

Übung zu Betriebssysteme

Kontextwechsel

Wintersemester 2021/22

Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Prüfung

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich
- Anmeldung via Waffel
 - Freischaltung vermutlich heute abend, Mail kommt

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich
- Anmeldung via Waffel
 - Freischaltung vermutlich heute abend, Mail kommt
- Prüfungsprotokolle bei der FSI Informatik

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich
- Anmeldung via Waffel
 - Freischaltung vermutlich heute abend, Mail kommt
- Prüfungsprotokolle bei der FSI Informatik

Rechnerübung Abgabe Aufgabe 4

Prüfung

- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich
- Anmeldung via Waffel
 - Freischaltung vermutlich heute abend, Mail kommt
- Prüfungsprotokolle bei der FSI Informatik

Rechnerübung Abgabe Aufgabe 4

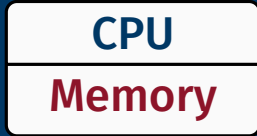
- Termin am Freitag, den 7. Januar

Prüfung

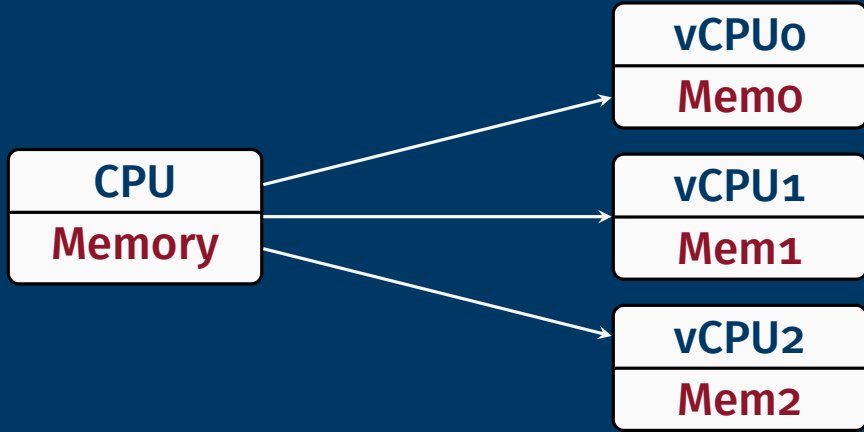
- Prüfungstermine
 - KW 7 (ab 14. Februar, 1. Prüfungszeitraum der TechFak)
 - KW 10 (ab 7. März)
 - KW 14 & 15 (ab 4. April, 2. Prüfungszeitraum)
- Präsenz- und Fernprüfung möglich
- Anmeldung via Waffel
 - Freischaltung vermutlich heute abend, Mail kommt
- Prüfungsprotokolle bei der FSI Informatik

Rechnerübung Abgabe Aufgabe 4

- Termin am Freitag, den 7. Januar
 - Remoteabgabe?
 - verschieben auf Dienstag, 11. Januar?



tatsächliche
Hardware



tatsächliche
Hardware

illusionierte
Hardware

- Speicher virtualisieren

- CPU virtualisieren

■ Speicher virtualisieren ist einfach:

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

Memory



■ CPU virtualisieren

■ Speicher virtualisieren ist einfach:

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

Memory



■ CPU virtualisieren

- nicht teilbar im Ort

■ Speicher virtualisieren ist einfach:

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

Memory



■ CPU virtualisieren

- 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

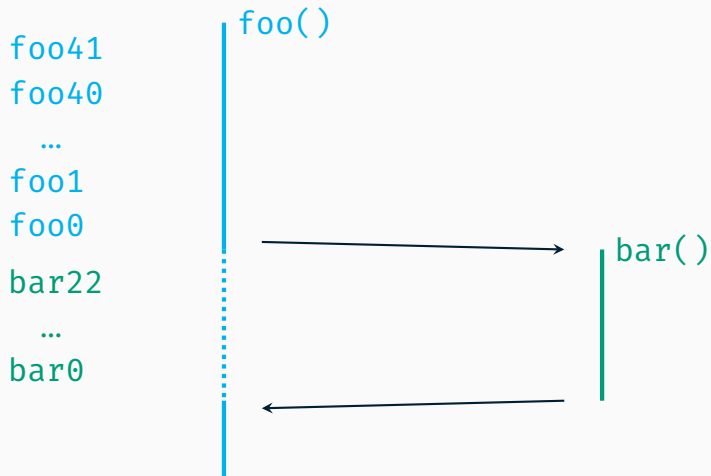
foo0

bar22

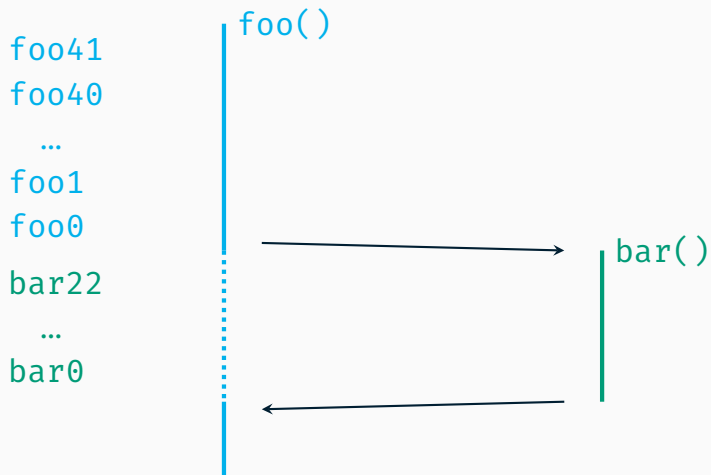
...

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

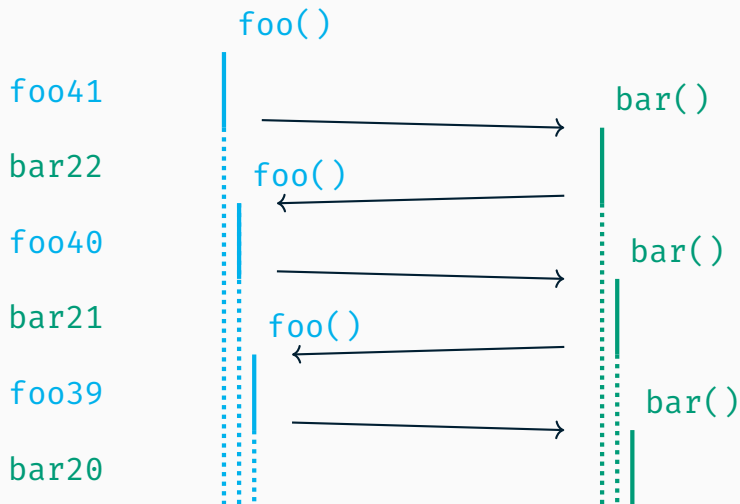
foo40

bar21

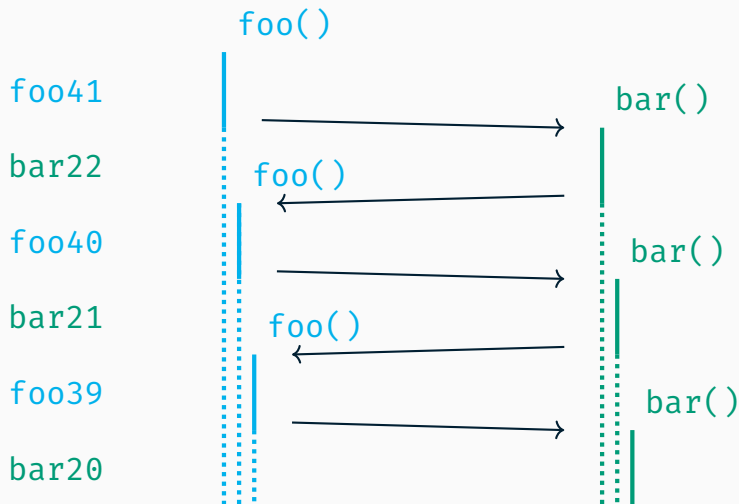
foo39

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
 - Umschaltefunktion
- } Kontext

Umschaltung



Kontrollflusszustand **sichern** & **laden**

- Stack
 - Register
 - Umschaltfunktion
- } Kontext

```
context_switch(StackPointer *curr, StackPointer *next)
```

