister
rflags
eflags
flags
rip
eip
ip


rax
eax
ax
ah
al
rcx
ecx
cx
ch
cl
rdx
edx
dx
dh
dl
rbx
ebx
bx
bh
bl
rsp
esp
sp
rbp
ebp
bp
rsi
esi
si
rdi
edi
di
r8
r8d
r8w
r8l
r9
r9d
r9w
r9l
r10
r10d
r10w
r10l
...
r15
r15d
r15w
r15l
rflags
eflags
flags
rip
eip
ip


rax
eax
ax
ah
al
rcx
ecx
cx
ch
cl
rdx
edx
dx
dh
dl
rbx
ebx
bx
bh
bl
rsp
esp
sp
rbp
ebp
bp
rsi
esi
si
rdi
edi
di
r8
r8d
r8w
r8l
r9
r9d
r9w
r9l
r10
r10d
r10w
r10l
r11
r11d
r11w
r11l
r12
r12d
r12w
r12l
r13
r13d
r13w
r13l
r14
r14d
r14w
r14l
r15
r15d
r15w
r15l
rflags
eflags
flags
rip
eip
ip



flüchtige (*scratch* / *caller-save*) Register

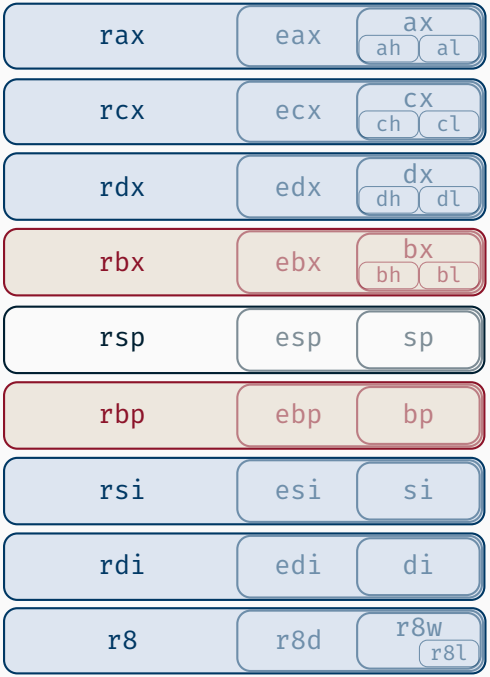


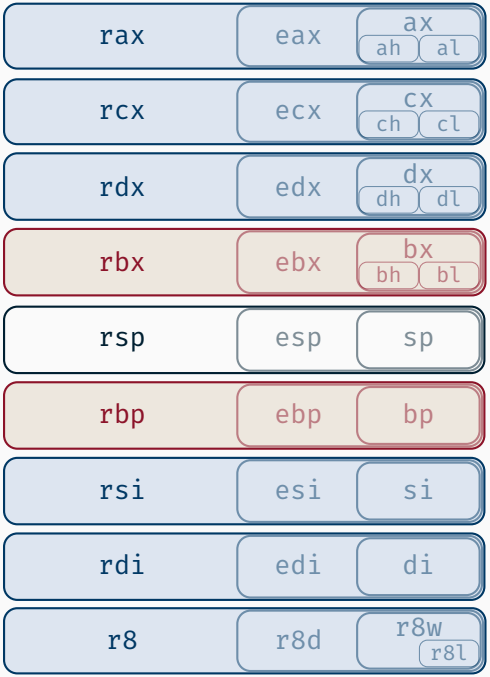
flüchtige (*scratch* / *caller-save*) Register

nicht-flüchtig (*non-scratch* / *callee-save*)

Wieso nicht gleich die **flüchtigen Register** beim Funktionsaufruf nutzen?

Bild: twemoji


rax
eax
ax
ah
al
rcx
ecx
cx
ch
cl
rdx
edx
dx
dh
dl
rbx
ebx
bx
bh
bl
rsp
esp
sp
rbp
ebp
bp
rsi
esi
si
rdi
edi
di
r8
r8d
r8w
r8l


rax
eax
ax
ah
al
Rückgabewert
rcx
ecx
cx
ch
cl
4. Parameter
rdx
edx
dx
dh
dl
3. Parameter
rbx
ebx
bx
bh
bl
rsp
esp
sp
rbp
ebp
bp
rsi
esi
si
2. Parameter
rdi
edi
di
1. Parameter
r8
r8d
r8w
r8l
5. Parameter

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung
push r12

; 2. Parameter in Register rsi
mov esi, 0x2a

; 1. Parameter in Register rdi
mov edi, 0x17

; Funktionsaufruf
call func
```

Syntaxunterschiede bei x86-Assembler

Intel:

```
mov edi, 0x17
```

(Ziel, Quelle)

`objdump -M intel`

AT&T:

```
mov $0x17, %edi
```

(Quelle, Ziel)

Standard bei `objdump`

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung
push r12

; 2. Parameter in Register rsi
mov esi, 0x2a

; 1. Parameter in Register rdi
mov edi, 0x17

; Funktionsaufruf
call func
```

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

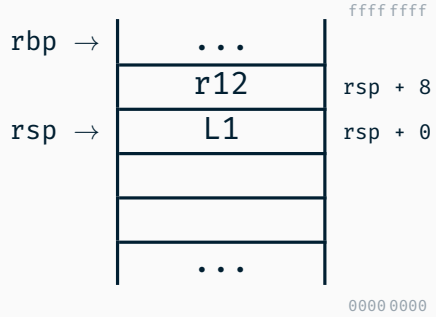
```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
```

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
```

Stack beim Funktionsaufruf:



Zur Erinnerung – für den Stack auf x64 gilt

- Der Stack „wächst“ von *oben* (hohe Adresse) nach *unten* (niedrige Adresse)
- Der Stackzeiger (`rsp`) zeigt auf das zuletzt hinzugefügte Datum

  `push X` verringert zuerst den Wert von `rsp` um 8 Byte und legt dann `X` an die resultierende Adresse

  `pop X` liest den Inhalt des Speichers, auf das `rsp` zeigt, in `X` und erhöht anschließend den Wert von `rsp` um 8 Byte.

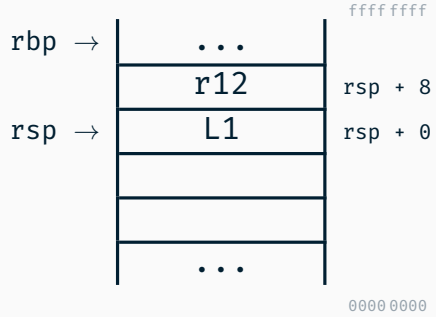
- Bei Funktionsaufrufen muss der Stack an 16 Byte ausgerichtet sein

Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

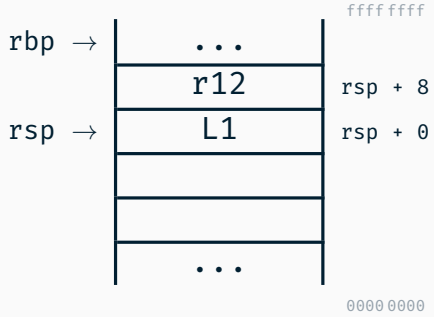
```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
```

Stack beim Funktionsaufruf:



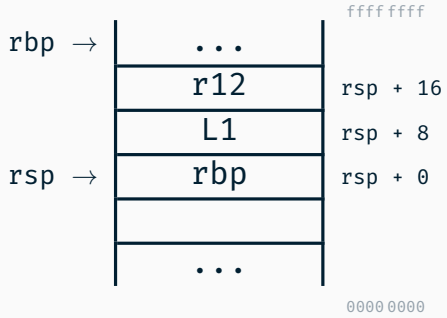
Parameterübergabe gemäß Konvention

