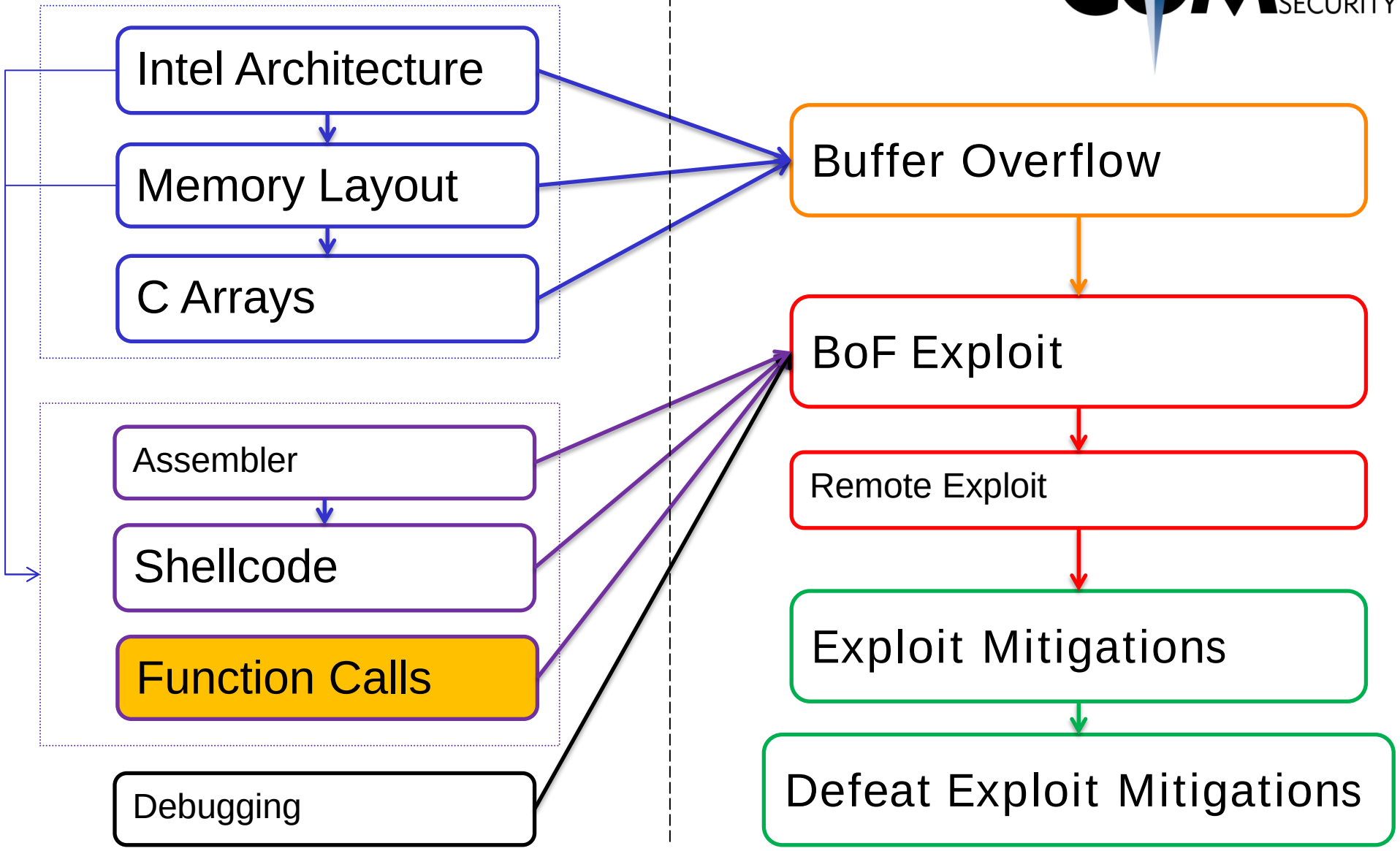
Function Call Convention

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Content


Intel Architecture
Memory Layout
C Arrays
Assembler
Shellcode
Function Calls
Debugging
Buffer Overflow
BoF Exploit
Remote Exploit
Exploit Mitigations
Defeat Exploit Mitigations

Function Call Convention

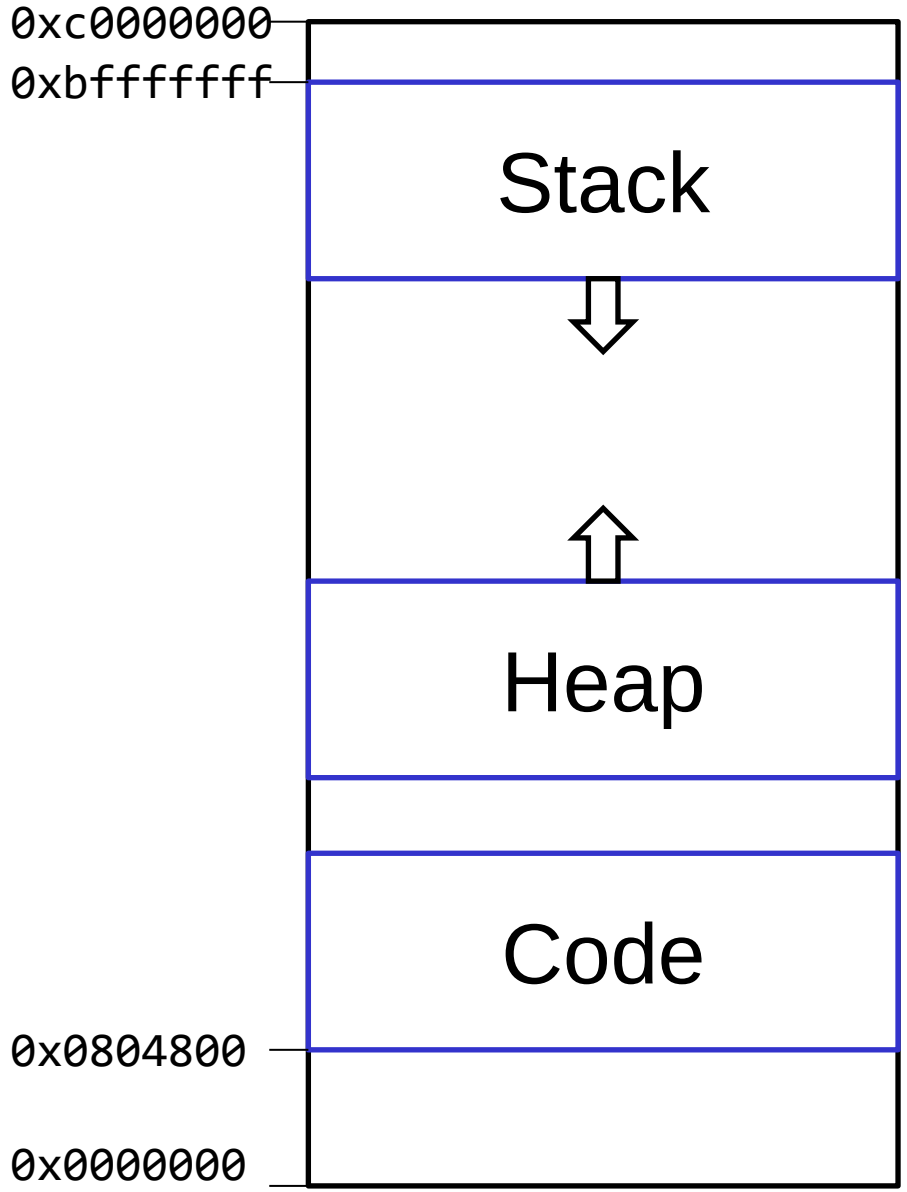
Function call convention:

- How functions work
- Program-metadata on the stack

Stack based buffer overflow:

- Overwrite program-metadata on the stack

x32 Memory Layout



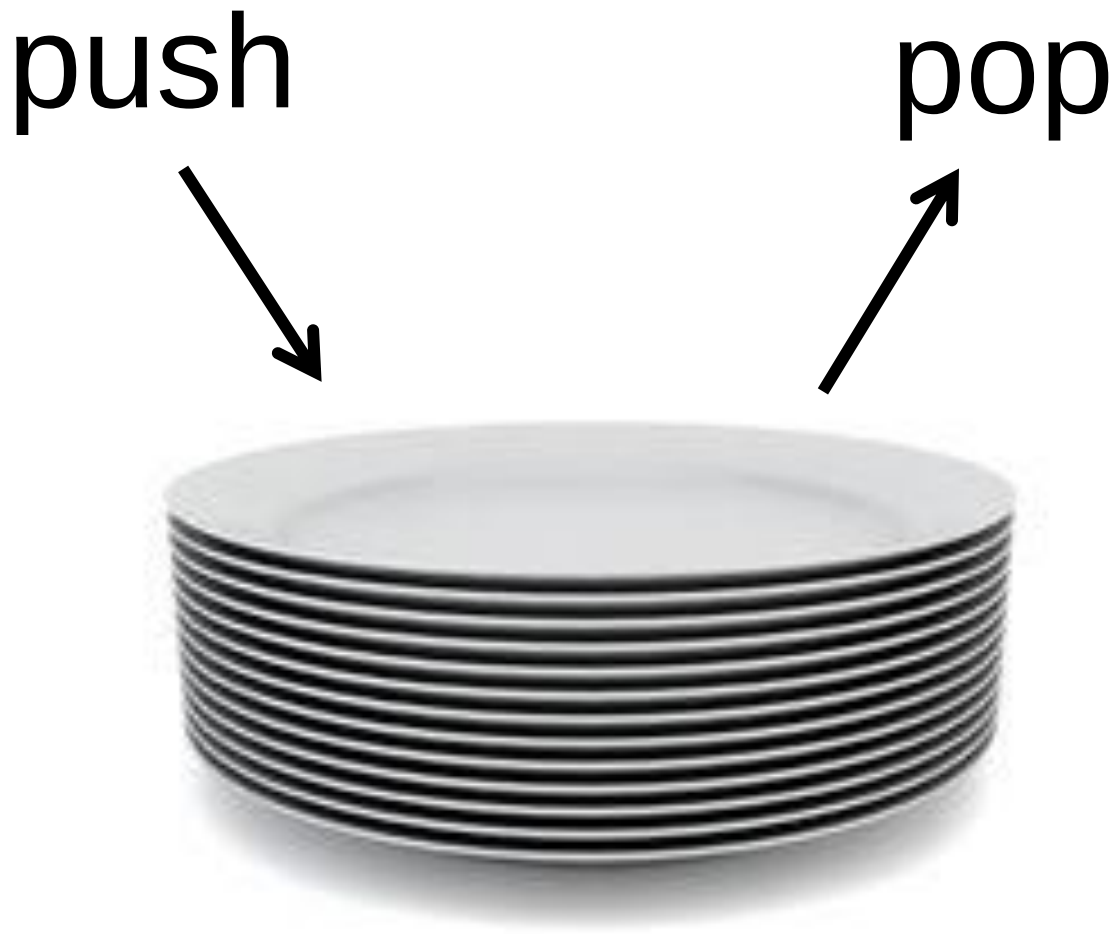
Stacks

How do they work?