Wechselt vom aktuellen Kontext **curr** 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

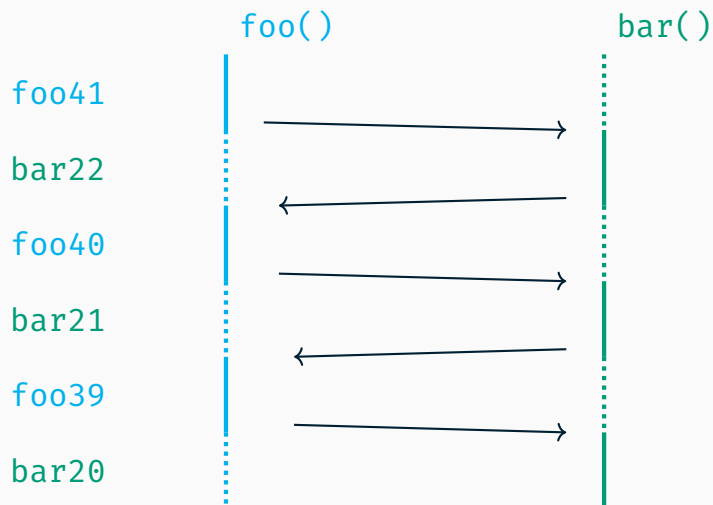
foo40

bar21

foo39

bar20

Umschaltung



Genau was wir wollen.

Bild: twemoji (modifiziert)



Exkurs:

**Von der Hochsprache
zum Maschinencode**

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache

(3. Generation)

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache

(3. Generation)

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache (3. Generation)

```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache
(3. Generation)

```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

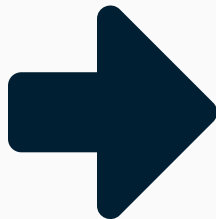
Maschinensprache
(1. Generation)

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache
(3. Generation)



```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

Maschinensprache
(1. Generation)

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

höhere Programmiersprache
(3. Generation)



```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

Maschinensprache
(1. Generation)

```
#include "user/app1/appl.h"
#include "device/textstream.h"
#include "interrupt/guarded.h"
#include "machine/core.h"

extern TextStream kout;

void Application::action() {
    unsigned core = Core::getID();
    for (unsigned n = 0; ; ++n) {
        Guarded section;
        kout.setPos(0, id);
        kout << n;
        kout.flush();
    }
}
```

```
class TextStream : public OutputStream, public TextWindow {
    // ...

} kout;

void Application::action() {
    unsigned n = 0;
    unsigned core;
    core = Core::getID();
    while(true) {
        Guard::enter();
        kout.setPos(0, id);
        n += 1;
        kout.operator<<(n);
        kout.flush();
        Guard::leave();
    }
}
```

```
class TextStream : public OutputStream, public TextWindow {  
    // ...  
  
} kout;  
  
void Application::action(Application * this) {  
    unsigned n = 0;  
    unsigned core;  
    core = Core::getID();  
    while(true) {  
        Guard::enter();  
        TextWindow::setPos(&kout, 0, id);  
        n += 1;  
        OutputStream::operator<<(&kout, i);  
        TextStream::flush(&kout);  
        Guard::leave();  
    }  
}
```

```
class TextStream {
    class OutputStream { ... };
    class TextWindow { ... };
} kout;

void Application::action(Application * this) {
    unsigned n = 0;
    unsigned core;
    core = Core::getID();
    while(true) {
        Guard::enter();
        TextWindow::setPos(&(kout.TextWindow), 0, id);
        n += 1;
        OutputStream::operator<<(&(kout.OutputStream), i);
        TextStream::flush(&(kout.OutputStream));
        Guard::leave();
    }
}
```

```
class TextStream {
    class OutputStream { ... }; // offset 0
    class TextWindow { ... };    // offset 64
} kout;

void Application::action(Application * this) {
    unsigned n = 0;
    unsigned core;
    core = Core::getID();
    while(true) {
        Guard::enter();
        TextWindow::setPos(&kout+64, 0, id);
        n += 1;
        OutputStream::operator<<(&kout+0, i);
        TextStream::flush(&kout+0);
        Guard::leave();
    }
}
```

```
class TextStream {
    class OutputStream { ... }; // offset 0
    class TextWindow { ... }; // offset 64
} kout;

void _ZN11Application6actionEv(Application * this) {
    unsigned n = 0;
    unsigned core;
    core = _ZN4Core5getIDEv();
    while(true) {
        _ZN5Guard5enterEv();
        _ZN10TextWindow6setPosEjj(&kout+64, 0, id);
        n += 1;
        _ZN12OutputStreamlsEj(&kout, i);
        _ZN10TextStream5flushEv(&kout);
        _ZN5Guard5leaveEv();
    }
}
```

```
void _ZN11Application6actionEv(Application * this) {

    unsigned n = 0
    unsigned core;
    core = _ZN4Core5getIDEv();

    while(true) {
        _ZN5Guard5enterEv();

        _ZN10TextWindow6setPosEjj(&kout+64, 0, core)

        n += 1;
        _ZN12OutputStream5Ej(&kout, n);

        _ZN10TextStream5flushEv(&kout);
        _ZN5Guard5leaveEv();
    }
}
```

```
_ZN11Application6actionEv:  
    endbr64  
    push    rbp  
    push    rbx  
    xor     ebx, ebx  
    sub     rsp, 0x8  
    call    _ZN4Core5getIDEv  
    mov     ebp, eax  
LOOP:  
    call    _ZN5Guard5enterEv  
    mov     edx, ebp  
    xor     esi, esi  
    mov     edi, kout+64  
    call    _ZN10TextWindow6setPosEjj  
    mov     esi, ebx  
    mov     edi, kout  
    add     ebx, 0x1  
    call    _ZN12OutputStreamlsEj  
    mov     edi, kout  
    call    _ZN10TextStream5flushEv  
    call    _ZN5Guard5leaveEv  
    jmp     LOOP
```

```
0100ba80 <_ZN11Application6actionEv>:
  100ba80:  endbr64
  100ba84:  push    rbp
  100ba85:  push    rbx
  100ba86:  xor     ebx, ebx
  100ba88:  sub     rsp, 0x8
  100ba8c:  call    1008be0 <_ZN4Core5getIDEv>
  100ba91:  mov     ebp, eax
  100ba93:  nop     DWORD PTR [rax+rax*1+0x0]
  100ba98:  call    1007e80 <_ZN5Guard5enterEv>
  100ba9d:  mov     edx, ebp
  100ba9f:  xor     esi, esi
  100baa1:  mov     edi, 0x1024584 <kout+64>
  100baa6:  call    1008d90 <_ZN10TextWindow6setPosEjj>
  100baab:  mov     esi, ebx
  100baad:  mov     edi, 0x1024520 <kout>
  100bab2:  add     ebx, 0x1
  100bab5:  call    100b880 <_ZN12OutputStreamlsEj>
  100baba:  mov     edi, 0x1024520 <kout>
  100babf:  call    10079e0 <_ZN10TextStream5flushEv>
  100bac4:  call    1007f00 <_ZN5Guard5leaveEv>
  100bac9:  jmp     100ba98 <_ZN11Application6actionEv+0x18>
```

01024520 <kout>

0100ba80 <_ZN11Application6actionEv>:

```
100ba80:  endbr64
100ba84:  push    rbp
100ba85:  push    rbx
100ba86:  xor     ebx, ebx
100ba88:  sub     rsp, 0x8
100ba8c:  call    1008be0
100ba91:  mov     ebp, eax
100ba93:  nop     DWORD PTR [rax+rax*1+0x0]
100ba98:  call    1007e80
100ba9d:  mov     edx, ebp
100ba9f:  xor     esi, esi
100baa1:  mov     edi, 0x1024584
100baa6:  call    1008d90
100baab:  mov     esi, ebx
100baad:  mov     edi, 0x1024520
100bab2:  add     ebx, 0x1
100bab5:  call    100b880
100baba:  mov     edi, 0x1024520
100babf:  call    10079e0
100bac4:  call    1007f00
100bac9:  jmp     100ba98
```

01024520 <kout>

0100ba80 <_ZN11Application6actionEv>:

100ba80:	endbr64	f3 0f 1e fa
100ba84:	push rbp	55
100ba85:	push rbx	53
100ba86:	xor ebx, ebx	31 db
100ba88:	sub rsp, 0x8	48 83 ec 08
100ba8c:	call 1008be0	e8 4f d1 ff ff
100ba91:	mov ebp, eax	89 c5
100ba93:	nop DWORD PTR [rax+rax*1+0x0]	0f 1f 44 00 00
100ba98:	call 1007e80	e8 e3 c3 ff ff
100ba9d:	mov edx, ebp	89 ea
100ba9f:	xor esi, esi	31 f6
100baa1:	mov edi, 0x1024584	bf 84 45 02 01
100baa6:	call 1008d90	e8 e5 d2 ff ff
100baab:	mov esi, ebx	89 de
100baad:	mov edi, 0x1024520	bf 20 45 02 01
100bab2:	add ebx, 0x1	83 c3 01
100bab5:	call 100b880	e8 c6 fd ff ff
100baba:	mov edi, 0x1024520	bf 20 45 02 01
100babf:	call 10079e0	e8 1c bf ff ff
100bac4:	call 1007f00	e8 37 c4 ff ff
100bac9:	jmp 100ba98	eb cd

01024520 <kout>

```
_ZN11Application6actionEv:  
    endbr64  
    push    rbp  
    push    rbx  
    xor     ebx, ebx  
    sub     rsp, 0x8  
    call    _ZN4Core5getIDEv  
    mov     ebp, eax  
LOOP:  
    call    _ZN5Guard5enterEv  
    mov     edx, ebp  
    xor     esi, esi  
    mov     edi, kout+64  
    call    _ZN10TextWindow6setPosEjj  
    mov     esi, ebx  
    mov     edi, kout  
    add     ebx, 0x1  
    call    _ZN12OutputStreamlsEj  
    mov     edi, kout  
    call    _ZN10TextStream5flushEv  
    call    _ZN5Guard5leaveEv  
    jmp     LOOP
```

```

_ZN11Application6actionEv:
    endbr64
    push    rbp
    push    rbx
    xor     ebx, ebx
    sub     rsp, 0x8
    call    _ZN4Core5getIDEv
    mov     ebp, eax
LOOP:
    call    _ZN5Guard5enterEv
    mov     edx, ebp
    xor     esi, esi
    mov     edi, kout+64
    call    _ZN10TextWindow6setPosEjj
    mov     esi, ebx
    mov     edi, kout
    add     ebx, 0x1
    call    _ZN12OutputStreamlsEj
    mov     edi, kout
    call    _ZN10TextStream5flushEv
    call    _ZN5Guard5leaveEv
    jmp     LOOP

```

Assembler

- Sprache der 2. Gen.
(maschinenorientiert)
- **Mnemonics** statt Bytes
- 1:1-Übersetzung

```

_ZN11Application6actionEv:
    endbr64
    push    rbp
    push    rbx
    xor     ebx, ebx
    sub     rsp, 0x8
    call    _ZN4Core5getIDEv
    mov     ebp, eax
LOOP:
    call    _ZN5Guard5enterEv
    mov     edx, ebp
    xor     esi, esi
    mov     edi, kout+64
    call    _ZN10TextWindow6setPosEjj
    mov     esi, ebx
    mov     edi, kout
    add     ebx, 0x1
    call    _ZN12OutputStreamlsEj
    mov     edi, kout
    call    _ZN10TextStream5flushEv
    call    _ZN5Guard5leaveEv
    jmp     LOOP

```

Assembler

- Sprache der 2. Gen.
(maschinenorientiert)
 - **Mnemonics** statt Bytes
 - 1:1-Übersetzung
 - für x64-Architektur
- Dokumentation im
Intel-Manual, Vol. 2



Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 2 (2A, 2B, 2C & 2D):
Instruction Set Reference, A-Z

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes:
Basic Architecture, Order Number 253665; Instruction Set Reference, A-Z, Order Number 325383;
System Programming Guide, Order Number 325384; Model-Specific Registers, Order Number
335592. Refer to all four volumes when evaluating your design needs.



Syntaxunterschiede bei x86-Assembler

Intel:

`mov edi, 0x17`

(Ziel, Quelle)

`objdump -M intel`

Standard bei `nasm` (Netwide Assembler)

AT&T:

`mov $0x17, %edi`

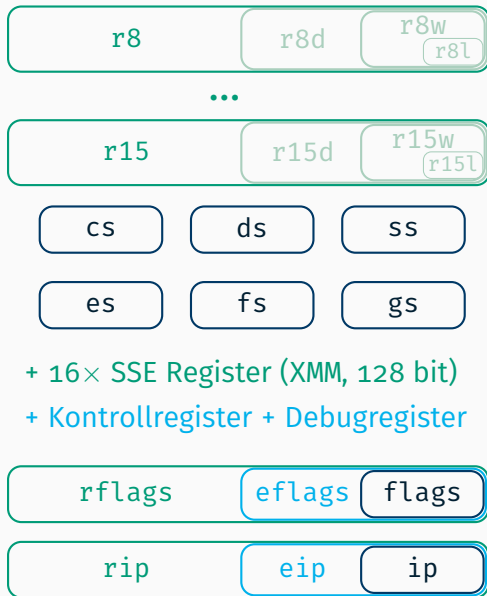
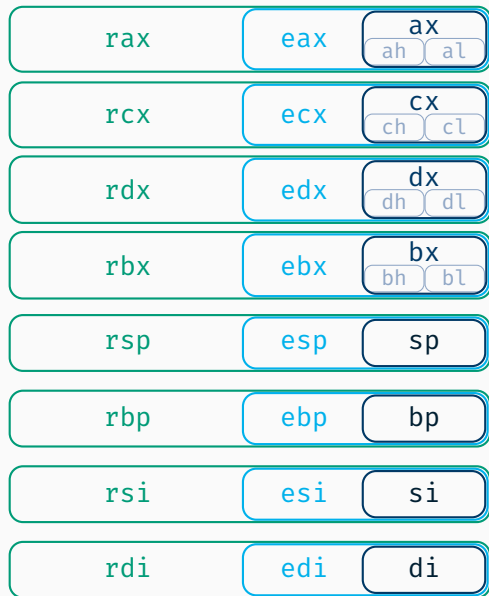
(Quelle, Ziel)

Standard bei `objdump`

und `GCC inline asm`

Berechnungen auf Registern,

Long Mode Register



Berechnungen auf Registern, Zugriff mittels **mov-Instruktion**

Assembler: mov

Berechnungen auf Registern, Zugriff mittels **mov-Instruktion**

Register $\leftarrow$ **Konstante** `mov rax, 42`

Assembler: mov

Berechnungen auf Registern, Zugriff mittels **mov-Instruktion**

Register $\leftarrow$ **Konstante** `mov rax, 42`

Register $\leftarrow$ **Register** `mov rax, rcx`

Assembler: mov

Berechnungen auf Registern, Zugriff mittels **mov-Instruktion**

Register $\leftarrow$ **Konstante** `mov rax, 42`

Register $\leftarrow$ **Register** `mov rax, rcx`

Register $\leftrightarrow$ **Speicher** `mov rax, [0xb8000]`
 `mov [0xb8000 + 4], rax`

`mov rax, [rcx + rdx]`

`mov rax, [rsp]`

~~`mov [0xb8000], [0xb8002]`~~

Besonderheiten bei mov



Besonderheiten bei mov



`mov eax, XX` `eax` wird gesetzt, obere Hälfte **genullt**

Besonderheiten bei mov



`mov eax, XX` `eax` wird gesetzt, obere Hälfte **genullt**

`mov ax, XX` `ax` wird gesetzt, **Rest unverändert**

`mov al, XX` `al` wird gesetzt, **Rest unverändert**

Falls Rest gesetzt werden muss: `movzx`, `movsx`

Details zur Performance bei partiellem Schreiben: → [StackOverflow](#)

Operationen mit dem Stack

Stack $\leftarrow$ **Register** push rax



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.