```
func:
```



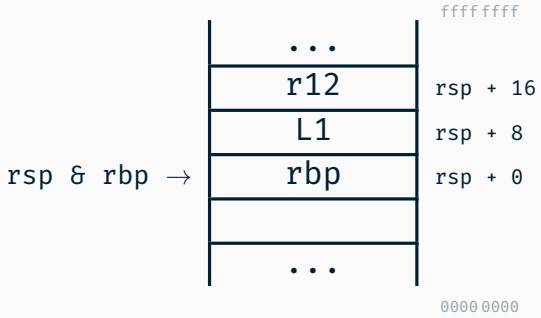
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp
```



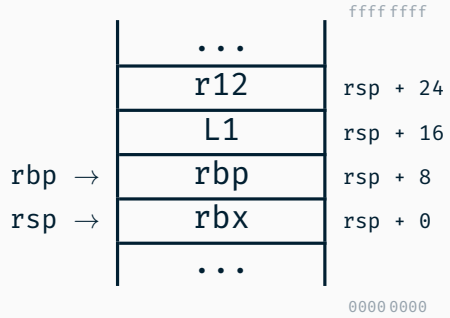
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp
```



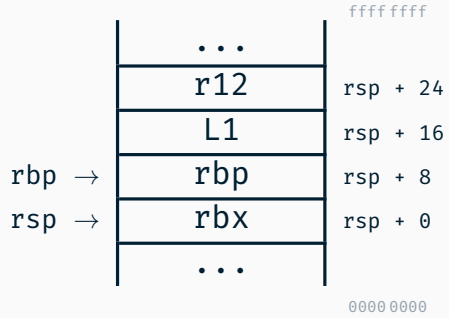
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp

  ; ggf. weitere Register sichern
  push rbx
```



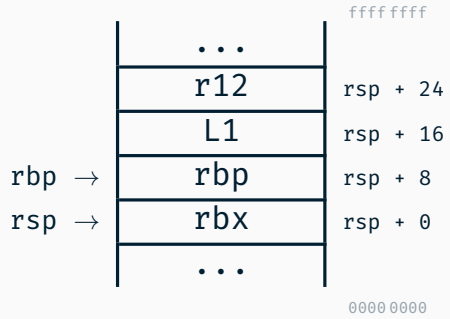
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp

  ; ggf. weitere Register sichern
  push rbx

  ...
```



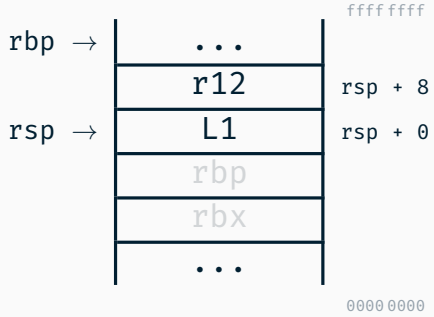
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp

  ; ggf. weitere Register sichern
  push rbx

  ...

  ; Rückgabewert nach rax
  mov eax, 0xd
```



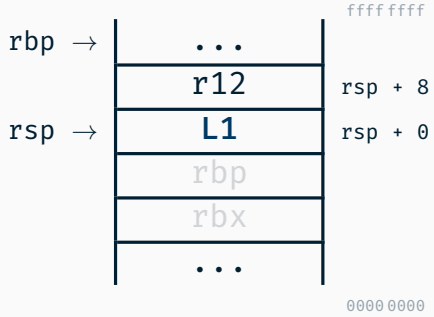
Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp

  ; ggf. weitere Register sichern
  push rbx

  ...

  ; Rückgabewert nach rax
  mov eax, 0xd

  ; Register wiederherstellen
  pop rbx
  pop rbp
```



Parameterübergabe gemäß Konvention

```
func:

  ; alten Rahmenzeiger sichern
  push rbp

  ; aktuellen Rahmenzeiger setzen
  mov rbp, rsp

  ; ggf. weitere Register sichern
  push rbx

  ...

  ; Rückgabewert nach rax
  mov eax, 0xd

  ; Register wiederherstellen
  pop rbx
  pop rbp

  ; Rücksprung
  ret
```

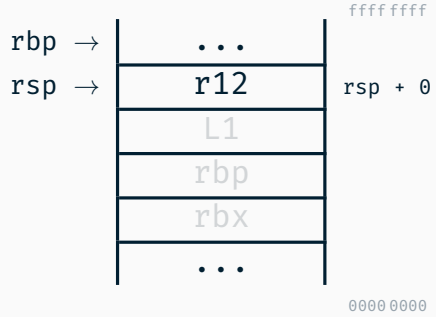


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
```

Stack nach Funktionsaufruf:

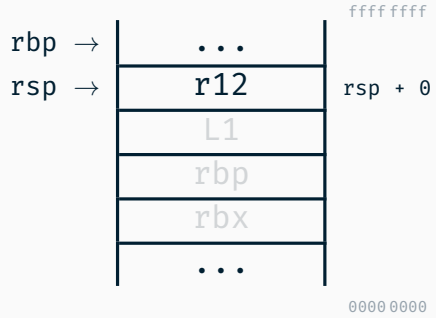


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
  ; Rückgabewert in rax
  mov rsi, rax
```

Stack nach Funktionsaufruf:

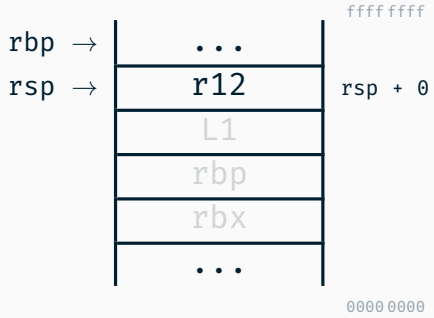


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
  ; ggf. Register Sicherung
  push r12

  ; 2. Parameter in Register rsi
  mov esi, 0x2a

  ; 1. Parameter in Register rdi
  mov edi, 0x17

  ; Funktionsaufruf
  call func
  ; pushed implizit die
  ; Rücksprungadresse
L1:
  ; Rückgabewert in rax
  mov rsi, rax

  ; ggf. Register wiederherstellen
  pop r12
```

Stack nach Funktionsaufruf:



Umsetzung des Kontextwechsel

Umschaltung

```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;
    context_switch(
        &stack_foo,
        &stack_bar
      );
  }

}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;
    context_switch(
        &stack_bar,
        &stack_foo
      );
  }

}
```

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

1. Register sichern

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

1. Register sichern (*via Stack*)

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

1. Register sichern (*via Stack*)
2. Stackpointer (`rsp`) sichern

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

1. Register sichern (*via Stack*)
2. Stackpointer (`rsp`) sichern
3. Stackpointer (`rsp`) laden

Umschaltung im Detail

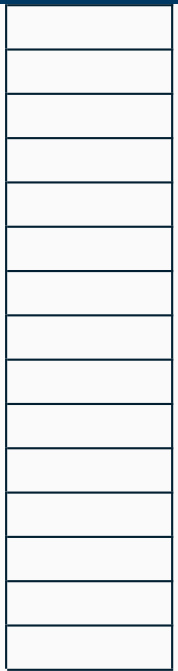
Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

1. Register sichern (*via Stack*)
2. Stackpointer (`rsp`) sichern
3. Stackpointer (`rsp`) laden
4. Register wiederherstellen