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Stack



Stack

0x10000

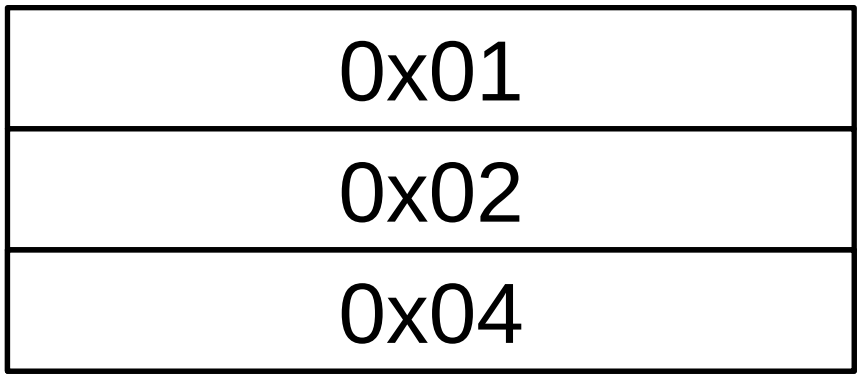
0x00010

push

pop

Stack

```
Push 0x1
Push 0x2
Push 0x3
Pop
Push 0x4
```



x32 Call Convention

Functions and the Stack

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

What is a function?

- Self contained subroutine
- Re-usable
- Can be called from anywhere
- After function is finished: Jump to the calling function (calee)

x32 Call Convention

```
void main(void) {
  int blubb = 0;
  foobar(blubb);
  return;
}

void foobar (int arg1) {
  char compass1[];
  char compass2[];
}
```

What does the function foobar() need?

- Function Argument:
  - blubb
- Local variables
  - Compass1
  - Compass2
- And: Address of next instruction in main()
  - &return

x32 Call Convention

Saved IP (&__libc_start)

Saved Frame Pointer

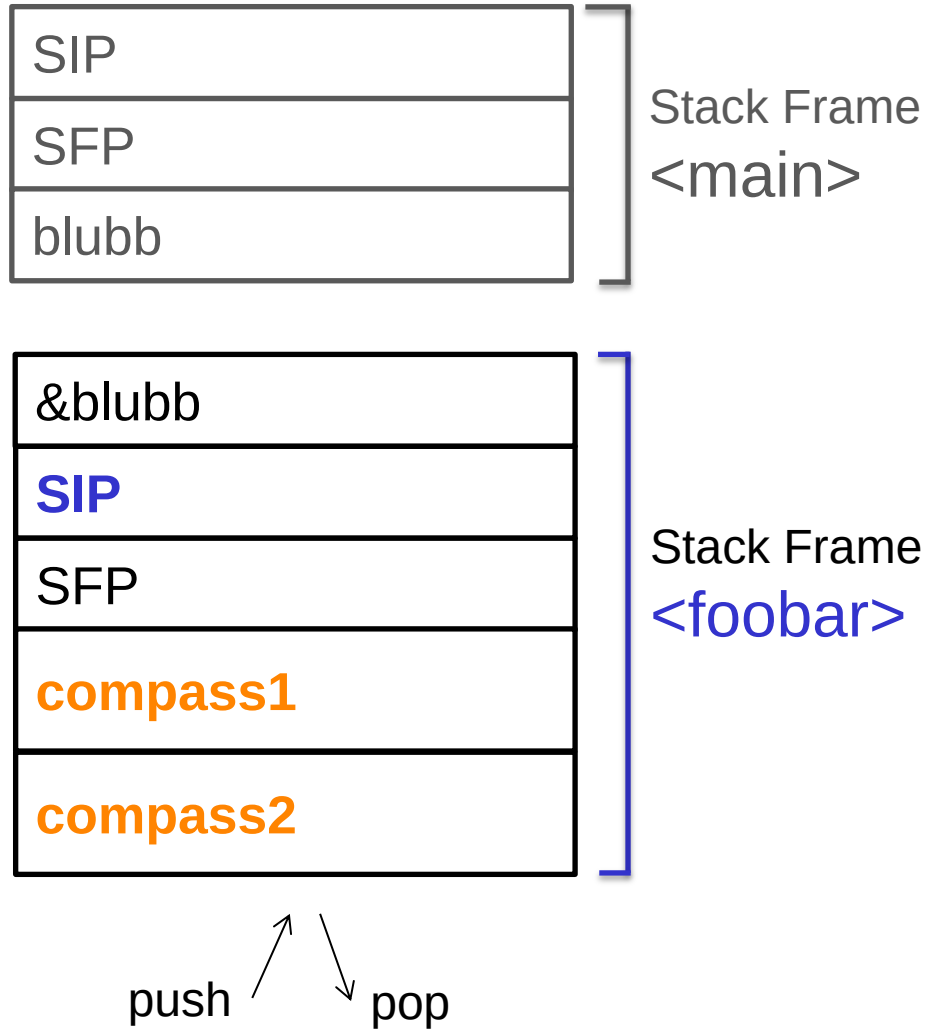
Local Variables <main>

Argument for <foobar>

Saved IP (&return)

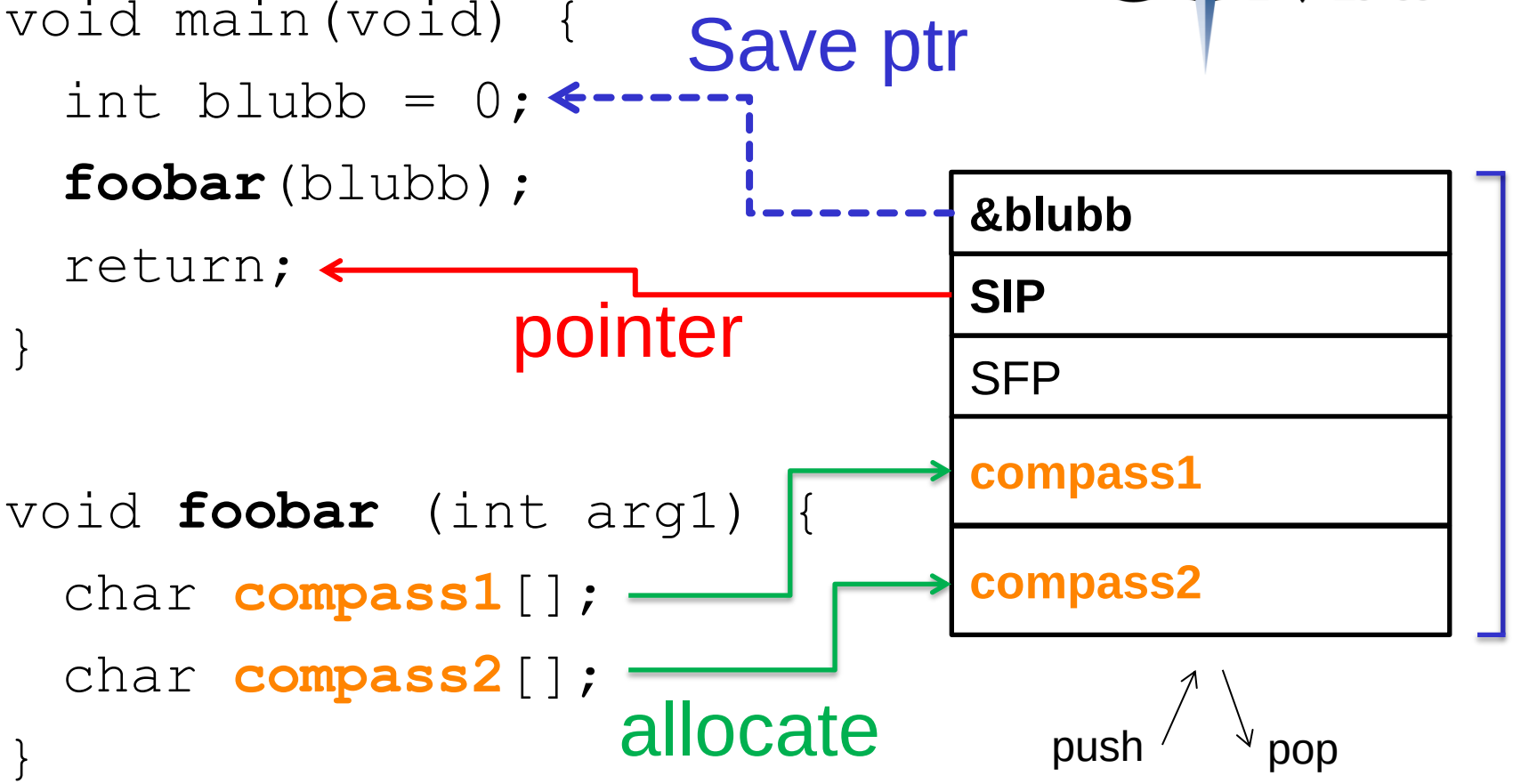
Saved Frame Pointer

Local Variables <foobar>



x32 Call Convention

```
void main(void) {
  int blubb = 0;
  foobar(blubb);
  return;
}

void foobar (int arg1) {
  char compass1[];
  char compass2[];
}
```



x32 Call Convention

Saved IP (&__libc_start)