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

Operationen mit dem Stack

Stack $\leftarrow$ **Register** push rax

Operationen mit dem Stack

Stack $\leftarrow$ **Register** `push rax`



Operationen mit dem Stack

Stack $\leftarrow$ **Register** `push rax`
 `sub rsp, 8`



Operationen mit dem Stack

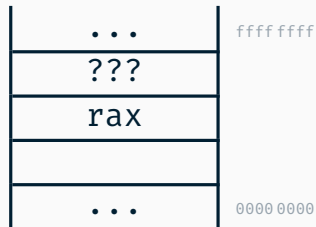
Stack $\leftarrow$ **Register**

```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```

rsp $\rightarrow$



Operationen mit dem Stack

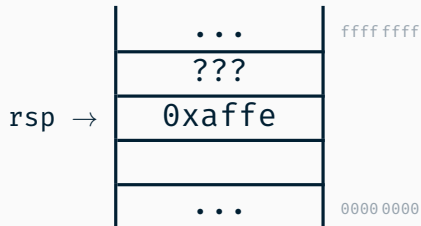
Stack $\leftarrow$ **Register**

```
mov eax, 0xaffe
```

```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```



Operationen mit dem Stack

Stack $\leftarrow$ **Register**

```
mov eax, 0xaffe
```

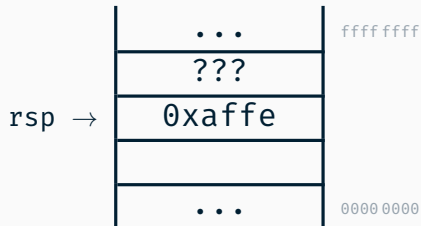
```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```

Register $\leftarrow$ **Stack**

```
pop rcx
```



Operationen mit dem Stack

Stack $\leftarrow$ **Register**

```
mov eax, 0xaffe
```

```
push rax
```

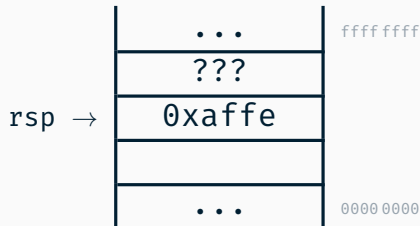
```
sub rsp, 8
```

```
mov [rsp], rax
```

Register $\leftarrow$ **Stack**

```
pop rcx
```

```
mov rcx, [rsp]
```



Operationen mit dem Stack

Stack $\leftarrow$ **Register**

```
mov eax, 0xaffe
```

```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```

Register $\leftarrow$ **Stack**

```
pop rcx
```

```
mov rcx, [rsp]
```

```
add rsp, 8
```



Mathematische Operationen

Addition/Subtraktion	<code>add rax, 4</code>	<code>x += 4;</code>
	<code>sub QWORD [rax], rcx</code>	<code>*x -= rcx;</code>
	<code>inc rax</code>	<code>x++;</code>
	<code>dec rax</code>	<code>x--;</code>

Mathematische Operationen

Addition/Subtraktion	<code>add rax, 4</code>	<code>x += 4;</code>
	<code>sub QWORD [rax], rcx</code>	<code>*x -= rcx;</code>
	<code>inc rax</code>	<code>x++;</code>
	<code>dec rax</code>	<code>x--;</code>
Bit-Operationen	<code>and rax, 0xff</code>	<code>x &= 0xff;</code>
	<code>or rax, [rcx]</code>	<code>x = *rcx;</code>
	<code>xor rax, 0xaa</code>	<code>x ^= 0xaa;</code>
	<code>not rax</code>	<code>x = !x;</code>
	<code>neg rcx</code>	<code>x = ~x;</code>
	<code>shl rax, 1</code>	<code>x <<= 1;</code>
	<code>shr rax, 1</code>	<code>x >>= 1;</code>

Kontrollfluss-Beeinflussung

Sprungmarke LABEL:

 <...>

Einfache Sprünge jmp LABEL

Bedingte Sprünge je/jne/jg/jge/jl/jle/jz LABEL

Kontrollfluss-Beeinflussung

Sprungmarke LABEL:

 <...>

Einfache Sprünge jmp LABEL

Bedingte Sprünge je/jne/jg/jge/jl/jle/jz LABEL

Letztere sind abhängig von den Flags im Statusregister (**rflags**)

Kontrollfluss-Beeinflussung

Sprungmarke LABEL:

<...>

Einfache Sprünge jmp LABEL

Bedingte Sprünge je/jne/jg/jge/jl/jle/jz LABEL

Letztere sind abhängig von den Flags im Statusregister (**rflags**)

Mathematische Operationen und Vergleiche setzen *carry, zero & sign flag*
(Bit 0, 6 & 7 im Statusregister):

Vergleiche cmp rax, rdx

cmp rax, 0x10

$\cong$ sub rax, 0x10 (aber rax unverändert)

Funktionsaufrufe

Funktionsaufruf `call function`
`push <next RIP>`
`jmp function`

Rückkehr aus Funktion `ret`
`pop rip`

Rückkehr aus Interrupt `iretq`
Stellt Hardware-Kontext (`rip`, `cs`,
`rflags`) wieder her

Weiterer 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 ahal
rcx	ecx	cx chcl
rdx	edx	dx dhdl
rbx	ebx	bx bhbl
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 ahal	r9	r9d	r9w r9l
rcx	ecx	cx chcl	r10	r10d	r10w r10l
rdx	edx	dx dhdl	r11	r11d	r11w r11l
rbx	ebx	bx bhbl	r12	r12d	r12w r12l
rsp	esp	sp	r13	r13d	r13w r13l
rbp	ebp	bp	r14	r14d	r14w r14l
rsi	esi	si	r15	r15d	r15w r15l
rdi	edi	di	rflags	eflags	flags
r8	r8d	r8w r8l	rip	eip	ip

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

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

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

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 ahal
rcx	ecx	cx chcl
rdx	edx	dx dhdl
rbx	ebx	bx bhbl
rsp	esp	sp
rbp	ebp	bp
rsi	esi	si
rdi	edi	di
r8	r8d	r8w r8l

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

Rückgabewert

4. Parameter

3. Parameter

2. Parameter

1. Parameter

5. Parameter

Parameterübergabe gemäß Konvention

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

Parameterübergabe gemäß Konvention

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

```
; ggf. Register Sicherung  
push r9
```

```
; 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 r9
```

```
; 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 r9
```

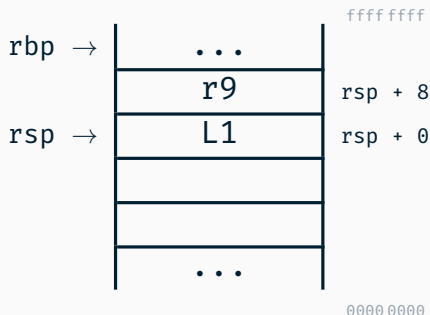
```
; 2. Parameter in Register rsi  
mov esi, 0x2a
```

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

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

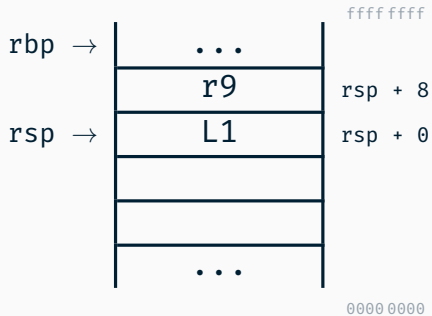
```
L1:
```

Stack beim Funktionsaufruf:



Parameterübergabe gemäß Konvention

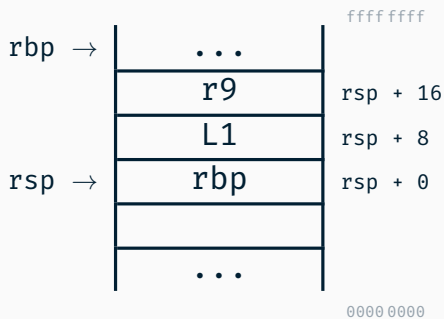
func:



Parameterübergabe gemäß Konvention

func:

```
; alten Rahmenzeiger sichern  
push rbp
```



Parameterübergabe gemäß Konvention

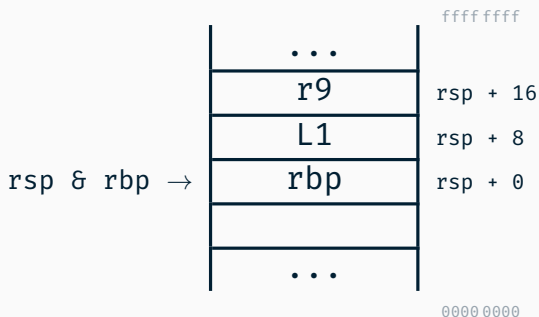
func:

; alten Rahmenzeiger sichern

push rbp

; aktuellen Rahmenzeiger setzen

mov rbp, rsp



Parameterübergabe gemäß Konvention

func:

; alten Rahmenzeiger sichern

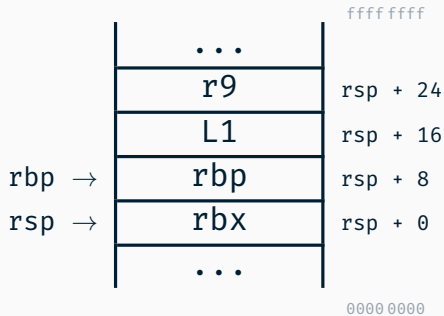
push rbp

; aktuellen Rahmenzeiger setzen

mov rbp, rsp

; ggf. weitere Register sichern

push rbx



Parameterübergabe gemäß Konvention

func:

; alten Rahmenzeiger sichern

push rbp

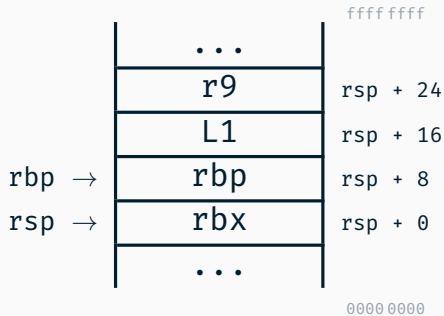
; aktuellen Rahmenzeiger setzen

mov rbp, rsp

; ggf. weitere Register sichern

push rbx

...



Parameterübergabe gemäß Konvention

func:

; alten Rahmenzeiger sichern

push rbp

; aktuellen Rahmenzeiger setzen

mov rbp, rsp

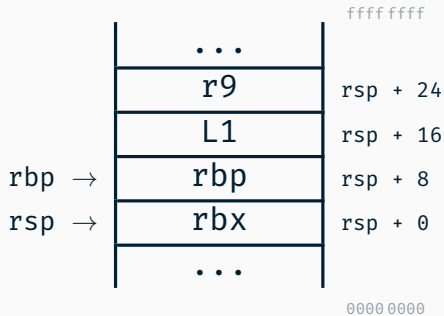
; ggf. weitere Register sichern

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

...

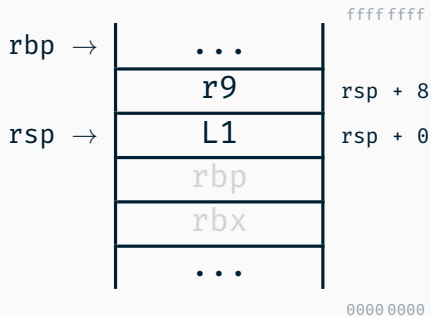
; Rückgabewert nach rax

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

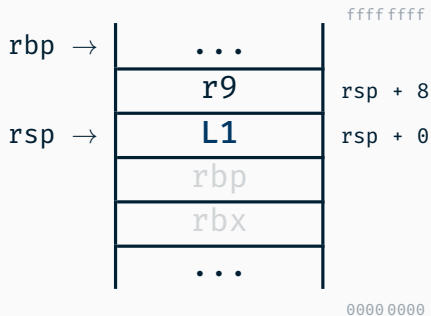
; Register wiederherstellen

pop rbx

pop rbp

; Rücksprung

ret



Parameterübergabe gemäß Konvention

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

```
; ggf. Register Sicherung
```

```
push r9
```

```
; 2. Parameter in Register rsi
```

```
mov esi, 0x2a
```

```
; 1. Parameter in Register rdi
```

```
mov edi, 0x17
```

```
; Funktionsaufruf
```

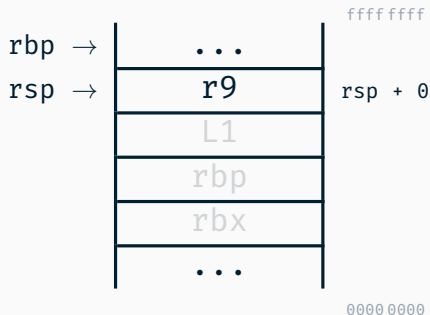
```
call func
```

```
; pushed implizit die
```

```
; Rücksprungadresse
```

```
L1:
```

Stack nach Funktionsaufruf:



Parameterübergabe gemäß Konvention

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

```
; ggf. Register Sicherung
```

```
push r9
```

```
; 2. Parameter in Register rsi
```

```
mov esi, 0x2a
```

```
; 1. Parameter in Register rdi
```

```
mov edi, 0x17
```

```
; Funktionsaufruf
```

```
call func
```

```
; pushed implizit die
```

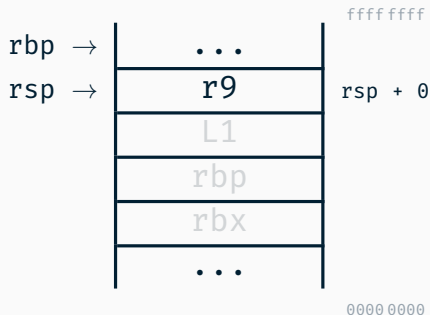
```
; Rücksprungadresse
```

```
L1:
```

```
; Rückgabewert in rax
```

```
mov rsi, rax
```

Stack nach Funktionsaufruf:



Parameterübergabe gemäß Konvention

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

```
; ggf. Register Sicherung
```

```
push r9
```

```
; 2. Parameter in Register rsi
```

```
mov esi, 0x2a
```

```
; 1. Parameter in Register rdi
```

```
mov edi, 0x17
```

```
; Funktionsaufruf
```

```
call func
```

```
; pushed implizit die
```

```
; Rücksprungadresse
```

```
L1:
```

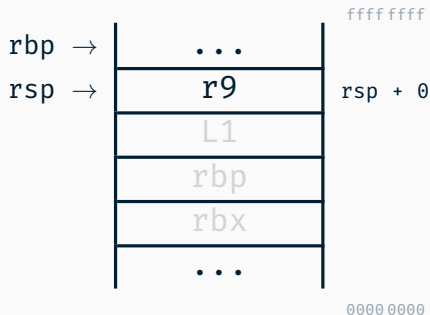
```
; Rückgabewert in rax
```

```
mov rsi, rax
```

```
; ggf. Register wiederherstellen
```

```
pop r9
```

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

[illegible]

11/11/2017

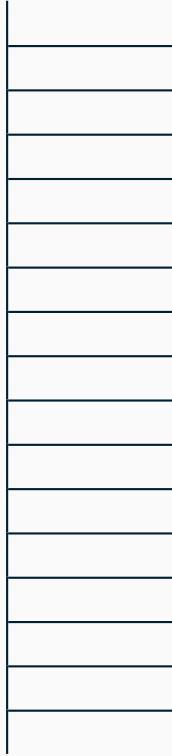
```
// Global sichtbar:
```

```
struct StackPointer {  
    void *kernel;  
  
    // Später auch User-  
    // Stackpointer (in BST)  
};
```

```
StackPointer stack_foo;
```

stack_foo →

0100 bff0



```
// Global sichtbar:
```

```
struct StackPointer {  
    void *kernel;  
  
    // Später auch User-  
    // Stackpointer (in BST)  
};
```

```
StackPointer stack_foo;
```

```
StackPointer stack_bar;
```

stack_foo →

0100 bff0

...

stack_bar →

0100 aef8

```
// Global sichtbar:
```

```
struct StackPointer {  
    void *kernel;  
  
    // Später auch User-  
    // Stackpointer (in BST)  
};
```

```
StackPointer stack_foo;
```

```
StackPointer stack_bar;
```

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

stack_foo → 010d 6aa8 0100 bff0

...

stack_bar → 0110 0188 0100 aef8

```
void foo(){
```

```
...
```

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

```
...
```

rsp →

...

010d 6a68

stack_foo →

010d 6aa8

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

```
void foo(){  
    ...  
  
    // Sicherung der  
    // flüchtigen Register  
    // von Kontext foo
```

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

```
    ...
```

rsp →

...