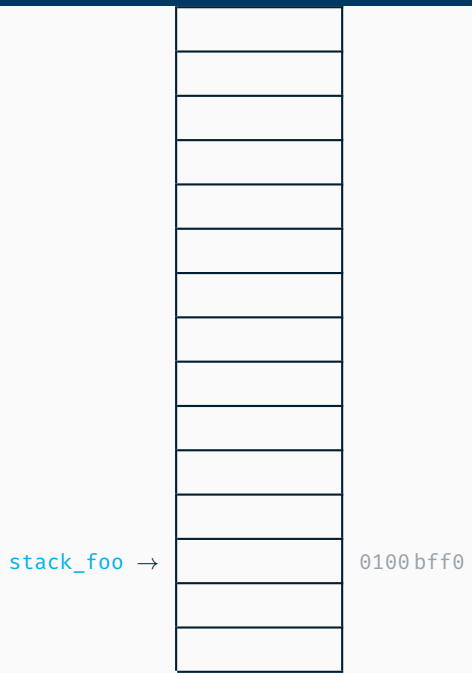
Umsetzung im Detail

```
// Global sichtbar:

struct stack {
  void *kernel;

  // Später auch User-
  // Stackpointer (in BST)
};
```



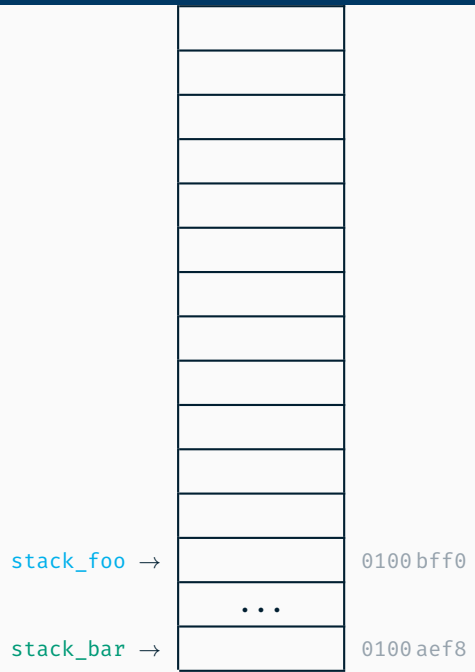
Umsetzung im Detail

```
// Global sichtbar:

struct stack {
  void *kernel;

  // Später auch User-
  // Stackpointer (in BST)
};

stack stack_foo;
```



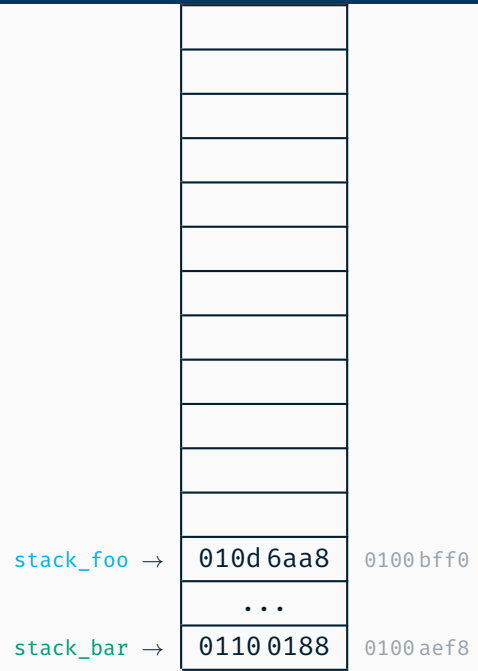
Umsetzung im Detail

```
// Global sichtbar:

struct stack {
  void *kernel;

  // Später auch User-
  // Stackpointer (in BST)
};

stack stack_foo;

stack stack_bar;
```



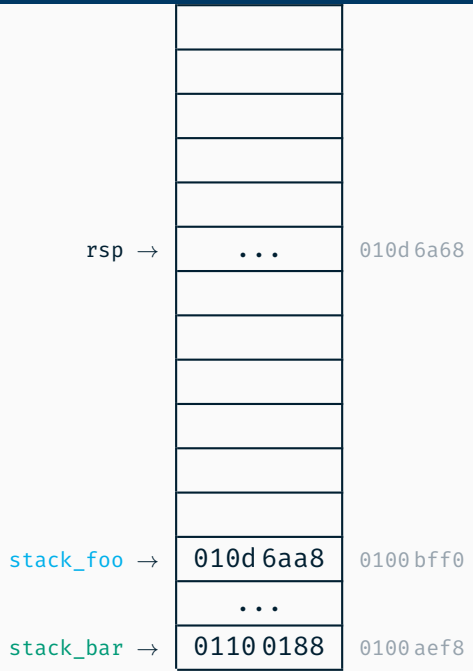
Umsetzung im Detail

```
// Global sichtbar:

struct stack {
  void *kernel;

  // Später auch User-
  // Stackpointer (in BST)
};

stack stack_foo;

stack stack_bar;

// Sinnvoll initialisieren
// (mehr dazu später...)
```



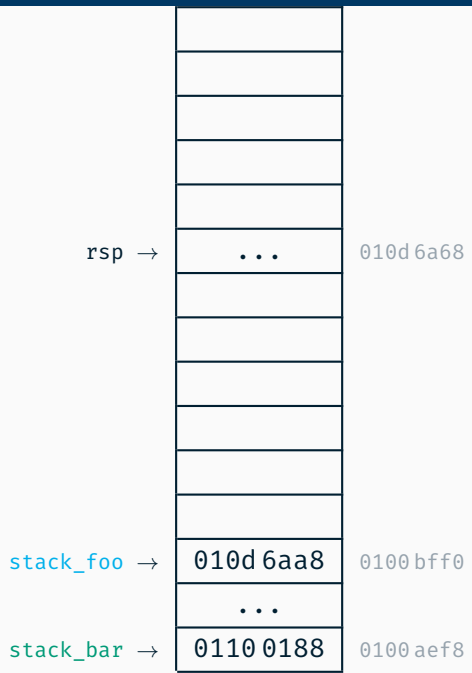
Umsetzung im Detail

```
void foo(){
  ...




  context_switch(
      &stack_foo,
      &stack_bar
    );

  ...
```



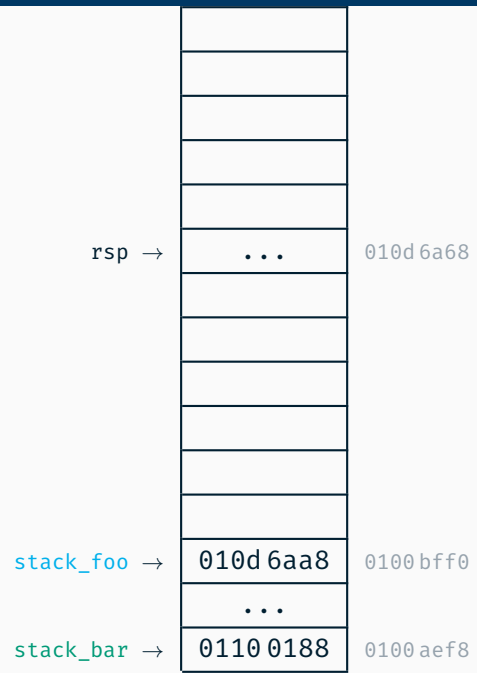
Umsetzung im Detail

```
void foo(){
  ...

  // Sicherung der
  // flüchtigen Register
  // von Kontext foo



  context_switch(
      &stack_foo,
      &stack_bar
    );


  ...
```