Saved Frame Pointer

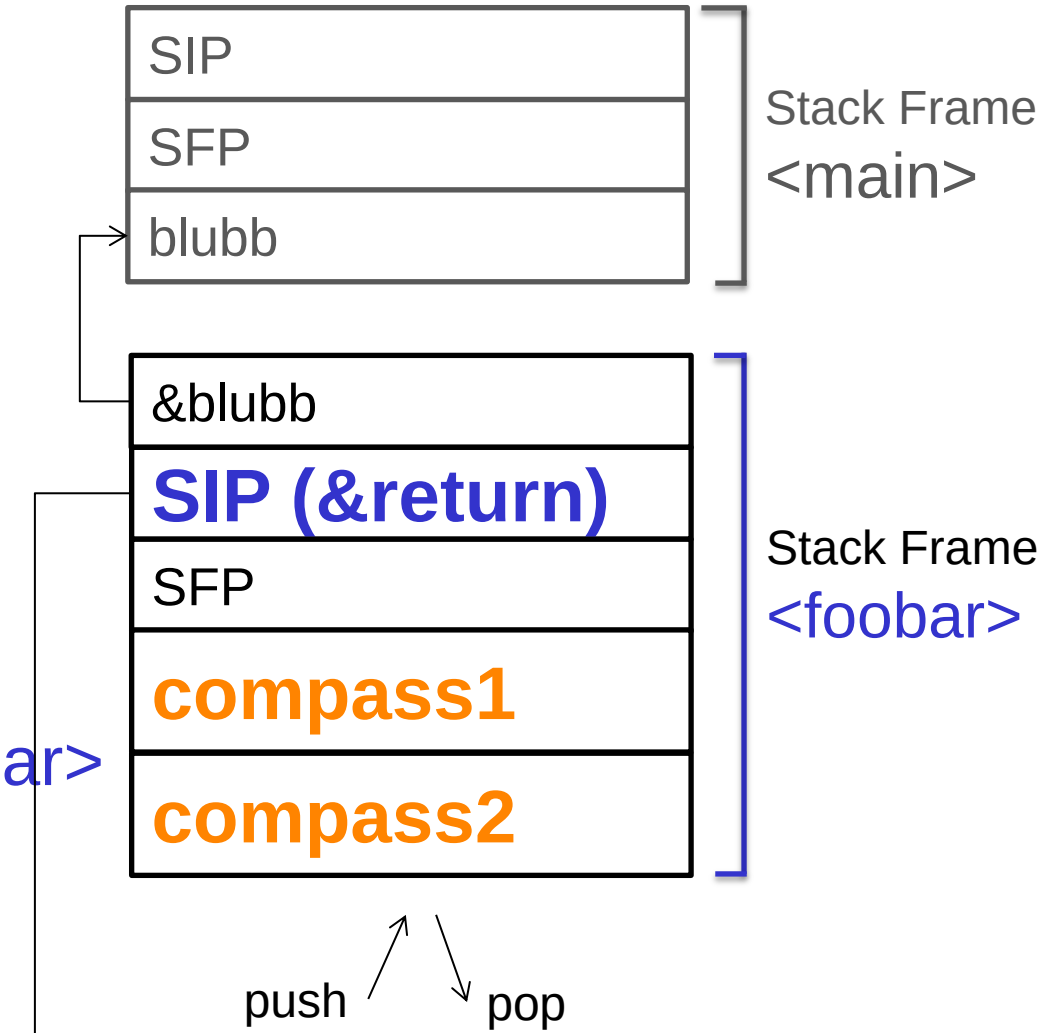
Local Variables <main>

Argument for <foobar>

Saved IP (&return)

Saved Frame Pointer

Local Variables <foobar>



SIP: Stored Instruction Pointer

- Copy of EIP
- Points to the address where control flow continues after end of function
  - (return, ret)
- Usually points into the code section

x32 Call Convention

Attention! Assembler ahead!

- AT&T vs Intel syntax

Intel syntax:

```
mov     eax,1
mov     ebx,0ffh
int     80h
```

AT&T syntax:

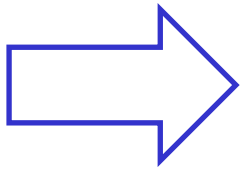
```
movl    $1,%eax
movl    $0xff,%ebx
int     $0x80
```

Don’t hang me if I messed this up somewhere

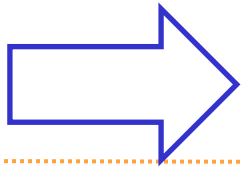
x32 Call Convention

In ASM:

```
call 0x11223344 <&foobar>
```



```
push EIP
jmp 0x11223344

<function code> (0x11223344)

ret
```

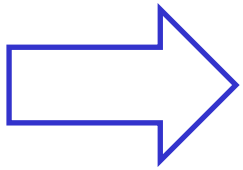


```
pop eip
```

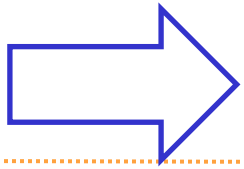
x32 Call Convention

In ASM:

```
call 0x11223344 <&foobar>
```



```
push EIP
jmp 0x11223344
mov ebp, esp
<function code>
mov esp, ebp
ret
```



```
pop eip
```

x32 Call Convention

In ASM:

```
call 0x11223344 <&foobar>
     push EIP
     jmp 0x11223344
mov ebp, esp
```

Prolog

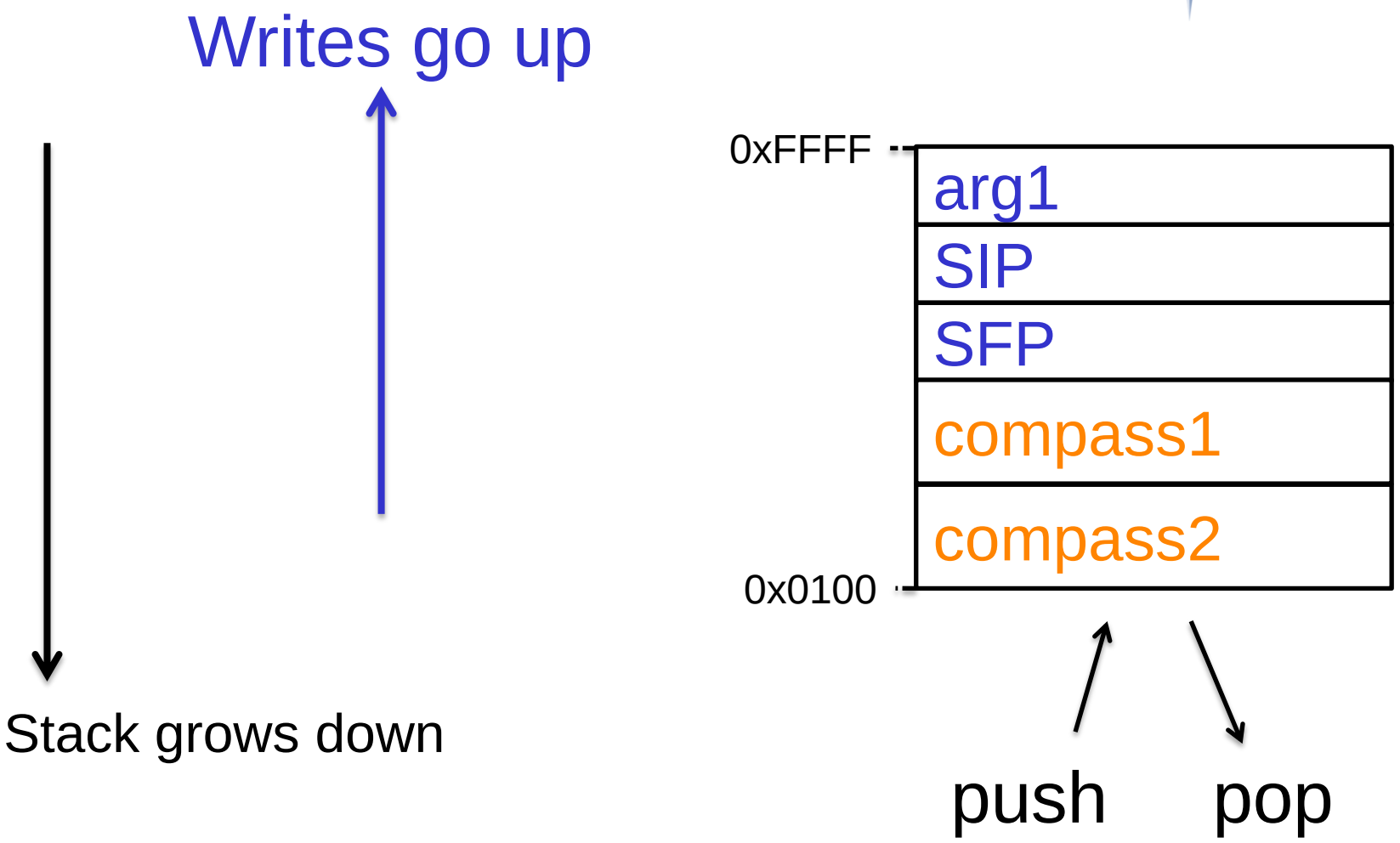
```
<function code>
```

Function

```
mov esp, ebp
ret
     pop eip
```

Epilog

x32 Call Convention



x32 Call Convention

Recap:

- User data is on the stack
- Also: important stuff is on the stack (Instruction Pointer, SIP)
- Stack grows down ⇩
- Writes go up ⇧

x32 Call Convention Details

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

x32 Call Convention Details

```
int add(int x, int y) {
  int sum;
  sum = x + y;
  return sum;
}
```

x32 Call Convention Details

```
c = add(3, 4)
```

C

```
push 4

push 3

call add
```

ASM

```
push 4

push 3

push EIP

jmp add
```

ASM, detailed

x32 Call Convention Details

```
push 4
push 3
push EIP
jmp add
```

add():

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov eax, DWORD PTR [ebp + 0xc]
mov edx, DWORD PTR [ebp + 0x8]
add eax, edx
mov DWORD PTR [ebp – 0x04], eax
mov eax, DWORD PTR [ebp – 0x04]

leave
ret
```

x32 Call Convention Details

```
push 4
push 3
push EIP
jmp add
```

add():

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov eax, DWORD PTR [ebp + 0xc]
mov edx, DWORD PTR [ebp + 0x8]
add eax, edx
mov DWORD PTR [ebp – 0x04], eax
mov eax, DWORD PTR [ebp – 0x04]

mov esp, ebp    ; leave
pop ebp         ; leave
ret
```

x32 Call Convention Details

```
push 4
push 3
push EIP
jmp add
```

add():

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov eax, DWORD PTR [ebp + 0xc]
mov edx, DWORD PTR [ebp + 0x8]
add eax, edx
mov DWORD PTR [ebp – 0x04], eax
mov eax, DWORD PTR [ebp – 0x04]

mov esp, ebp    ; leave
pop ebp         ; leave
pop eip         ; ret
```

x32 Call Convention Details