010d 6a68

stack_foo →

010d 6aa8

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

```
void foo(){  
    ...  
  
    // Sicherung der  
    // flüchtigen Register  
    // von Kontext foo  
    // durch Übersetzer
```

```
    context_switch(  
        &stack_foo,  
        &stack_bar  
    );  
  
    ...
```

rsp →

...

010d 6a68

stack_foo →

010d 6aa8

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

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

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

```
    ...
```

rsp →

r8

stack_foo →

010d 6aa8

stack_bar →

0110 0188

...

010d 6a68

010d 6a60

0100 bff0

...

0100 aef8

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

```
    context_switch(  
        &stack_foo,  
        &stack_bar  
    );  
  
    ...
```

rsp →

r8

010d 6a68

010d 6a60

stack_foo →

010d 6aa8

0100 bff0

stack_bar →

0110 0188

0100 aef8

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

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

LF00:

...

rsp →

stack_foo →

stack_bar →

...

r8

LF00

010d 6aa8

...

0110 0188

010d 6a68

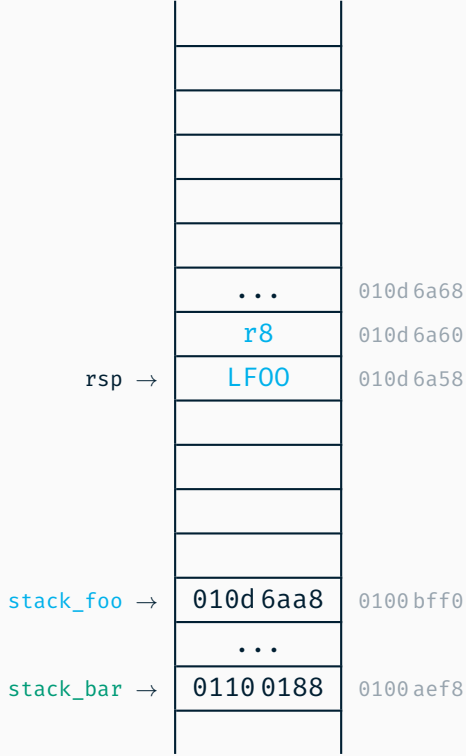
010d 6a60

010d 6a58

0100 bff0

0100 aef8

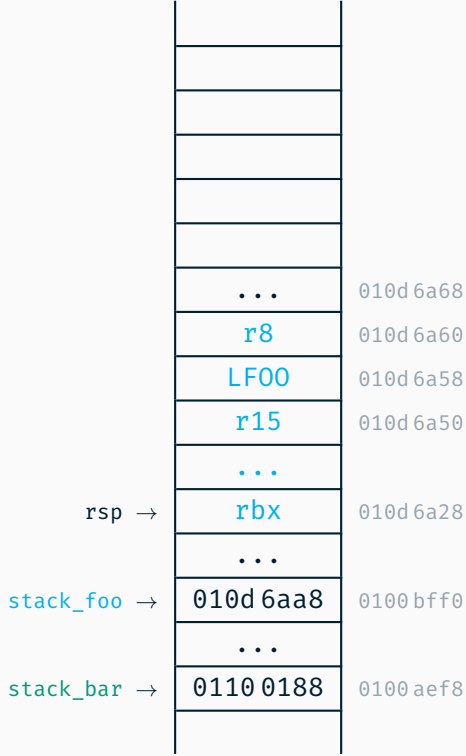
context_switch:



ret

context_switch:

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



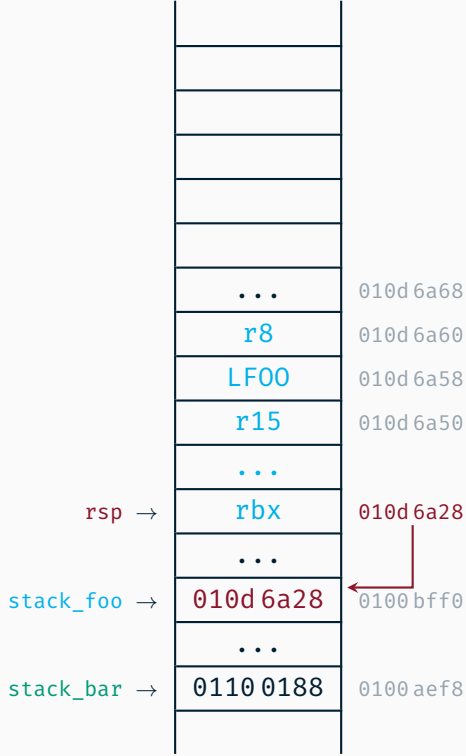
ret

context_switch:

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

```
;; Stackzeiger von  
;; foo sichern
```

ret



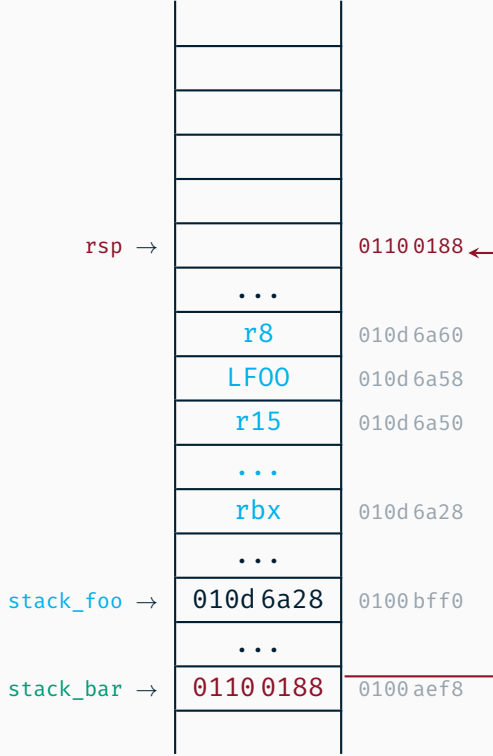
context_switch:

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

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

ret



context_switch:

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

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

ret

	r10	0110 01c0
	LBAR	0110 01b8
	r15	0110 01b0
	...	
rsp →	rbx	0110 0188
	...	
	r8	010d 6a60
	LF00	010d 6a58
	r15	010d 6a50
	...	
	rbx	010d 6a28
	...	
stack_foo →	010d 6a28	0100 bff0
	...	
stack_bar →	0110 0188	0100 aef8

context_switch:

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

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

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

ret

rsp →

stack_foo →

stack_bar →

r10

LBAR

r15

...

rbx

...

r8

LF00

r15

...

rbx

...

010d 6a28

...

0110 0188

0110 01c0

0110 01b8

0110 01b0

0110 0188

010d 6a60

010d 6a58

010d 6a50

010d 6a28

0100 bff0

0100 aef8

context_switch:

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

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

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

ret

rsp →

stack_foo →

stack_bar →

r10

LBAR

r15

...

rbx

...

r8

LF00

r15

...

rbx

...

010d 6a28

...

0110 0188

0110 01c0

0110 01b8

0110 01b0

0110 0188

010d 6a60

010d 6a58

010d 6a50

010d 6a28

0100 bff0

0100 aef8

```
void bar(){
```

```
...
```

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

```
LBAR:
```

```
...
```

rsp →

r10

0110 01c0

LBAR

0110 01b8

r15

0110 01b0

...

rbx

0110 0188

...

r8

010d 6a60

LF00

010d 6a58

r15

010d 6a50

...

rbx

010d 6a28

...

stack_foo →

010d 6a28

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

```
void bar(){
```

```
...
```

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

```
LBAR:
```

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

```
...
```

rsp →

r10

0110 01c0

LBAR

0110 01b8

r15

0110 01b0

...

rbx

0110 0188

...

r8

010d 6a60

LF00

010d 6a58

r15

010d 6a50

...

rbx

010d 6a28

...

stack_foo →

010d 6a28

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

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 1. Parameter

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 1. Parameter

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 1. Parameter

Was wird für jede Koroutine gebraucht?