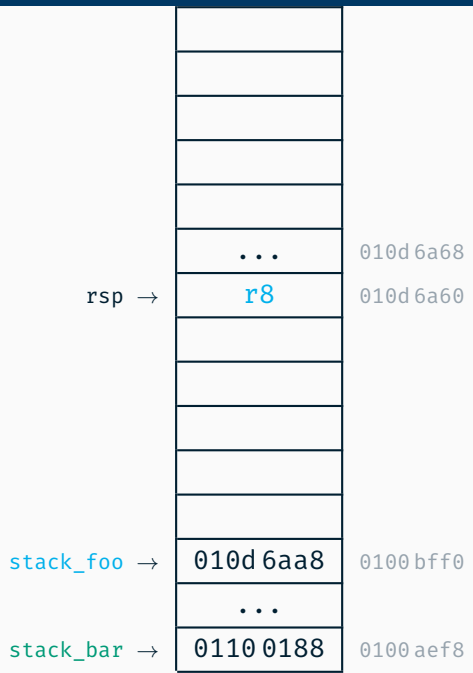
Umsetzung im Detail

```
void foo(){
  ...

  // Sicherung der
  // flüchtigen Register
  // von Kontext foo
  // durch Übersetzer


  context_switch(
      &stack_foo,
      &stack_bar
    );


  ...
```



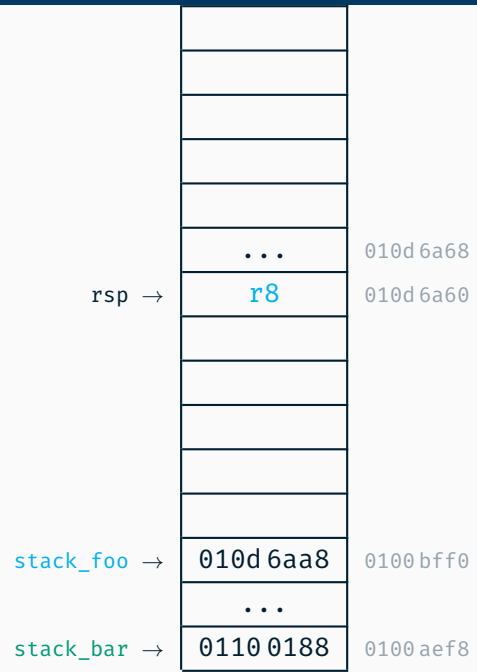
Umsetzung im Detail

```
void foo(){
  ...

  // Sicherung der
  // flüchtigen Register
  // von Kontext foo
  // durch Übersetzer
  // (z.B. r8)

  context_switch(
      &stack_foo,
      &stack_bar
    );


  ...
```



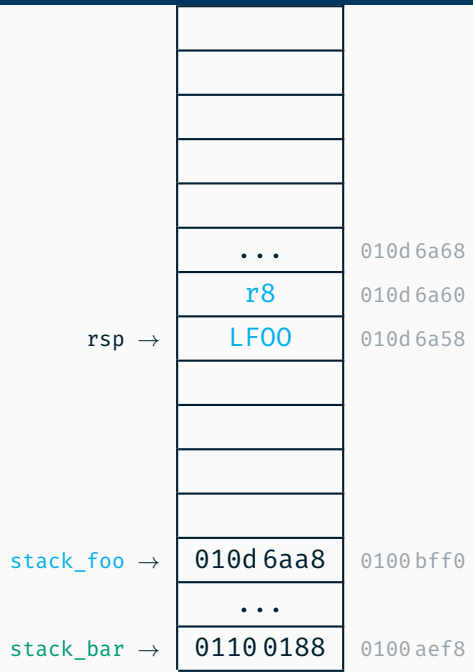
Umsetzung im Detail

```
void foo(){
  ...

  // Sicherung der
  // flüchtigen Register
  // von Kontext foo
  // durch Übersetzer
  // (z.B. r8)

  context_switch(
      &stack_foo,
      &stack_bar
    );


  ...
```



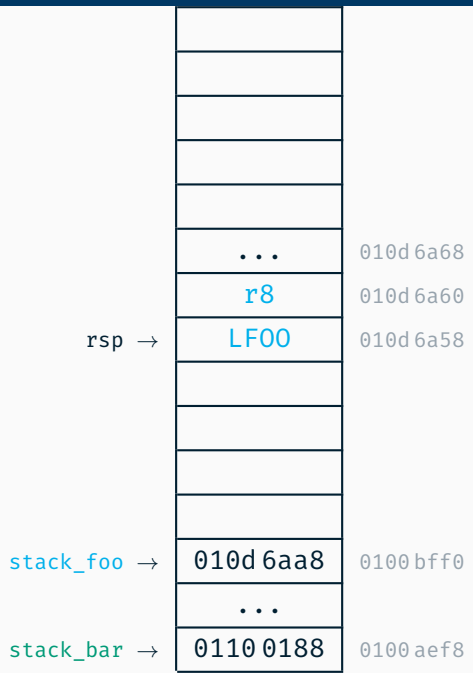
Umsetzung im Detail

```
void foo(){
  ...

  // Sicherung der
  // flüchtigen Register
  // von Kontext foo
  // durch Übersetzer
  // (z.B. r8)

  context_switch(
      &stack_foo,
      &stack_bar
    );
LFOO:

  ...
```



Umsetzung im Detail

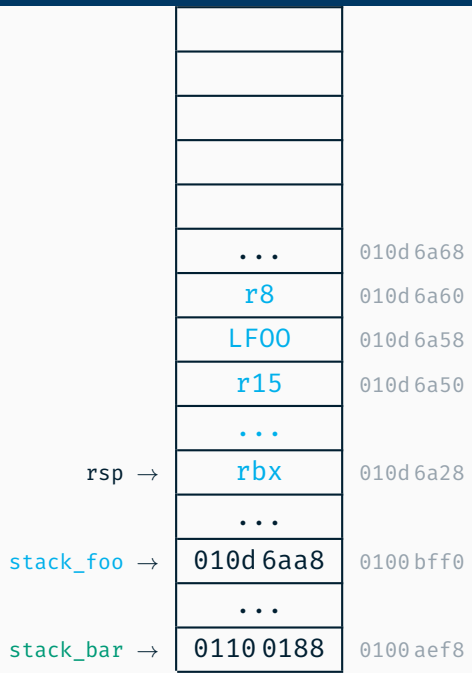
```
context_switch:
```




```
  ret
```

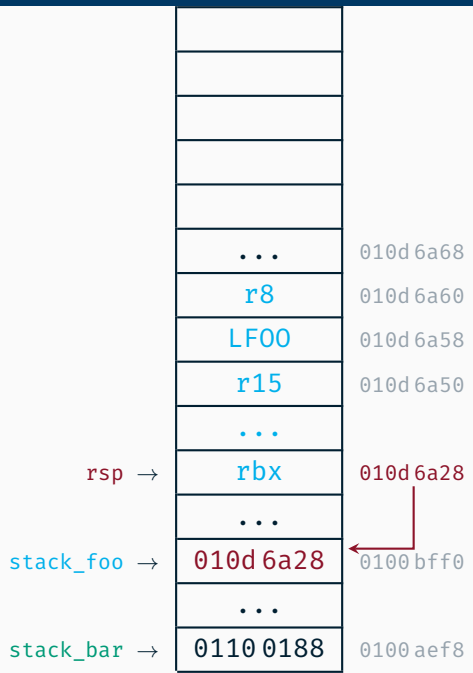
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ret
```



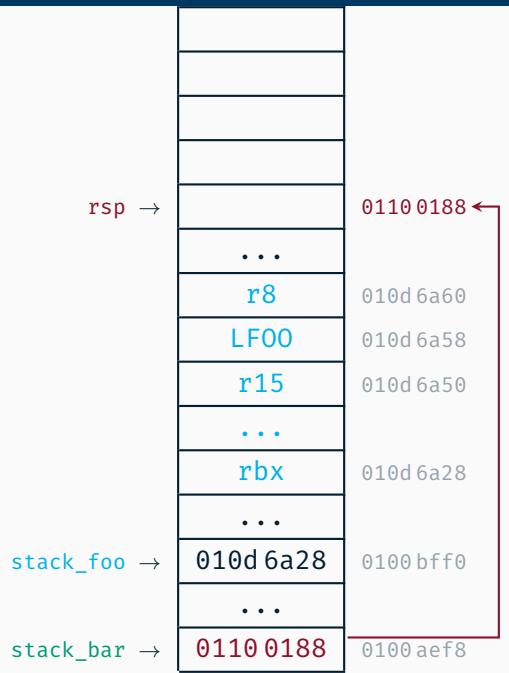
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ;; Stackzeiger von
  ;; foo sichern

  ret
```