```
push 4
push 3
push EIP
jmp add
```

add():

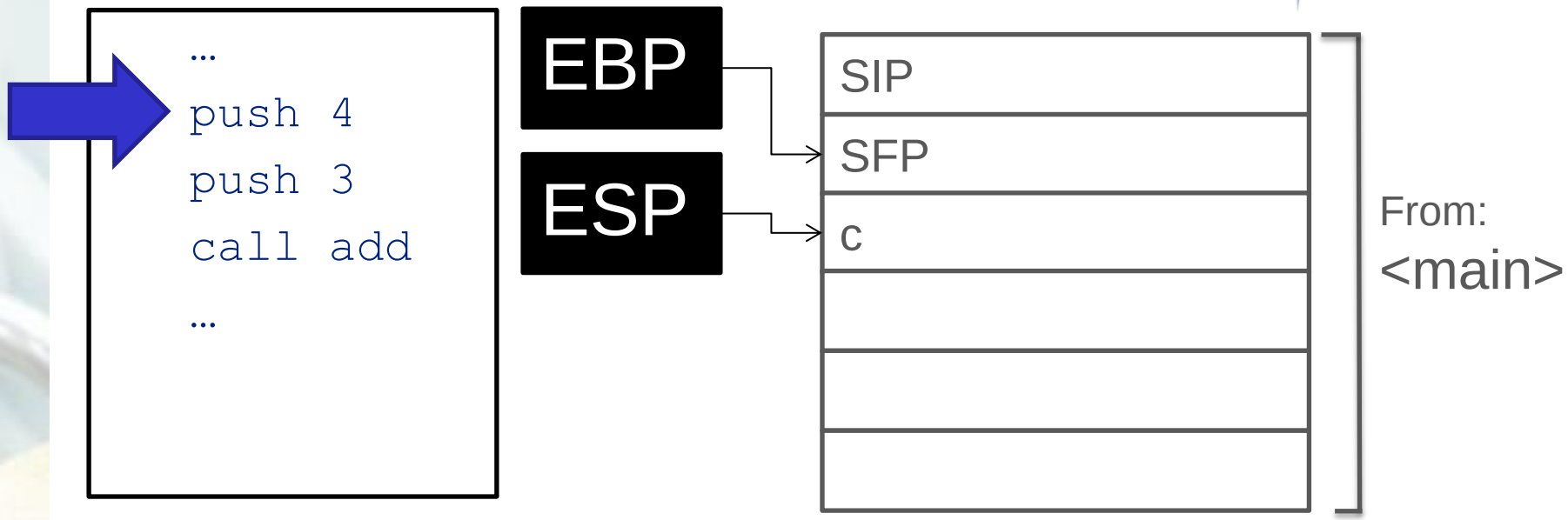
```
push ebp
mov ebp, esp,
sub esp, 0x10