- eigener, präparierter Stack
- Speicher für Stackzeiger (struct StackPointer)

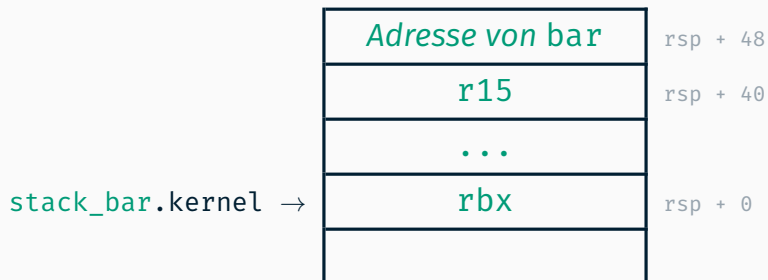
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`

<i>Adresse von bar</i>
r15
...
rbx

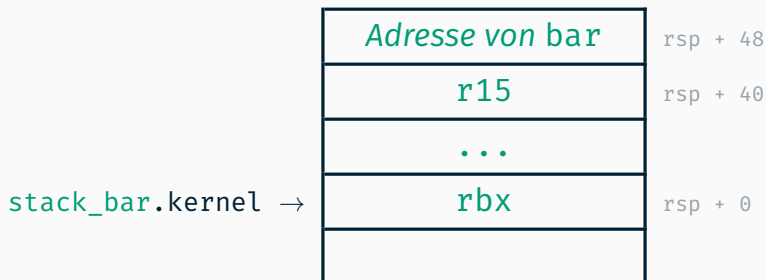
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



Stack aufsetzen

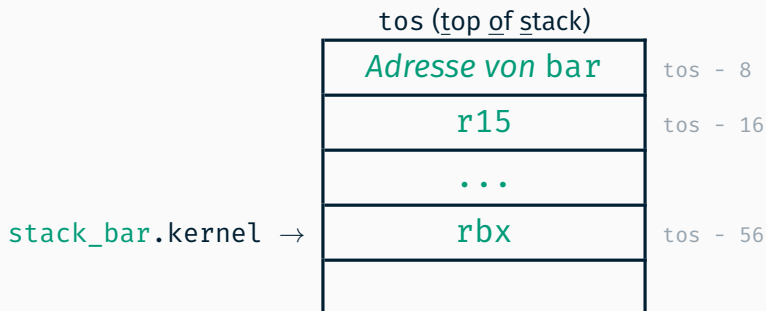
Stack für Einsprung in Koroutine `void bar()`



```
#define STACKSIZE 4096
char stackBar[STACKSIZE];
void* tos = stackFoo + STACKSIZE;
```

Stack aufsetzen

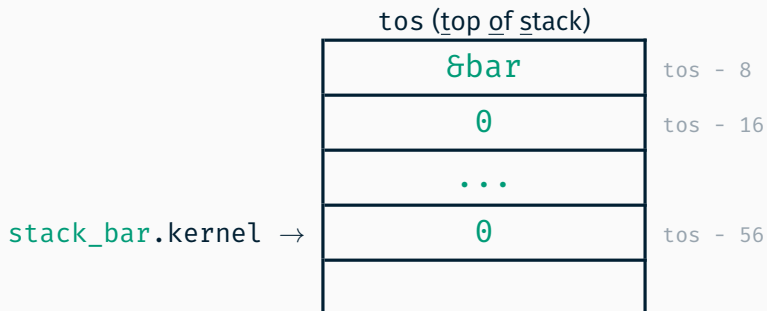
Stack für Einsprung in Koroutine `void bar()`



```
#define STACKSIZE 4096
char stackBar[STACKSIZE];
void* tos = stackFoo + STACKSIZE;
```

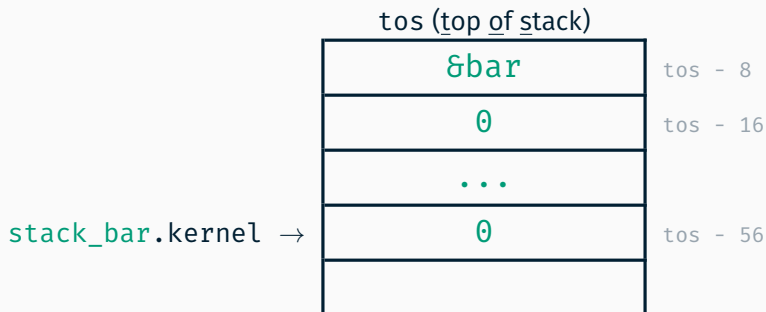
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



Stack aufsetzen

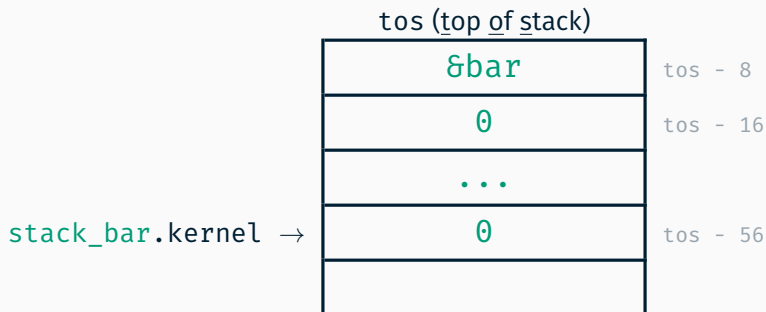
Stack für Einsprung in Koroutine `void bar()`



Was passiert nun bei `context_switch`?

Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



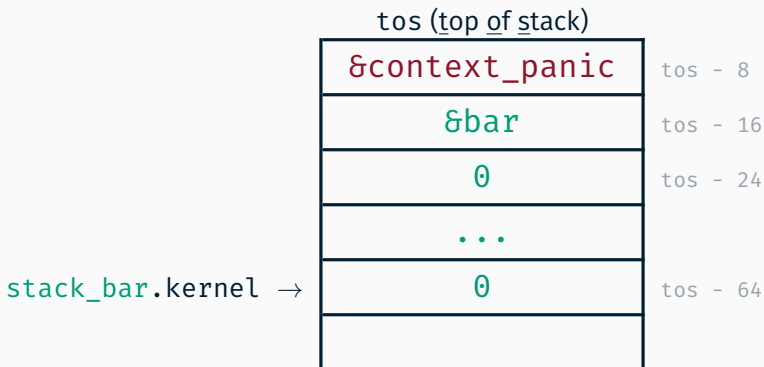
Was passiert wenn die Koroutine `bar` abgearbeitet ist?

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)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `bar_wrapper`

Stack aufsetzen mit Funktionsparameter

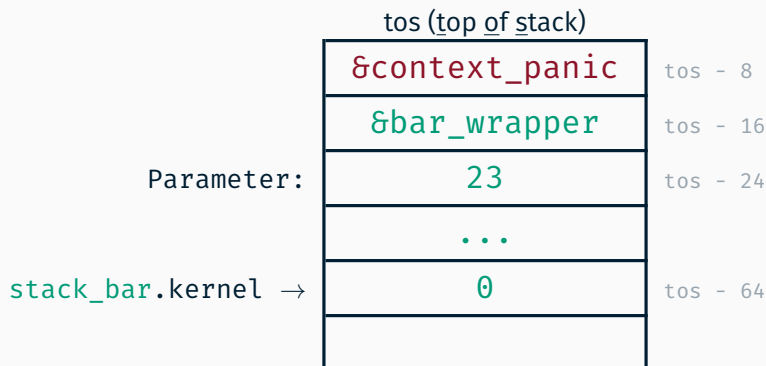
Stack für Einsprung in Koroutine `void bar(int i)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `bar_wrapper`

- 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)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `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 {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Threads

```
class Thread {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

Threads

```
class Thread {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
Thread * x = &app;  
x->action(); // Foo::action oder Bar::action ?
```

```
class Foo : public Thread {  
    void action() { ... }  
};
```

```
class Bar : public Thread {  
    void action() { ... }  
};
```

Threads

```
class Thread {  
    StackPointer sp;  
public:  
    Thread();  
    virtual void action() = 0;  
};
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

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

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                      void (*kickoff)(void*),  
                      void * param);
```

Präpariert einen Stack für den ersten Einsprung

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                      void (*kickoff)(void*),  
                      void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher tos als Stack aufsetzen

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                     void (*kickoff)(void*),  
                     void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher tos als Stack aufsetzen
- nach dem Einsprung soll kickoff mit param aufgerufen werden

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                     void (*kickoff)(void*),  
                     void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher tos als Stack aufsetzen
- nach dem Einsprung soll kickoff mit param aufgerufen werden
- Stackpointer kernel in Thread entsprechend setzen

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                     void (*kickoff)(void*),  
                     void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher tos als Stack aufsetzen
- nach dem Einsprung soll kickoff mit param aufgerufen werden
- Stackpointer kernel in Thread entsprechend setzen
- in C++ (statt Assembler)

Threads: Stack aufsetzen