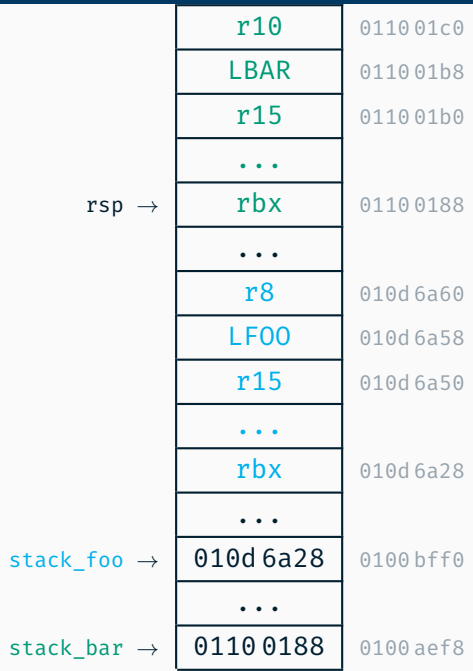
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden

  ret
```



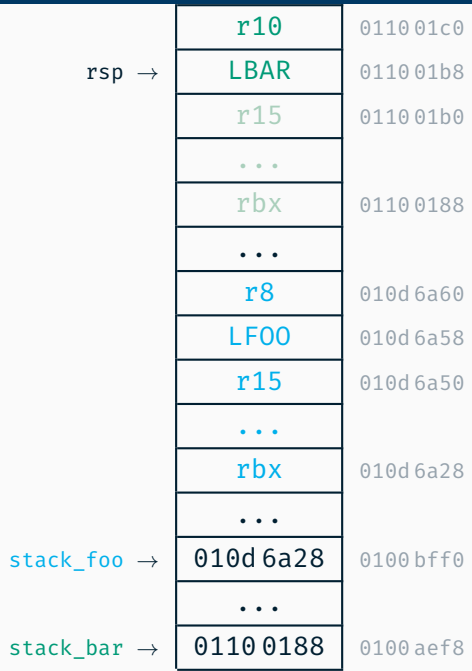
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden

  ret
```



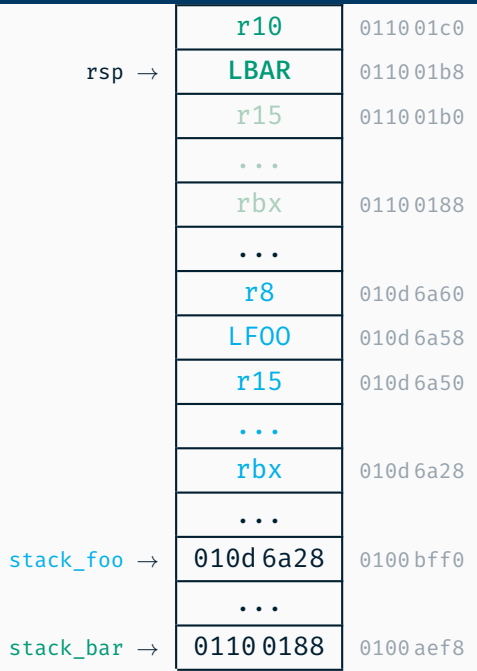
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden

  ;; nicht-flüchtige
  ;; Register von
  ;; bar laden

  ret
```



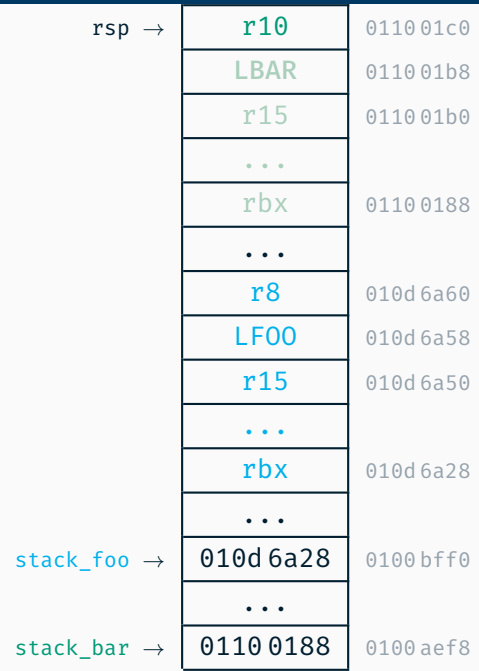
Umsetzung im Detail

```
context_switch:

  ;; nicht-flüchtige
  ;; Register von
  ;; foo sichern

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden

  ;; nicht-flüchtige
  ;; Register von
  ;; bar laden

  ret
```



Umsetzung im Detail

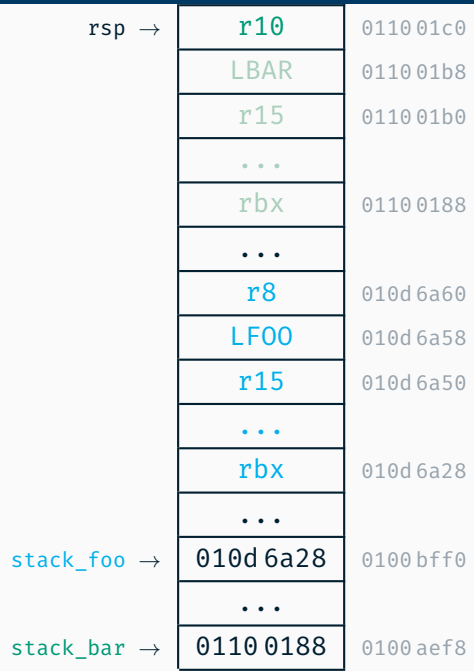
```
void bar(){
  ...

  context_switch(
      &stack_bar,
      &stack_foo
    );
LBAR:
```




```
  ...
```

Umsetzung im Detail

```
void bar(){
  ...

  context_switch(
      &stack_bar,
      &stack_foo
    );
LBAR:

  // Wiederhestellen der
  // flüchtigen Register
  // von Kontext bar
  // durch Übersetzer
  // (z.B. r10)