mov esp, ebp    ; leave
pop ebp         ; leave
pop eip         ; ret
```

Function Prolog

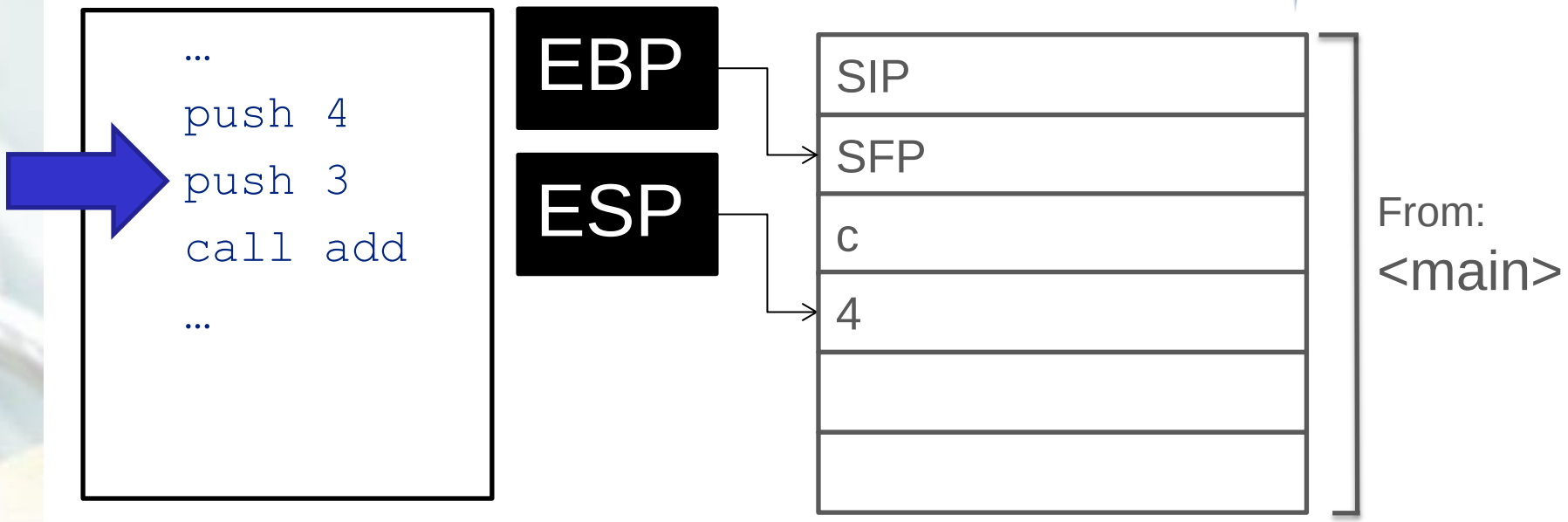
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

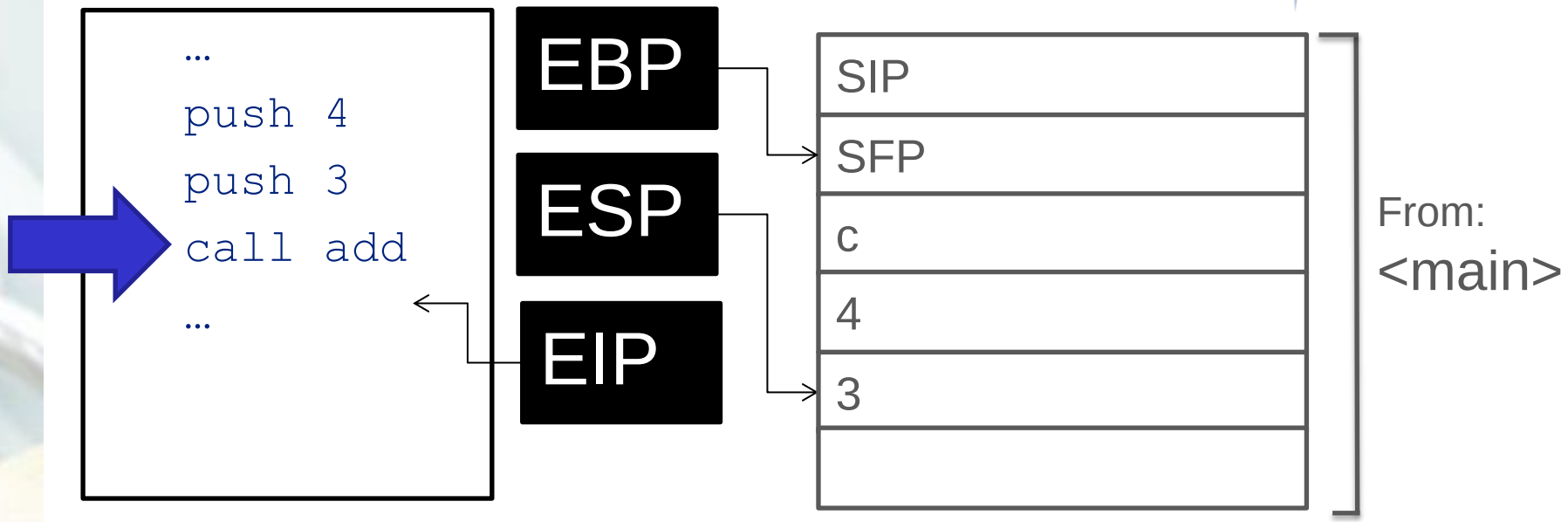
x32 Call Convention - Function Prolog



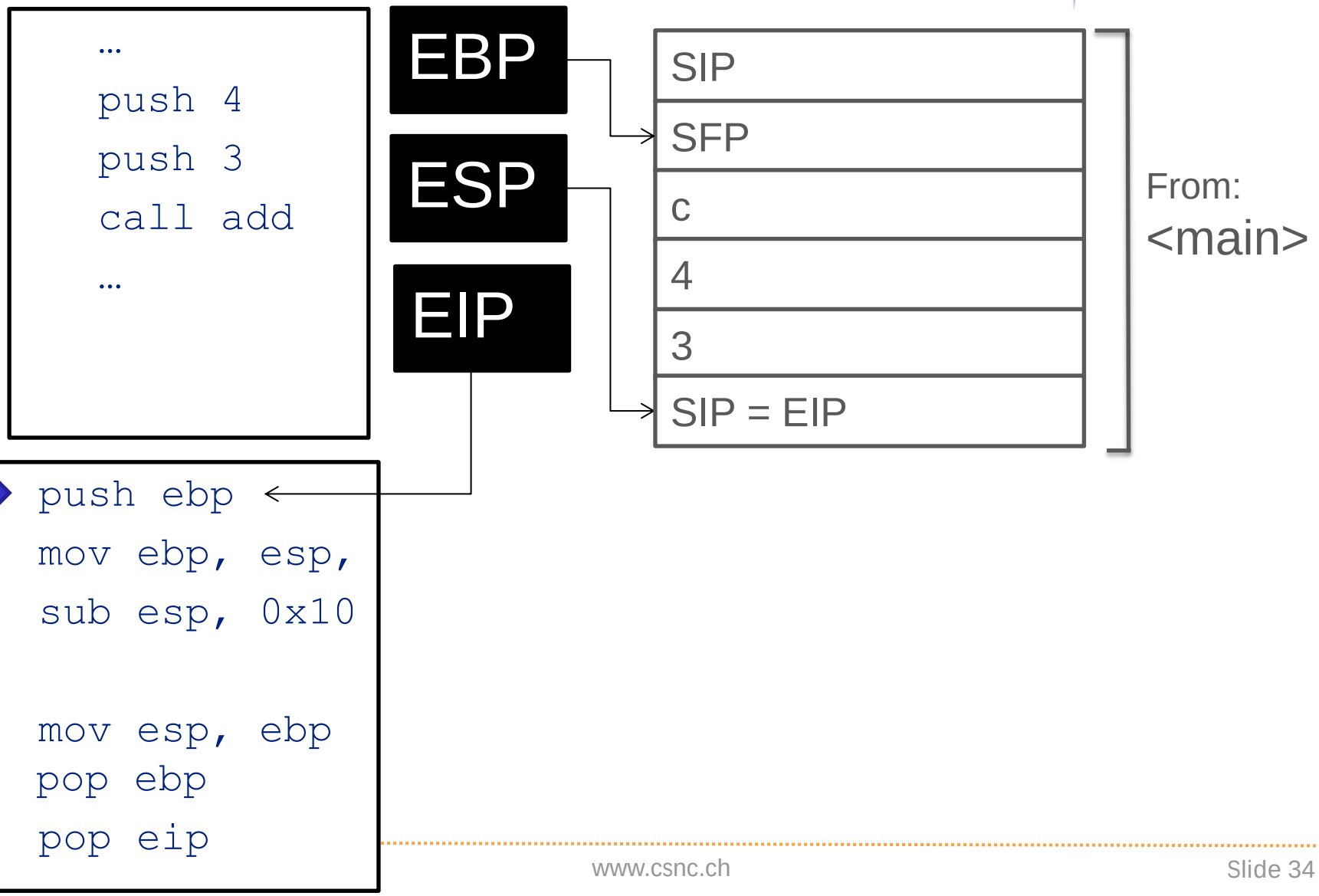
x32 Call Convention - Function Prolog



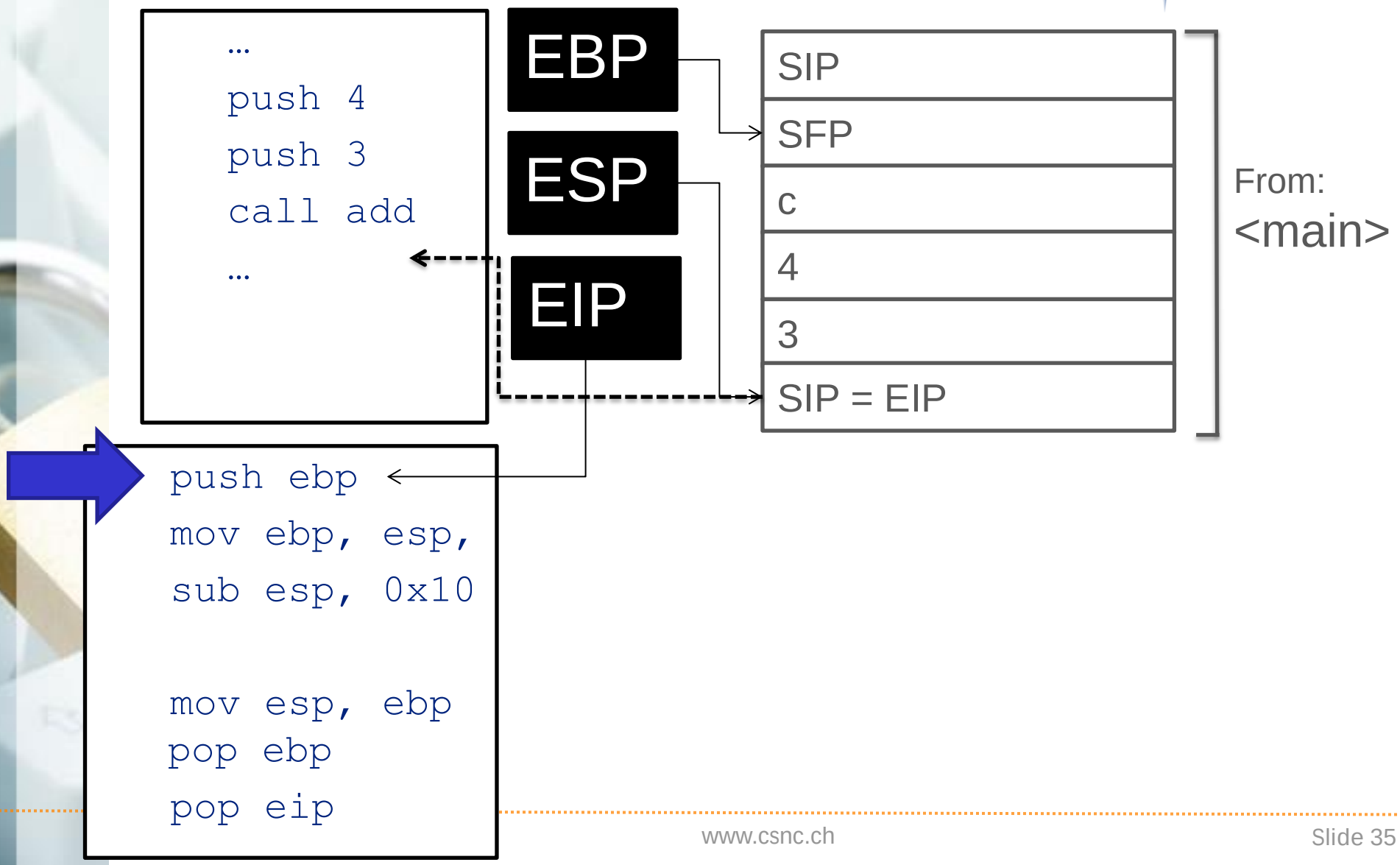
x32 Call Convention - Function Prolog



x32 Call Convention - Function Prolog

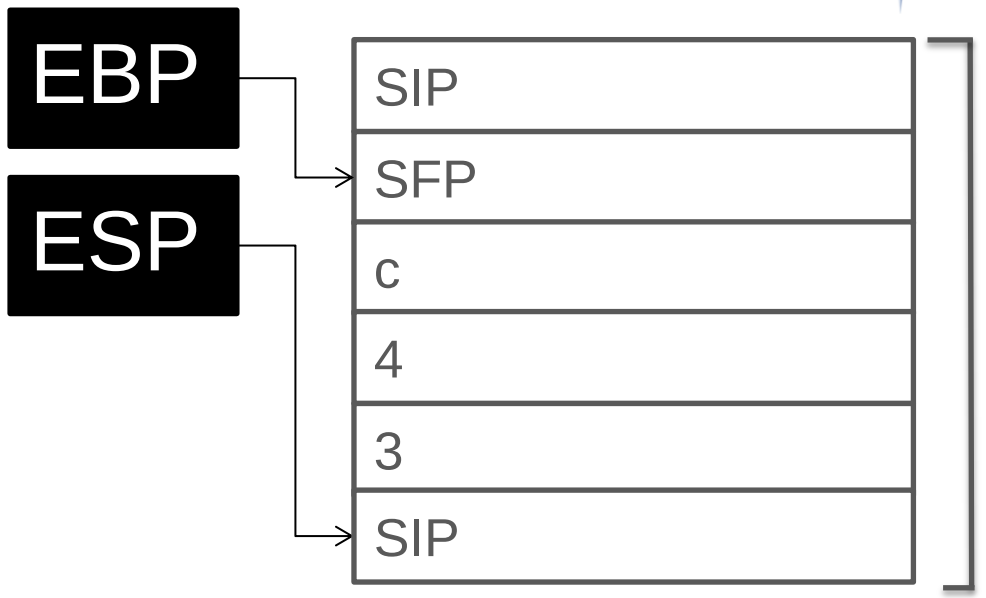


x32 Call Convention - Function Prolog



x32 Call Convention - Function Prolog

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```