```
void * prepareContext(void * tos,  
                     void (*kickoff)(void*),  
                     void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher tos als Stack aufsetzen
- nach dem Einsprung soll kickoff mit param aufgerufen werden
- Stackpointer kernel in Thread entsprechend setzen
- in C++ (statt Assembler)
- Pointerarithmetik ist hilfreich

```
#define STACKSIZE 4096  
char stackBar[STACKSIZE];  
char stackFoo[STACKSIZE];
```

```
#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];
```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →		0102 601f
stackFoo[4094] →		0102 601e
stackFoo[4093] →		0102 601d
stackFoo[4092] →		0102 601c
stackFoo[4091] →		0102 601b
stackFoo[4090] →		0102 601a
stackFoo[4089] →		0102 6019
stackFoo[4088] →		0102 6018
stackFoo[4087] →		0102 6017
stackFoo[4086] →		0102 6016
stackFoo[4085] →		0102 6015
stackFoo[4084] →		0102 6014
stackFoo[4083] →		0102 6013
stackFoo[4082] →		0102 6012
stackFoo[4081] →		0102 6011
stackFoo[4080] →		0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000

```
#define STACKSIZE 4096
```

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

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

```
void* tos = stackFoo + STACKSIZE;
```

```
void** rsp = (void**) tos;
```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →		0102 601f
stackFoo[4094] →		0102 601e
stackFoo[4093] →		0102 601d
stackFoo[4092] →		0102 601c
stackFoo[4091] →		0102 601b
stackFoo[4090] →		0102 601a
stackFoo[4089] →		0102 6019
stackFoo[4088] →		0102 6018
stackFoo[4087] →		0102 6017
stackFoo[4086] →		0102 6016
stackFoo[4085] →		0102 6015
stackFoo[4084] →		0102 6014
stackFoo[4083] →		0102 6013
stackFoo[4082] →		0102 6012
stackFoo[4081] →		0102 6011
stackFoo[4080] →		0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

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

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

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →		0102 6017
stackFoo[4086] →		0102 6016
stackFoo[4085] →		0102 6015
stackFoo[4084] →		0102 6014
stackFoo[4083] →		0102 6013
stackFoo[4082] →		0102 6012
stackFoo[4081] →		0102 6011
stackFoo[4080] →		0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

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

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

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

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

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

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

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

// ...

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000



Stapelüberlauf

- Die Stacks sind nur 4K groß



Stapelüberlauf

- Die Stacks sind nur 4K groß
- Die Stacks liegen (oft) hintereinander



Stapelüberlauf

- Die Stacks sind nur 4K groß
- Die Stacks liegen (oft) hintereinander
- Das gilt auch für die initialen Stacks beim Systemstart



Stapelüberlauf

- Die Stacks sind nur 4K groß
 - Die Stacks liegen (oft) hintereinander
 - Das gilt auch für die initialen Stacks beim Systemstart
- Mechanismus zum Erkennen von Überlaufen hilfreich („Stack Canary“)

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

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

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

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

// ...

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020
	...	0000 0000

```

#define STACKSIZE 4096
char stackBar[STACKSIZE];
char stackFoo[STACKSIZE];

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

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

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

// ...

```

	...	ffff ffff
stackBar[1] →	aa	0102 6021
stackBar[0] →	55	0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →	aa	0102 5021
stackFoo[0] →	55	0102 5020
	...	0000 0000

Scheduler

Zentrale Verwaltung der Koroutinen (Threads)

Zentrale Verwaltung der Koroutinen (Threads)

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

```
// 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()  
StackPointer dummy;  
context_switch(dummy, appfoo.sp)
```

AppFoo

```
kickoff(&appfoo)  
appfoo->action()  
kout << "foo" << endl  
scheduler.resume()  
context_switch(appfoo.sp, appbar.sp)
```

```
context_switch(appbar.sp, appfoo.sp)  
kout << "foo" << endl  
scheduler.resume()  
context_switch(appfoo.sp, appbar.sp)
```

```
context_switch(appbar.sp, appfoo.sp)
```

AppBar

```
kickoff(&appbar)  
appbar->action()  
kout << "bar" << endl  
scheduler.resume()
```

```
kout << "bar" << endl  
scheduler.resume()
```

foo

bar

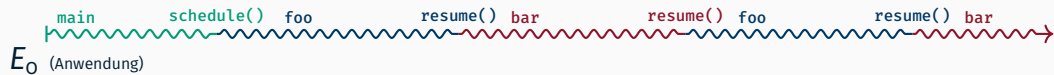
foo

bar

E_0 (Anwendung)

$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)



$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)

OOStuBS immer synchrone Aufrufe
→ Konsistenz gesichert

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

- Scheduling auf der Epilogebe

main()

//...

guard.enter()

scheduler.schedule()

StackPointer dummy;

context_switch(dummy, appfoo.sp)

AppFoo

kickoff(&appfoo)

guard.leave()

appfoo->action()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

context_switch(appbar.sp, appfoo.sp)

guard.leave()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

AppBar

kickoff(&appbar)

guard.leave()

appbar->action()

kout << "bar" << endl

guard.enter()

scheduler.resume()

guard.leave()

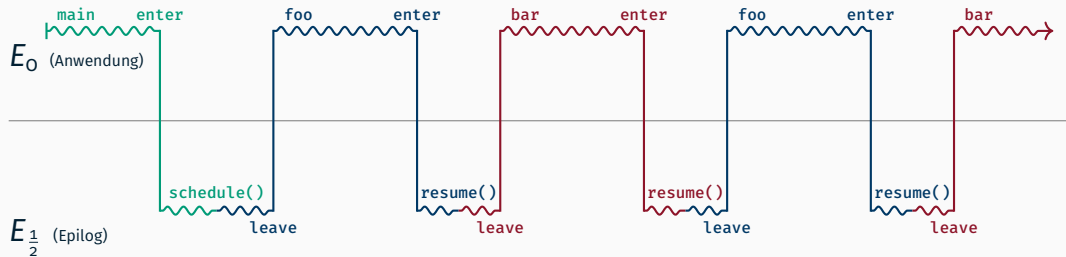
kout << "bar" << endl

foo

bar

foo

bar



E_1 (IRQ/Prolog)

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

- Scheduling auf der Epilogebe

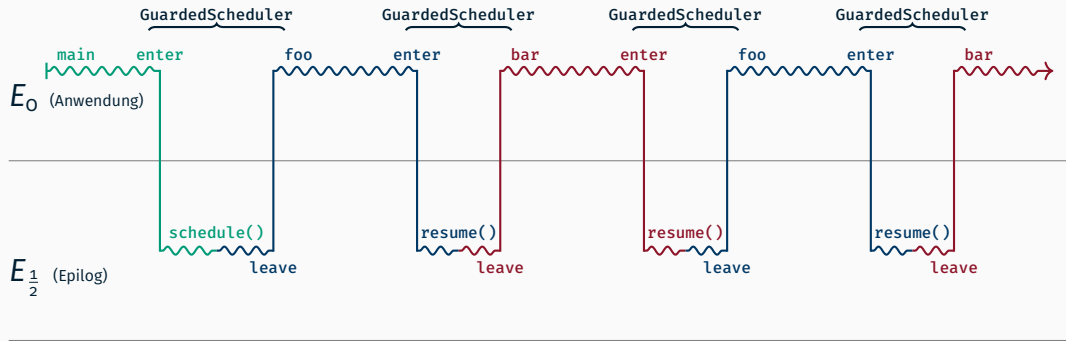
(→ automatische Synchronisation der Ready-Liste)

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

- Scheduling auf der Epilogebeane
(→ automatische Synchronisation der Ready-Liste)
- Kapselung in `GuardedScheduler`
(→ „Systemaufruf“ für Anwendungen)



E_1 (IRQ/Prolog)



Bild: twemoji

Fragen?

Bild: twemoji



Fragen?

Letzte Aufgabe des Jahres 2021,
wir wünschen euch ein frohes Fest!
(Abgabe bis zur nächsten Tafelübung)

Bild: twemoji