  ...
```



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`
→ `context_switch` mit entsprechenden Stack

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`
→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`
→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

- `context_switch` mit Dummy-Stackzeiger als `current`

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`
→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

- `context_switch` mit Dummy-Stackzeiger als `current`

Was wird für jede Koroutine gebraucht?

Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`
→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

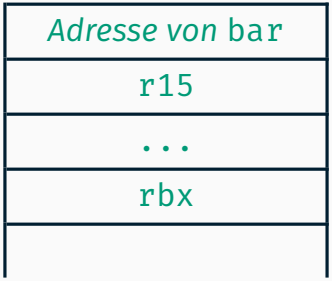
- `context_switch` mit Dummy-Stackzeiger als `current`

Was wird für jede Koroutine gebraucht?

- eigener, präparierter Stack
- Speicher für Stackzeiger (`struct stack`)

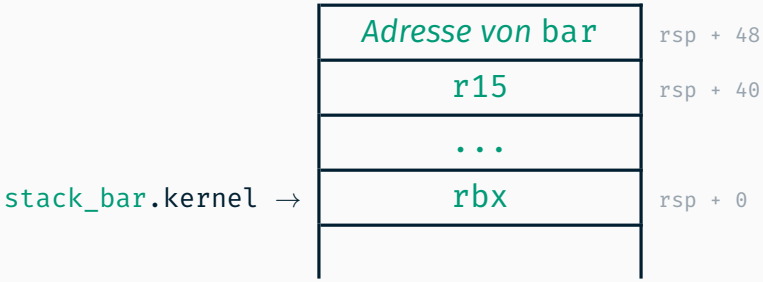
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



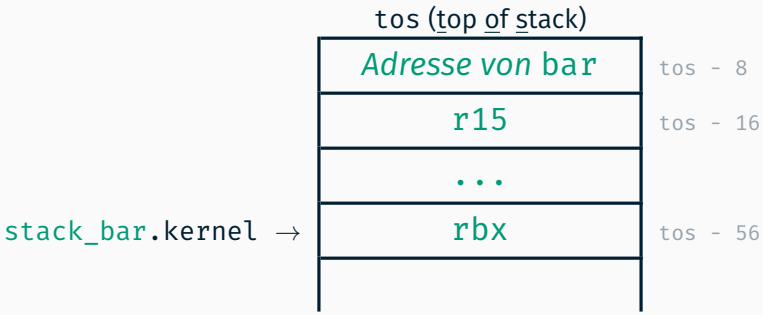
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



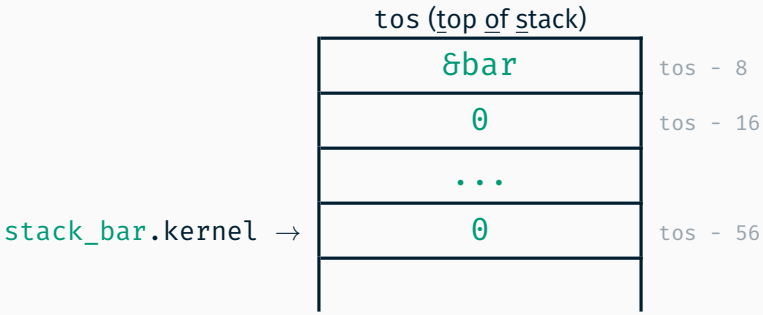
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



Stack aufsetzen

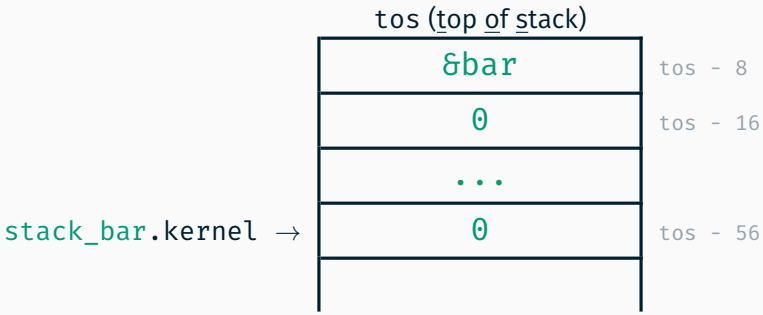
Stack für Einsprung in Koroutine `void bar()`



was passiert nun bei `context_switch`?

Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



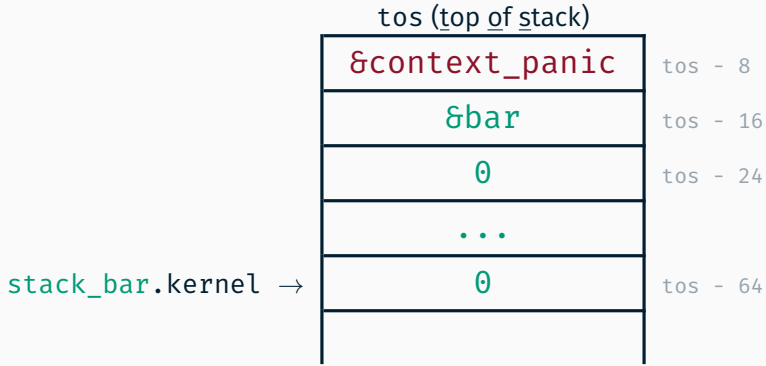
was passiert wenn die Koroutine `bar` abgearbeitet ist?

Stack aufsetzen (Robust)

```
void context_panic() {
  kernelpanic("Application should not return!1!!11");
}
```

Stack aufsetzen (Robust)

```
void context_panic() {
  kernelpanic("Application should not return!1!!11");
}
```



Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)`

Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)` mittels einer Hilfsfunktion (`bar_wrapper`)

Stack aufsetzen mit Funktionsparameter

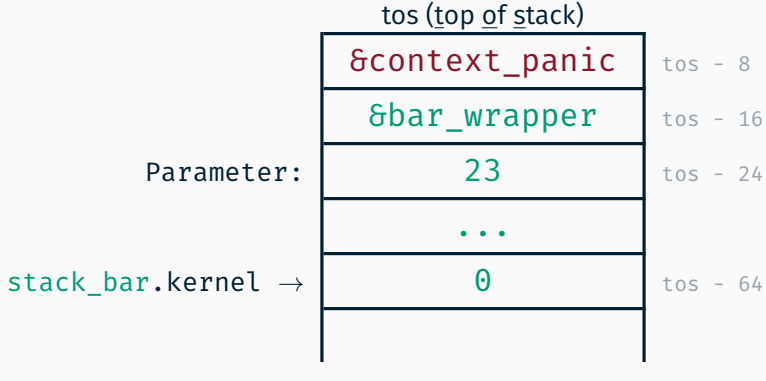
Stack für Einsprung in Koroutine `void bar(int i)` mittels einer Hilfsfunktion (`bar_wrapper`)

- liest Parameter aus nicht-flüchtigem Register (`r15`)
- schreibt Wert in flüchtiges Parameterregister (`rdi`)
- springt in eigentliche Funktion (`bar`)

Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)` mittels einer Hilfsfunktion (`bar_wrapper`)

- liest Parameter aus nicht-flüchtigem Register (`r15`)
- schreibt Wert in flüchtiges Parameterregister (`rdi`)
- springt in eigentliche Funktion (`bar`)



Umsetzung in OOStuBS/MPStuBS

Threads

```
class Thread {
    struct stack context;
  public:
    Thread(void* tos);
    virtual void action() = 0;
}
```

Threads

```
class Thread {
    struct stack context;
  public:
    Thread(void* tos);
    virtual void action() = 0;
}
```

Adresse einer virtuellen Member-Funktion nicht (ohne weiteres) ermittelbar

Threads

```
class Thread {
    struct stack context;
  public:
    Thread(void* tos);
    virtual void action() = 0;
}
```

Adresse einer virtuellen Member-Funktion nicht (ohne weiteres) ermittelbar

```
void kickoff (Thread* t){
  t->action();
}
```

Threads: Stack aufsetzen

```
void context_prepare(stack * target,
                     void * tos,
                     Thread * that);
```

Präpariert den Stack für den ersten Einsprung

Threads: Stack aufsetzen

```
void context_prepare(stack * target,
                     void * tos,
                     Thread * that);
```

Präpariert den Stack für den ersten Einsprung

- in C++ (statt Assembler)

Threads: Stack aufsetzen

```
void context_prepare(stack * target,
                     void * tos,
                     Thread * that);
```

Präpariert den Stack für den ersten Einsprung

- in C++ (statt Assembler)
- statisch übergebenen Speicher `tos` als Stack aufsetzen

Threads: Stack aufsetzen