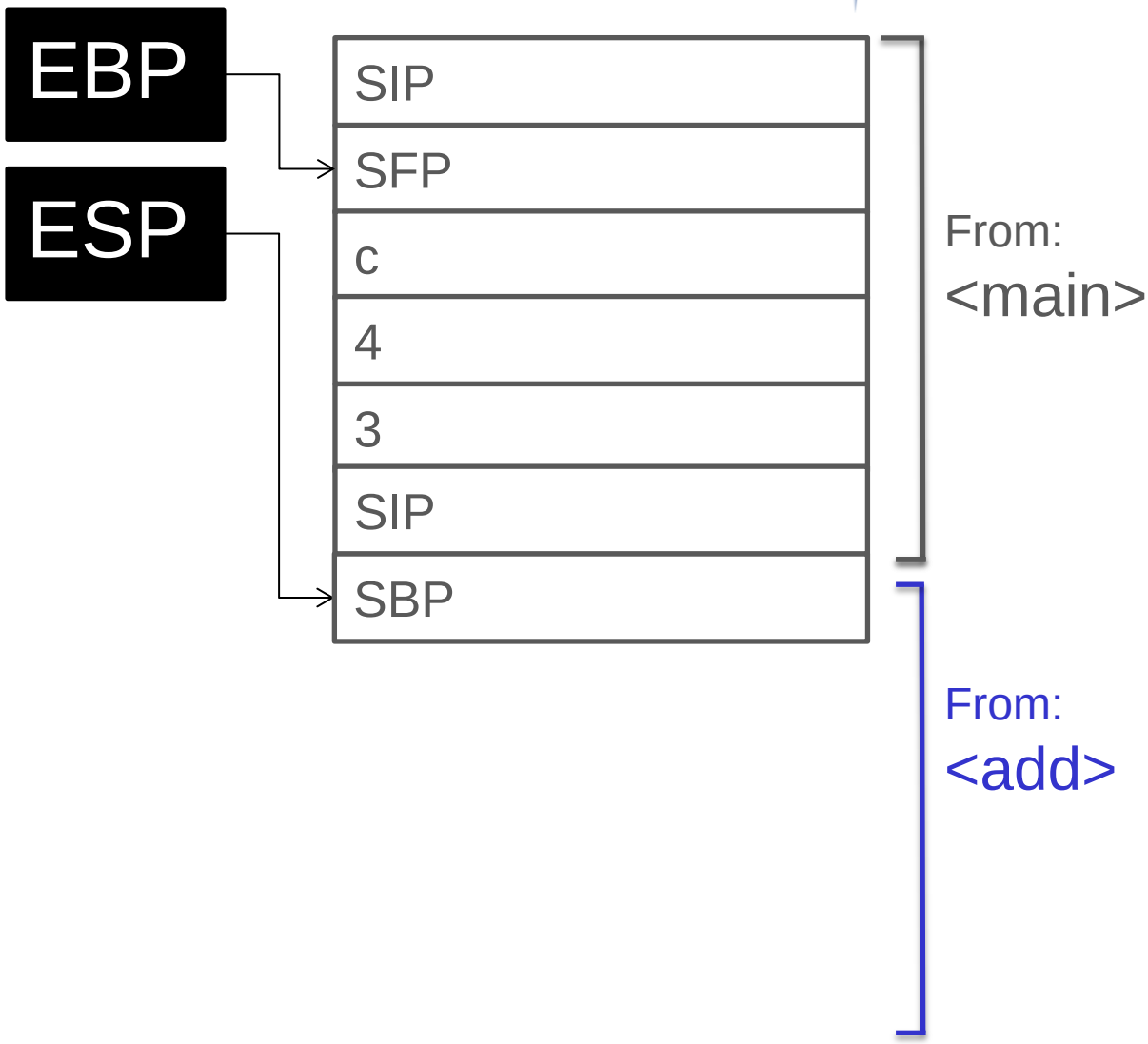


From:
<main>

From:
<add>

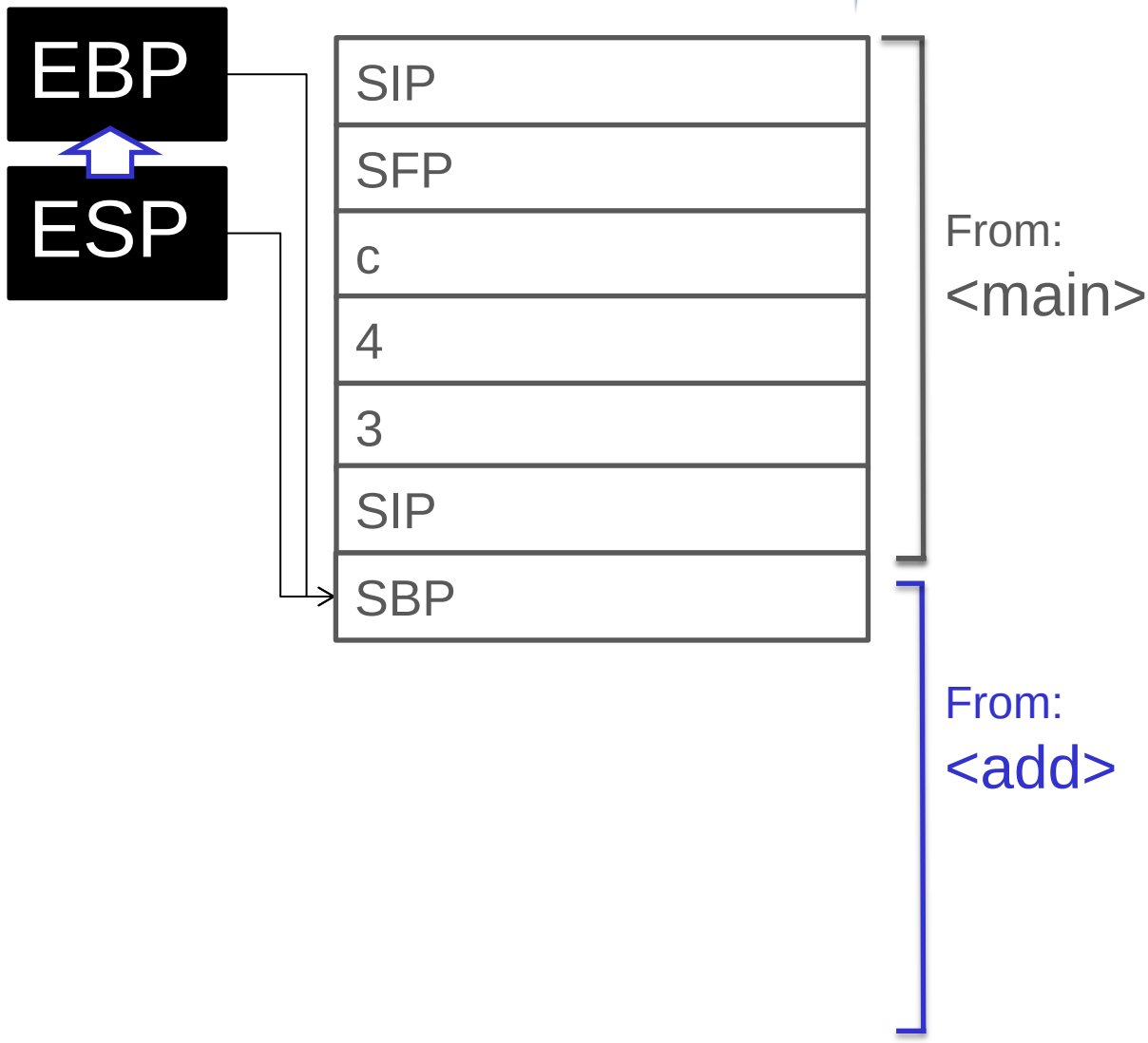
x32 Call Convention - Function Prolog

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```



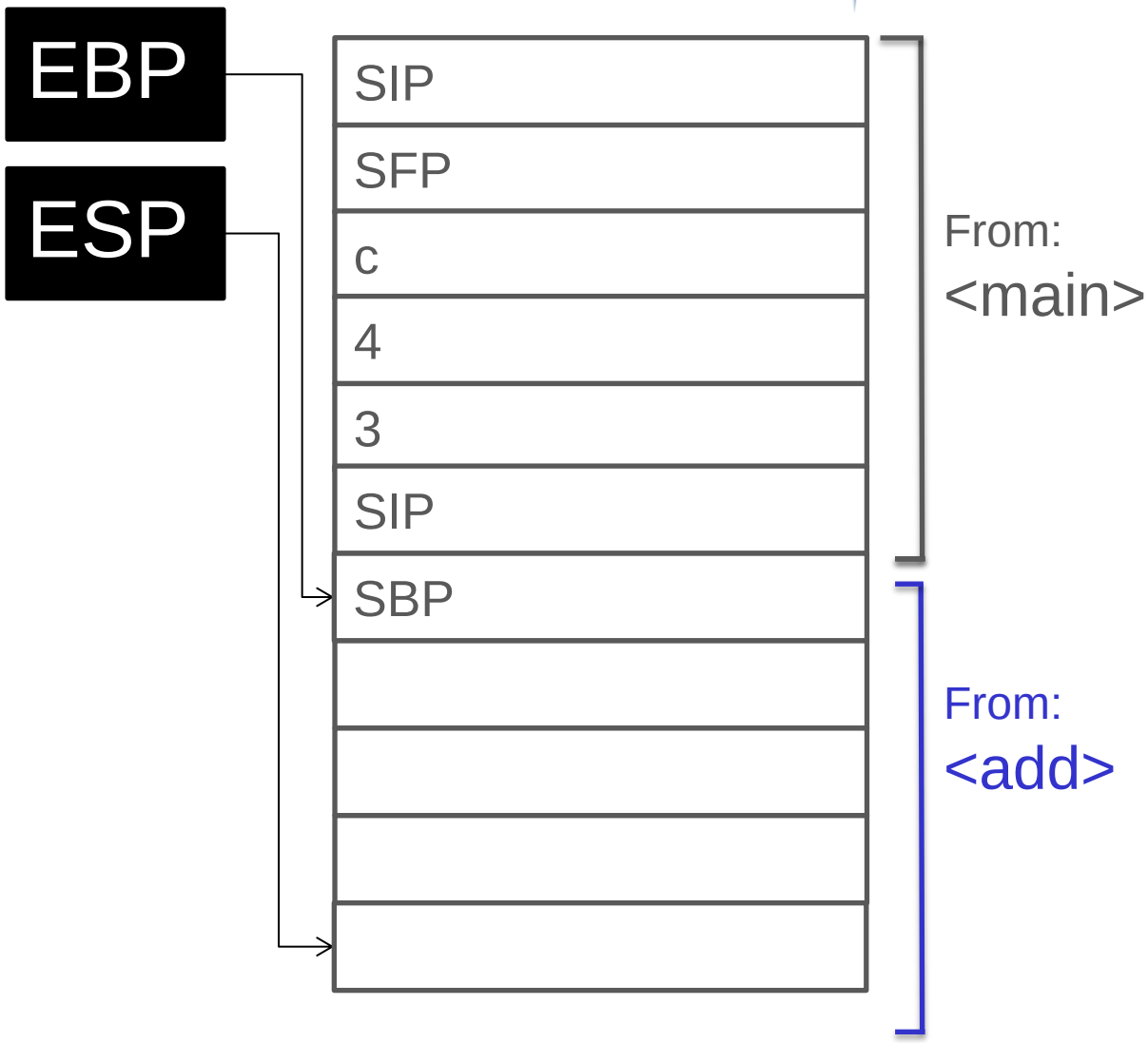
x32 Call Convention - Function Prolog

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```



x32 Call Convention - Function Prolog

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```



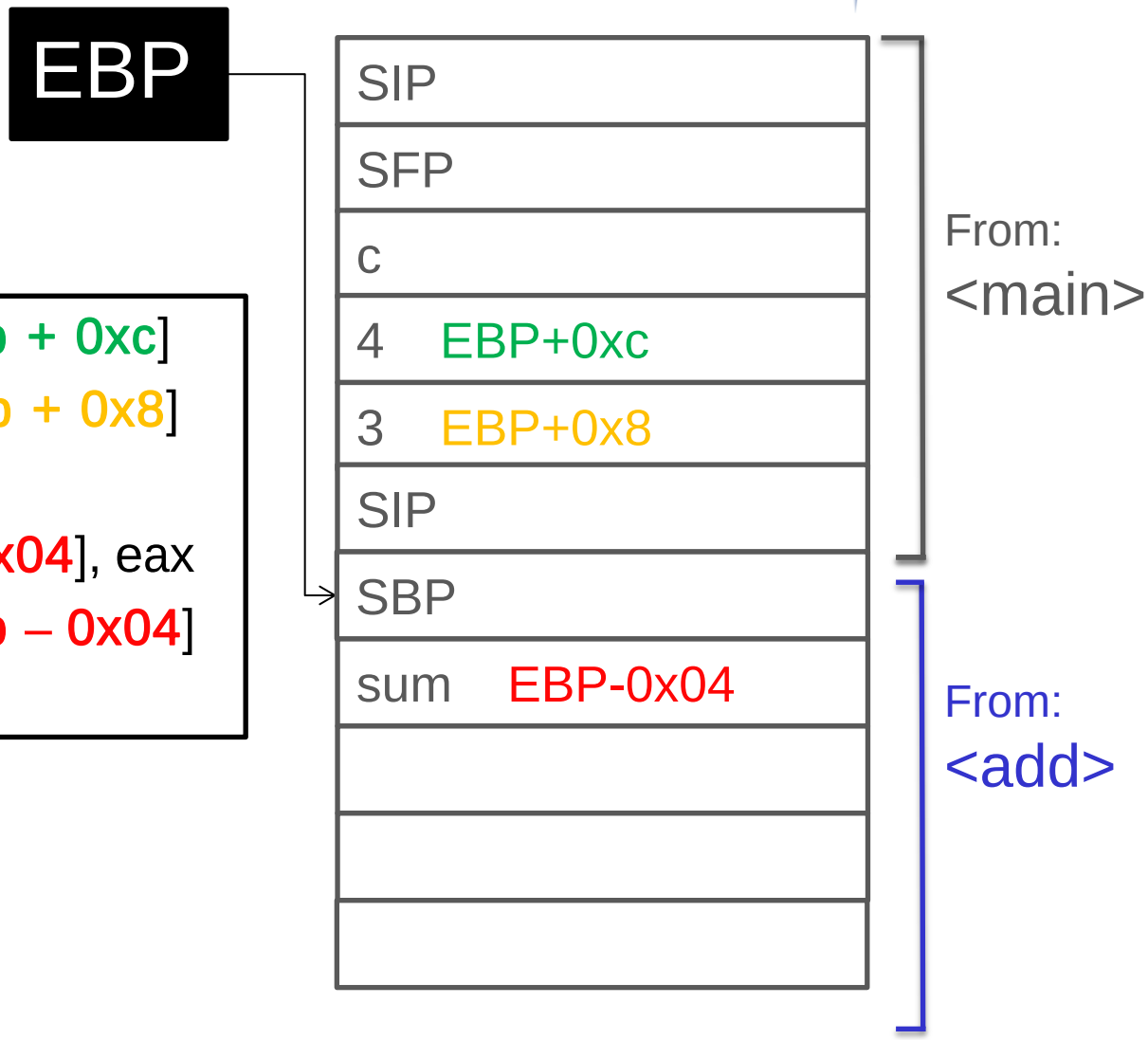
Execute Function

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

x32 Call Convention - Execute Function