```
void context_prepare(stack * target,
                     void * tos,
                     Thread * that);
```

Präpariert den Stack für den ersten Einsprung

- in C++ (statt Assembler)
- statisch übergebenen Speicher `tos` als Stack aufsetzen
- Pointerarithmetik ist hilfreich

Threads: Stack aufsetzen

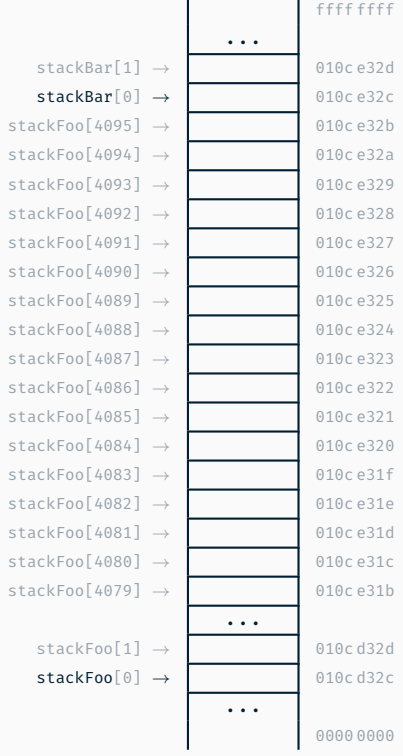
```
void context_prepare(stack * target,
                     void * tos,
                     Thread * that);
```

Präpariert den Stack für den ersten Einsprung

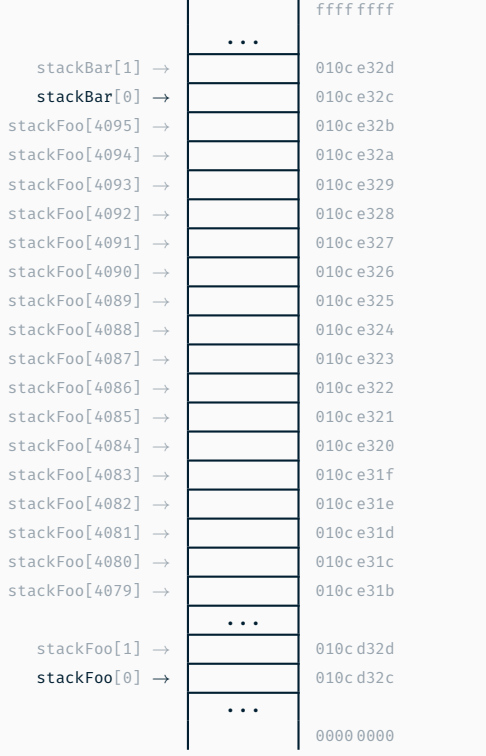
- in C++ (statt Assembler)
- statisch übergebenen Speicher `tos` als Stack aufsetzen
- Pointerarithmetik ist hilfreich
- Stackpointer in `target->kernel` entsprechend setzen

```
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];
```

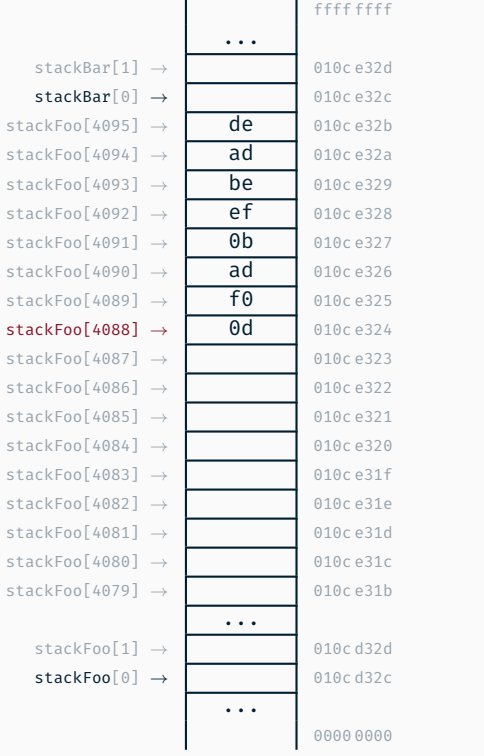
```c
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];
```



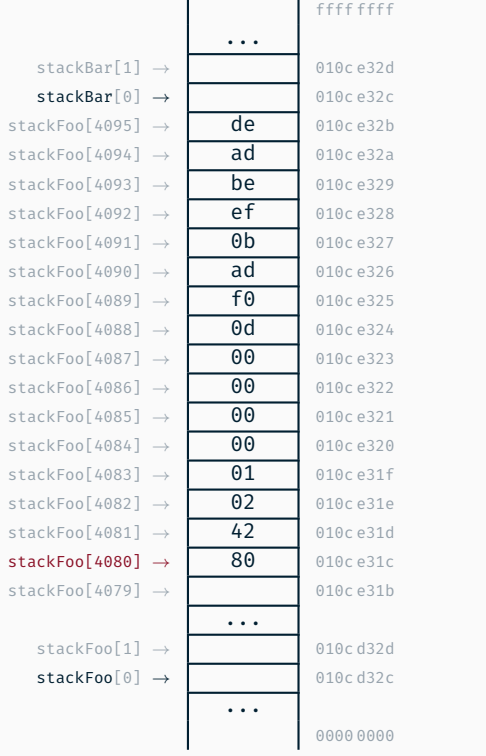
```
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;
```



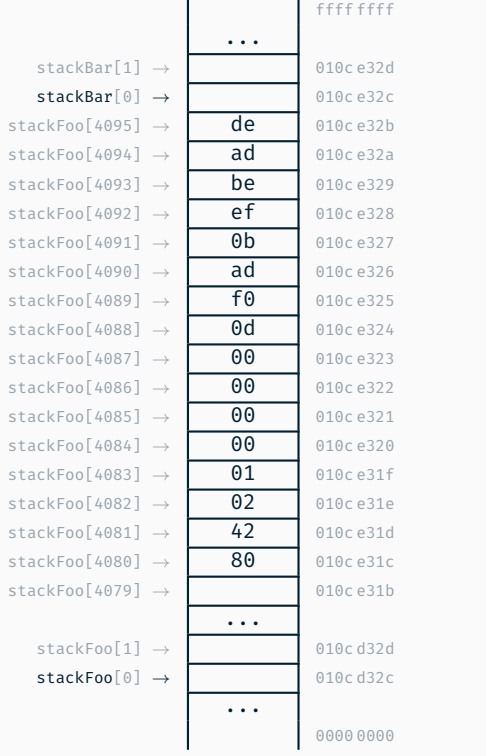
```c
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;

rsp--;
// rsp = &(stackFoo[4088])
*rsp = 0xdeadbeef0badf00d;
```



```
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;

rsp--;
// rsp = &(stackFoo[4088])
*rsp = 0xdeadbeef0badf00d;

rsp--;
// rsp = &(stackFoo[4080])
*rsp = &foo;
// &foo = 0x102 4280
```



```
#define STACKSIZE 4096

char stackFoo[STACKSIZE];
char stackBar[STACKSIZE];

void* tos = stackFoo + STACKSIZE;
void** rsp = (void**) tos;

rsp--;
// rsp = &(stackFoo[4088])
*rsp = 0xdeadbeef0badf00d;

rsp--;
// rsp = &(stackFoo[4080])
*rsp = &foo;
// &foo = 0x102 4280

// ...
```



Scheduler

Scheduler verwaltet Koroutinen (Threads) zentral

```
class Scheduler {
    Queue<Thread> routines;
  public:
    void ready(Thread *);
    void schedule();
    void resume();
};
```

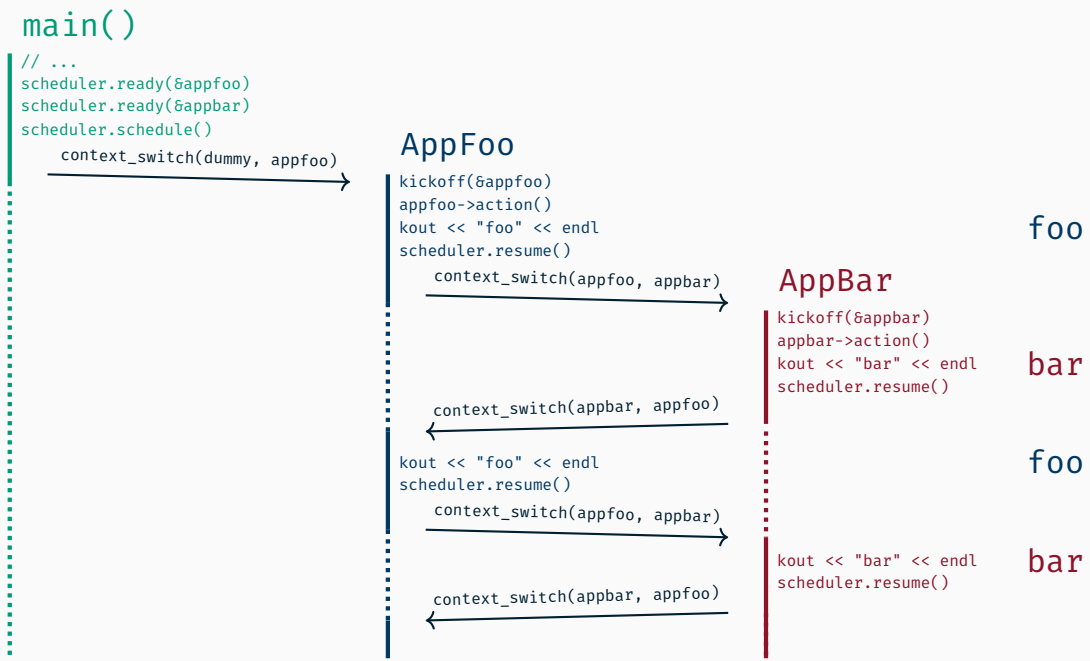
```
// Thread AppFoo
void AppFoo::action(){
  while (1){
    kout << "foo" << endl;
    scheduler.resume();
  }
}
```

```
// Thread AppBar
void AppBar::action(){
  while (1){
    kout << "bar" << endl;
    scheduler.resume();
  }
}
```

```
// Thread AppFoo
void AppFoo::action(){
  while (1){
    kout << "foo" << endl;
    scheduler.resume();
  }
}
```

```
// Thread AppBar
void AppBar::action(){
  while (1){
    kout << "bar" << endl;
    scheduler.resume();
  }
}
```

```
// main.cc
AppFoo appfoo;
AppBar appbar;
int main (){
  // ...
  scheduler.ready(&appfoo);
  scheduler.ready(&appbar);
  scheduler.schedule();
}
```


main()
// ...
scheduler.ready(&appfoo)
scheduler.ready(&appbar)
scheduler.schedule()
context_switch(dummy, appfoo)
AppFoo
kickoff(&appfoo)
appfoo->action()
kout << "foo" << endl
scheduler.resume()
context_switch(appfoo, appbar)
AppBar
kickoff(&appbar)
appbar->action()
kout << "bar" << endl
scheduler.resume()
foo
bar
context_switch(appbar, appfoo)
kout << "foo" << endl
scheduler.resume()
foo
context_switch(appfoo, appbar)
kout << "bar" << endl
scheduler.resume()
bar
context_switch(appbar, appfoo)

Scheduler

Scheduler verwaltet Koroutinen (Threads) zentral

```
class Scheduler {
    Queue<Thread> routines;
  public:
    void ready(Thread *);
    void schedule();
    void resume();
};
```

OOStuBS immer synchrone Aufrufe
$\rightarrow$ Konsistenz gesichert

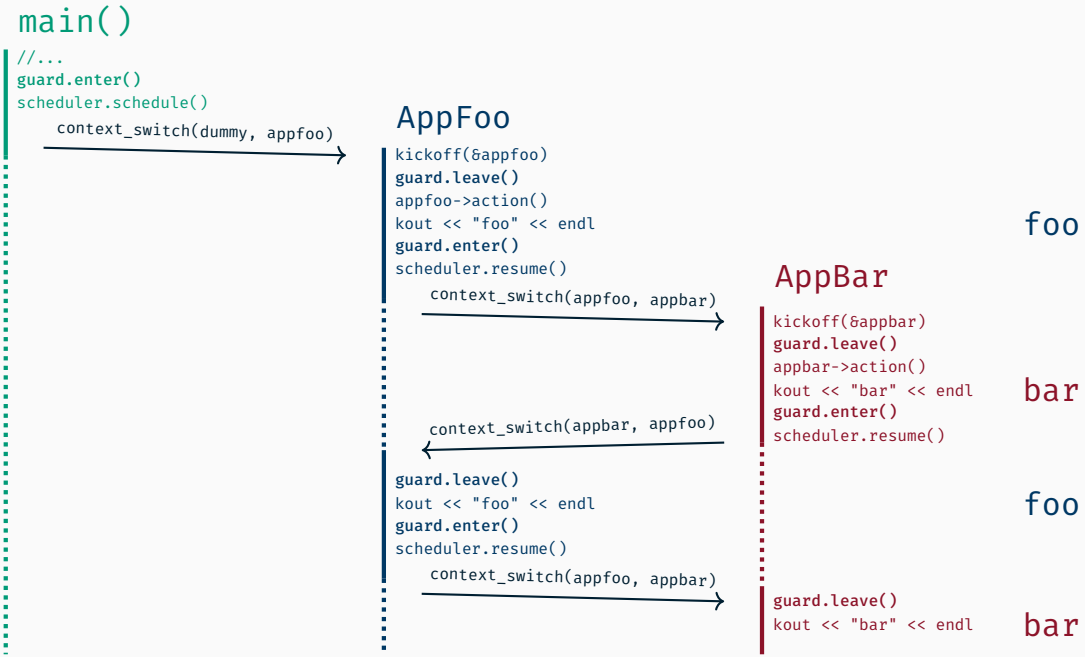
Scheduler

Scheduler verwaltet Koroutinen (Threads) zentral

```
class Scheduler {
    Queue<Thread> routines;
  public:
    void ready(Thread *);
    void schedule();
    void resume();
};
```

OOStuBS immer synchrone Aufrufe
$\rightarrow$ Konsistenz gesichert

MPStuBS Synchronisation notwendig


main()
//...
guard.enter()
scheduler.schedule()
context_switch(dummy, appfoo)
AppFoo
kickoff(&appfoo)
guard.leave()
appfoo->action()
kout << "foo" << endl
guard.enter()
scheduler.resume()
context_switch(appfoo, appbar)
AppBar
kickoff(&appbar)
guard.leave()
appbar->action()
kout << "bar" << endl
guard.enter()
scheduler.resume()
foo
context_switch(appbar, appfoo)
bar
guard.leave()
kout << "foo" << endl
guard.enter()
scheduler.resume()
foo
context_switch(appfoo, appbar)
guard.leave()
kout << "bar" << endl
bar

Fragen?

Nächste Woche (12. & 13. Dezember)
Abgabe von Aufgabe 3 im WinCIP

Bild: twemoji