```
mov eax, DWORD PTR [ebp + 0xc]
mov edx, DWORD PTR [ebp + 0x8]
add eax, edx
mov DWORD PTR [ebp – 0x04], eax
mov eax, DWORD PTR [ebp – 0x04]
```



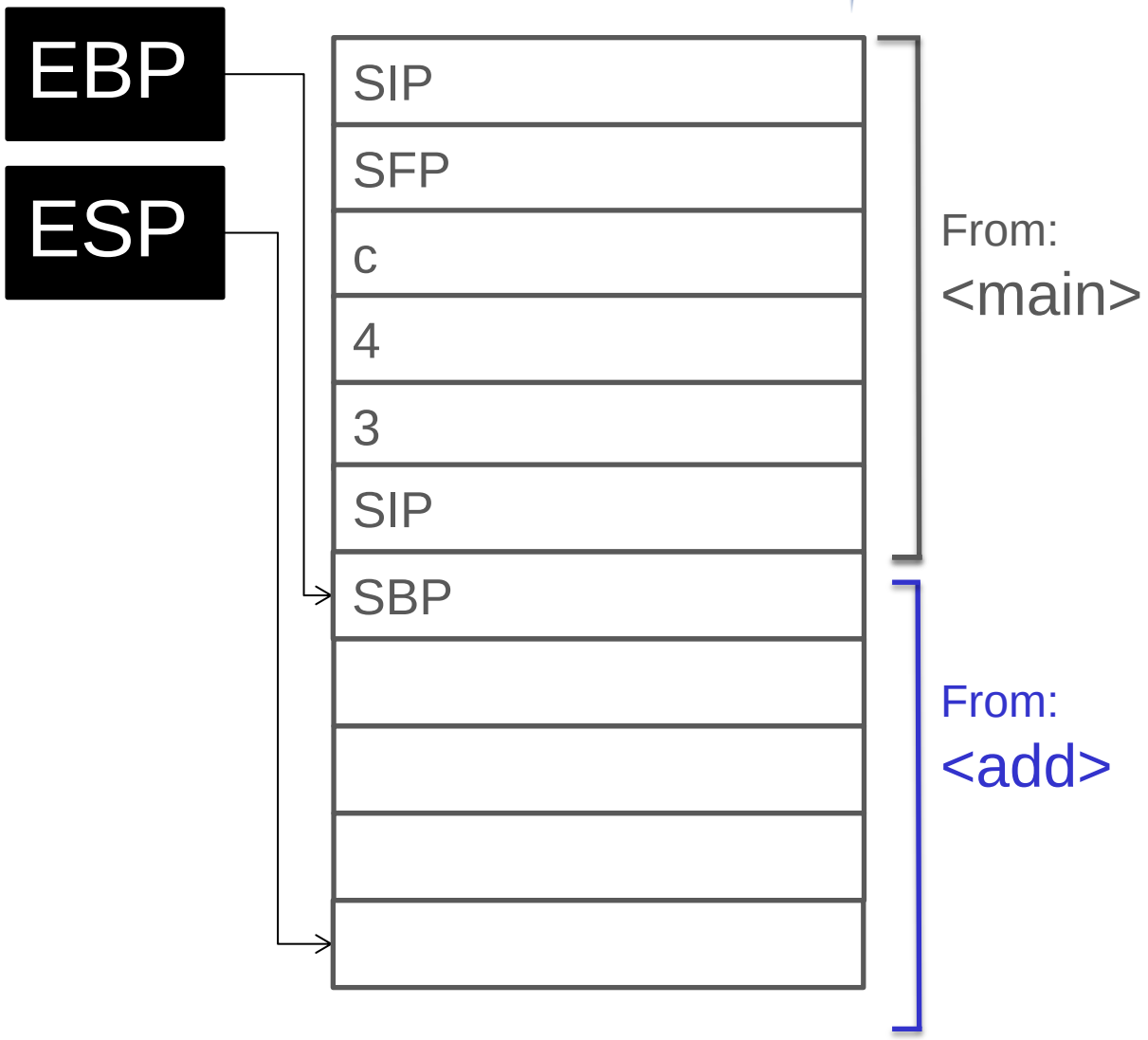
Function Epilog

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

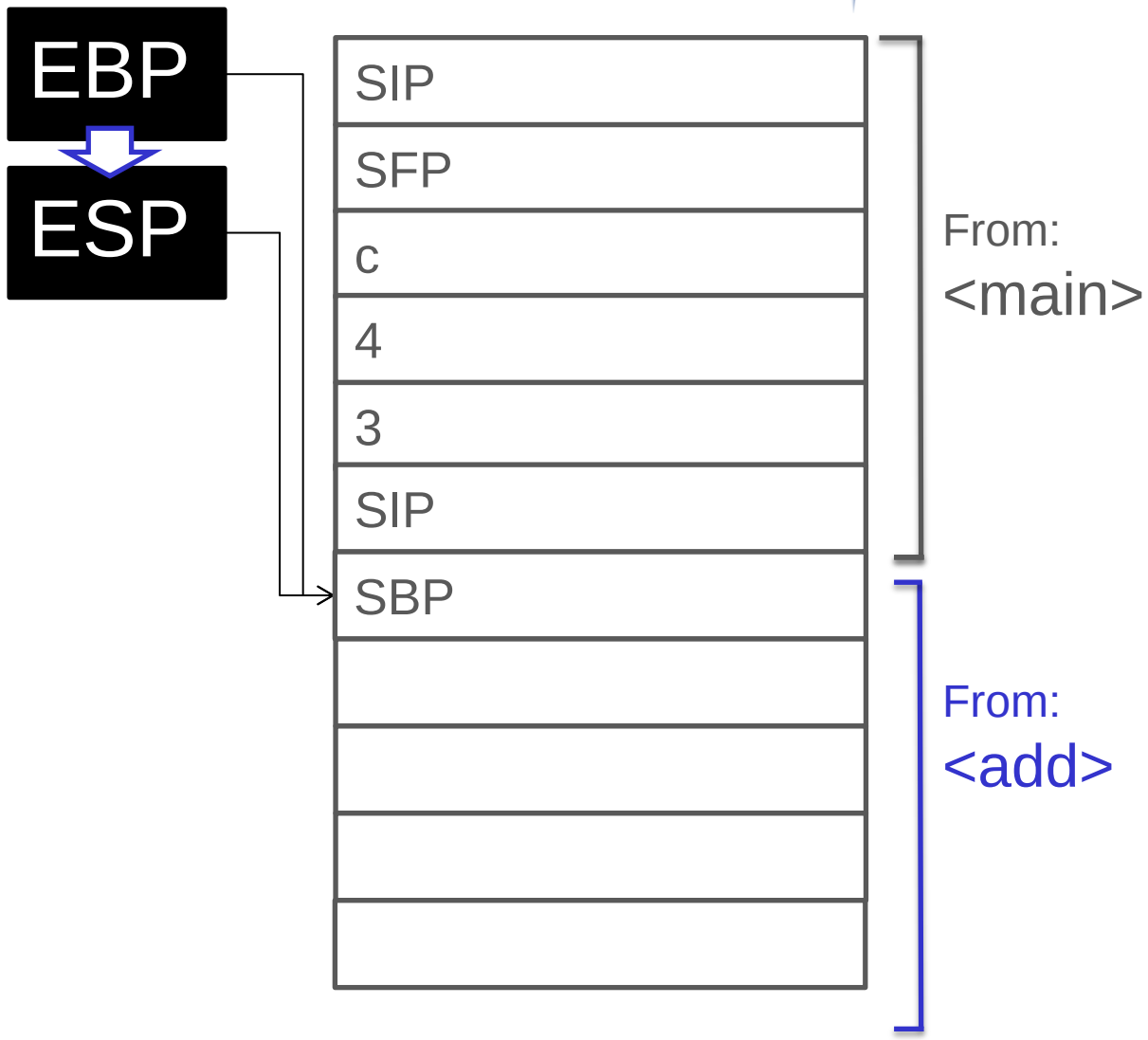
x32 Call Convention - Function Epilog

```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```

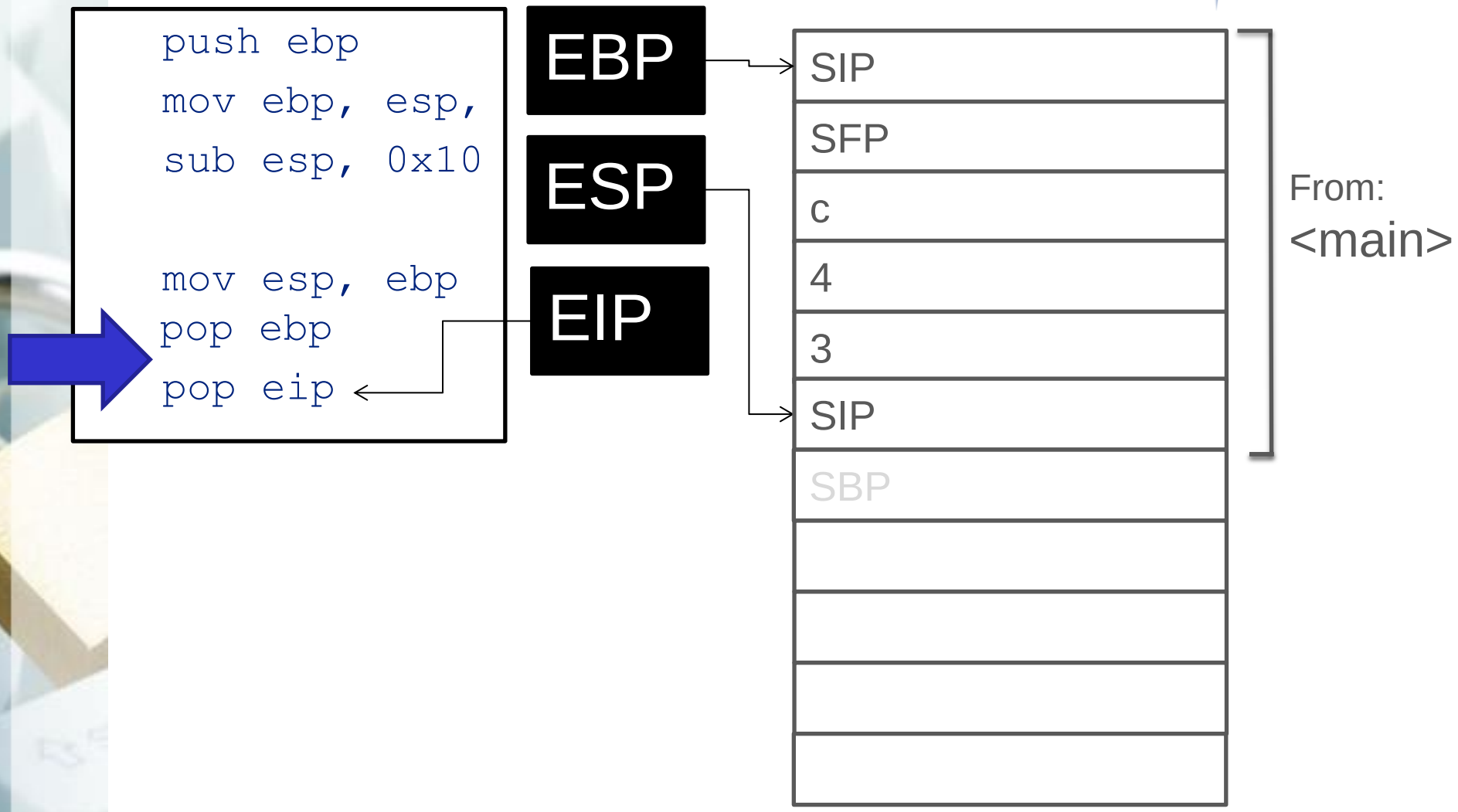


x32 Call Convention - Function Epilog

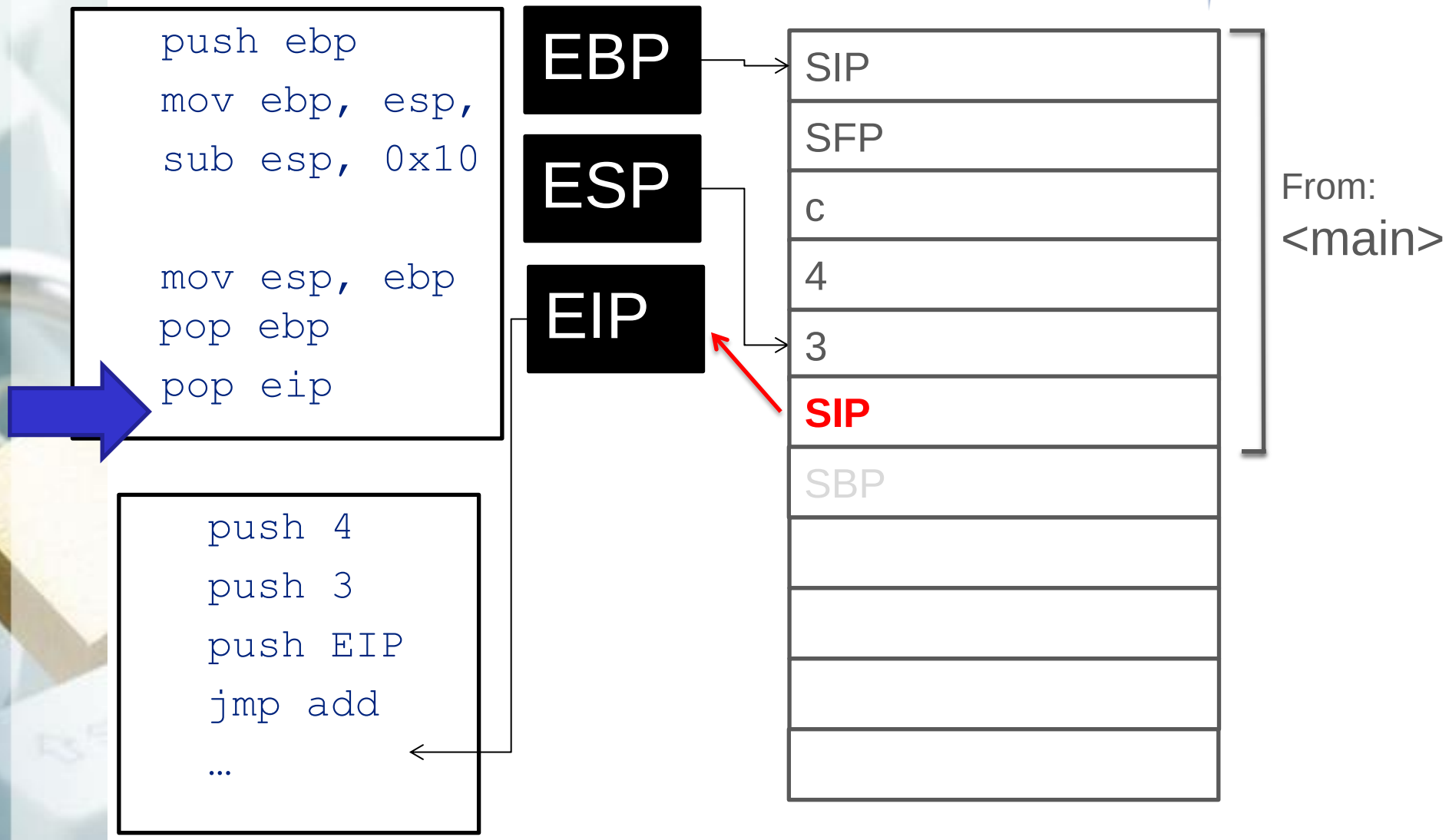
```
push ebp
mov ebp, esp,
sub esp, 0x10

mov esp, ebp
pop ebp
pop eip
```



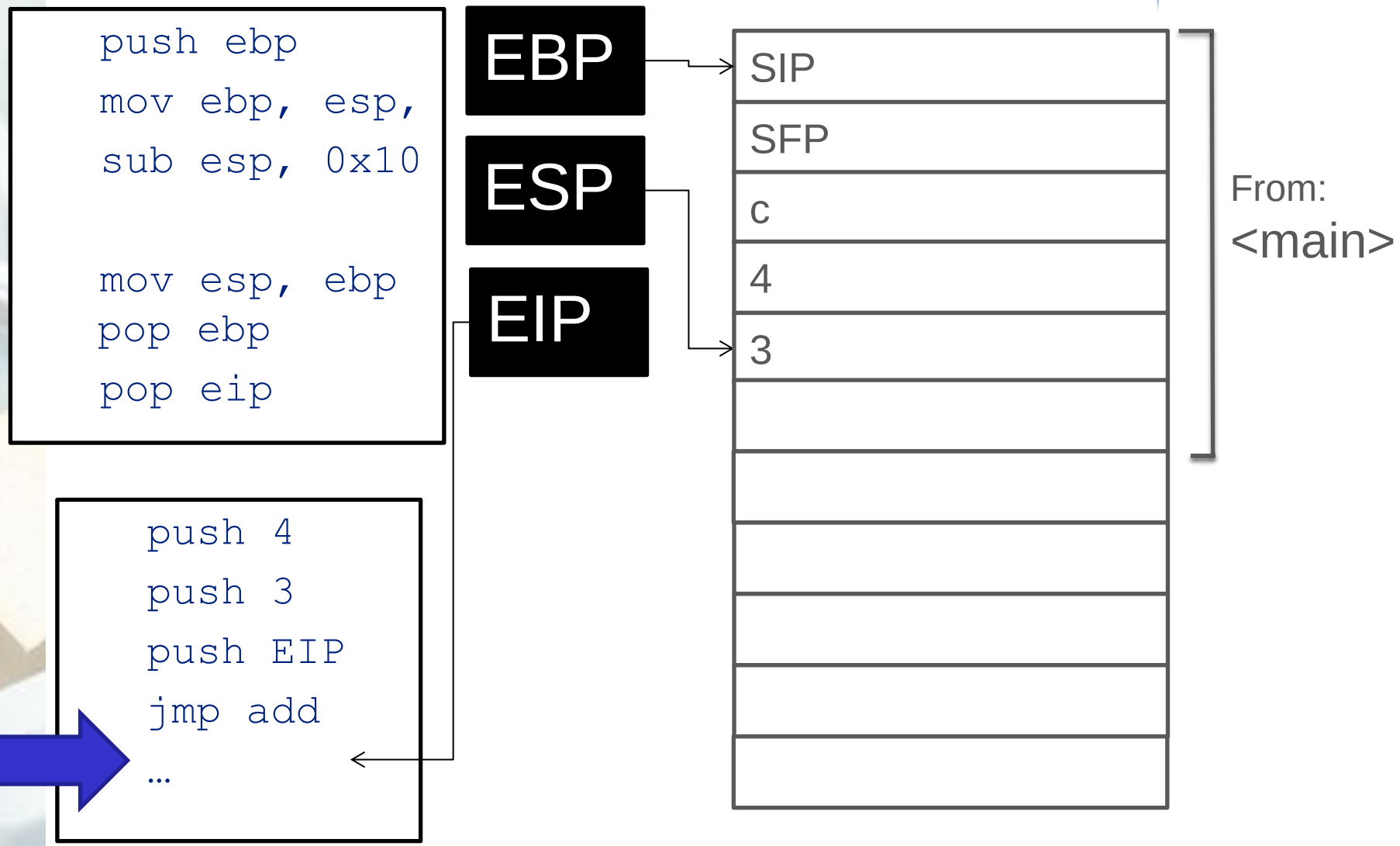
x32 Call Convention - Function Epilog



x32 Call Convention - Function Epilog



x32 Call Convention - Function Epilog



x32 Call Convention - Function Calling

```
call <addr> =
  push EIP
  jmp <addr>

leave =
  mov esp, ebp
  pop ebp

ret =
  pop eip
```

x32 Call Convention - Function Calling

Why “leave”?

- Opposite of “enter”

“enter”:

```
push ebp
mov ebp, esp
sub esp, imm
```

Why no “enter” used?

- enter:
  - 8 cycle latency
  - 10-20 micro ops
- call <addr>; mov ebp, esp; sub esp, imm:
  - 3 cycles latency
  - 4-6 micro ops

Function Call in x64

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

x32 Call Convention - Function Call in x64

Differences between x32 and x64 function calls:

Arguments are in registers (not on stack)

```
RDI, RSI, RDX, R8, R9
```

x32 Call Convention - Function Call in x64

Differences between x32 and x64 function calls

Different ASM commands doing the same thing

```
callq (call)
leaveq (leave)
retq (ret)
```

x32 Call Convention - Function Calling

Recap:

- When a function is called:
  - EIP is pushed on the stack (=SIP)
  - ("call" is doing implicit "push EIP")
- At the end of the function:
  - SIP is recovered into EIP
  - ("ret" is doing implicit "pop EIP")

Function Call Convention Cheat Sheet

x32	Parameter	Syscall nr in
x32 userspace	`stack`	
x32 syscalls	`ebx, ecx, edx, esi, edi, ebp`	eax

x64	Parameter	Syscall nr in
x64 userspace	`rdi, rsi, rdx, rcx, r8, r9`	
x64 syscall	`rdi, rsi, rdx, r10, r8, r9`	rax

http://stackoverflow.com/questions/2535989/what-are-the-calling-conventions-for-unix-linux-system-calls-on-x86-64