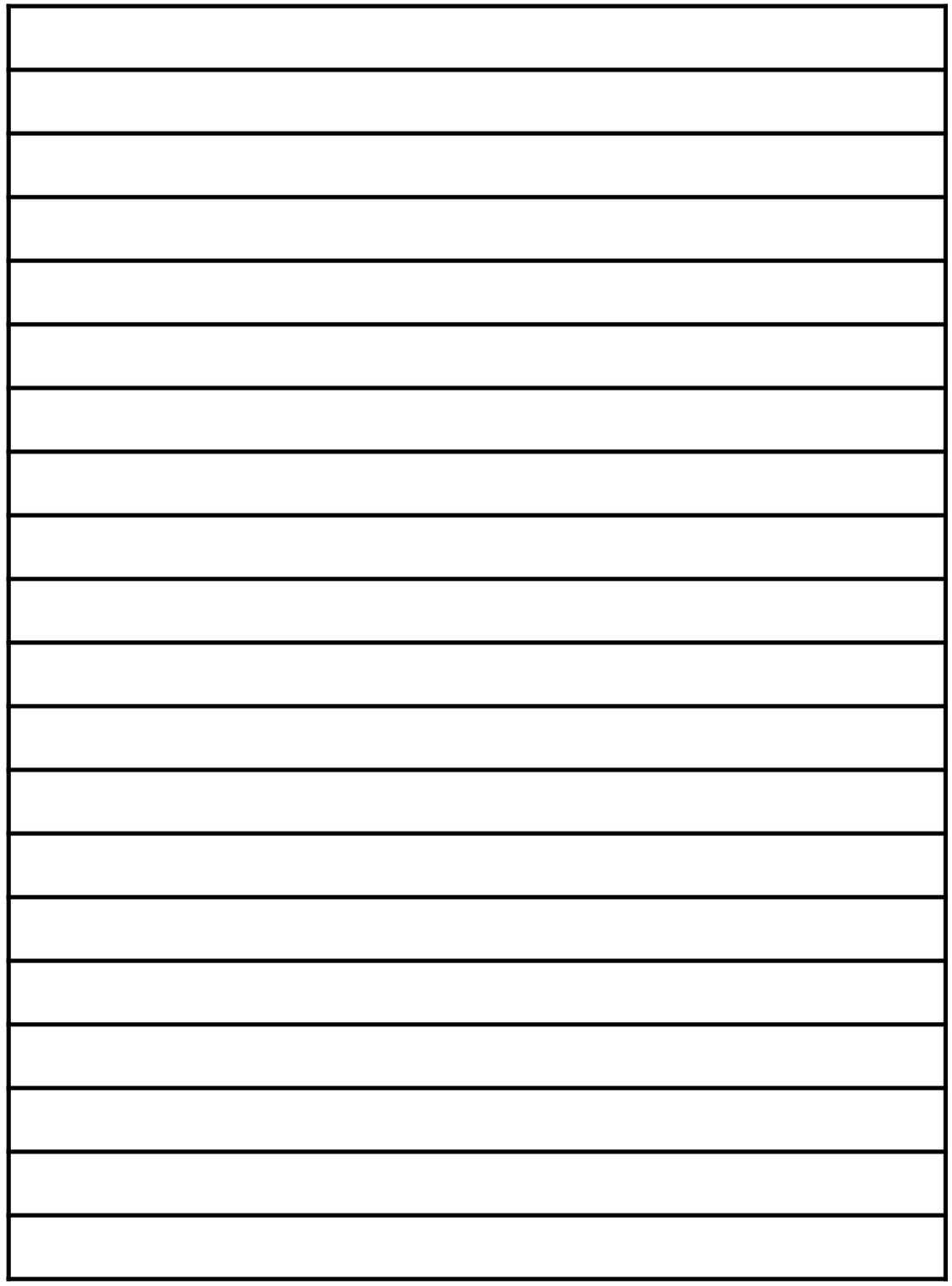
Calling Conventions

September 27, 2021

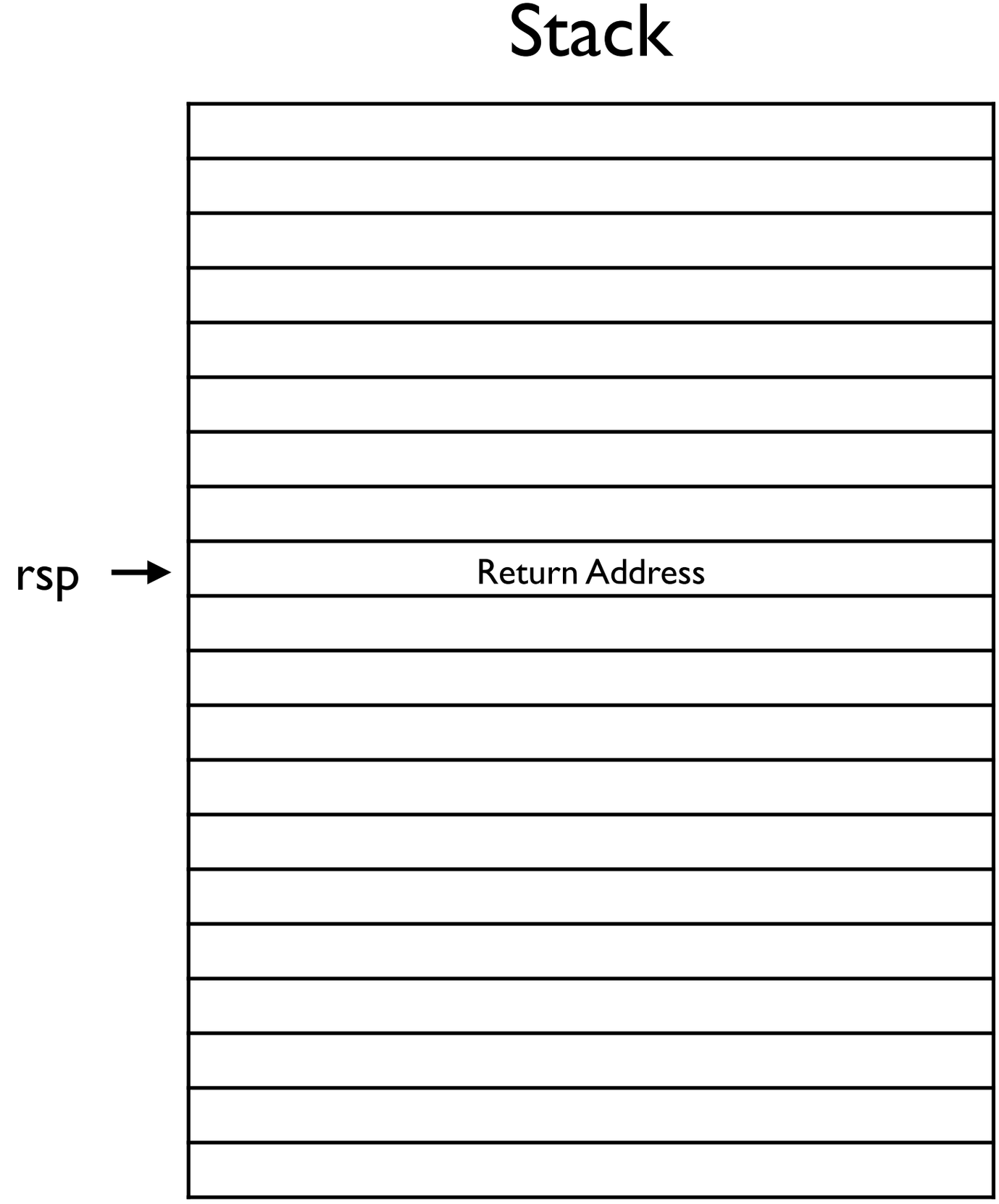
Calling Convention

- Where to put the return address?
- Where to put the returned value (if any)?
- Where to put the arguments?
- How to ensure local variables are preserved?
- How to ensure the stack is well-formed?

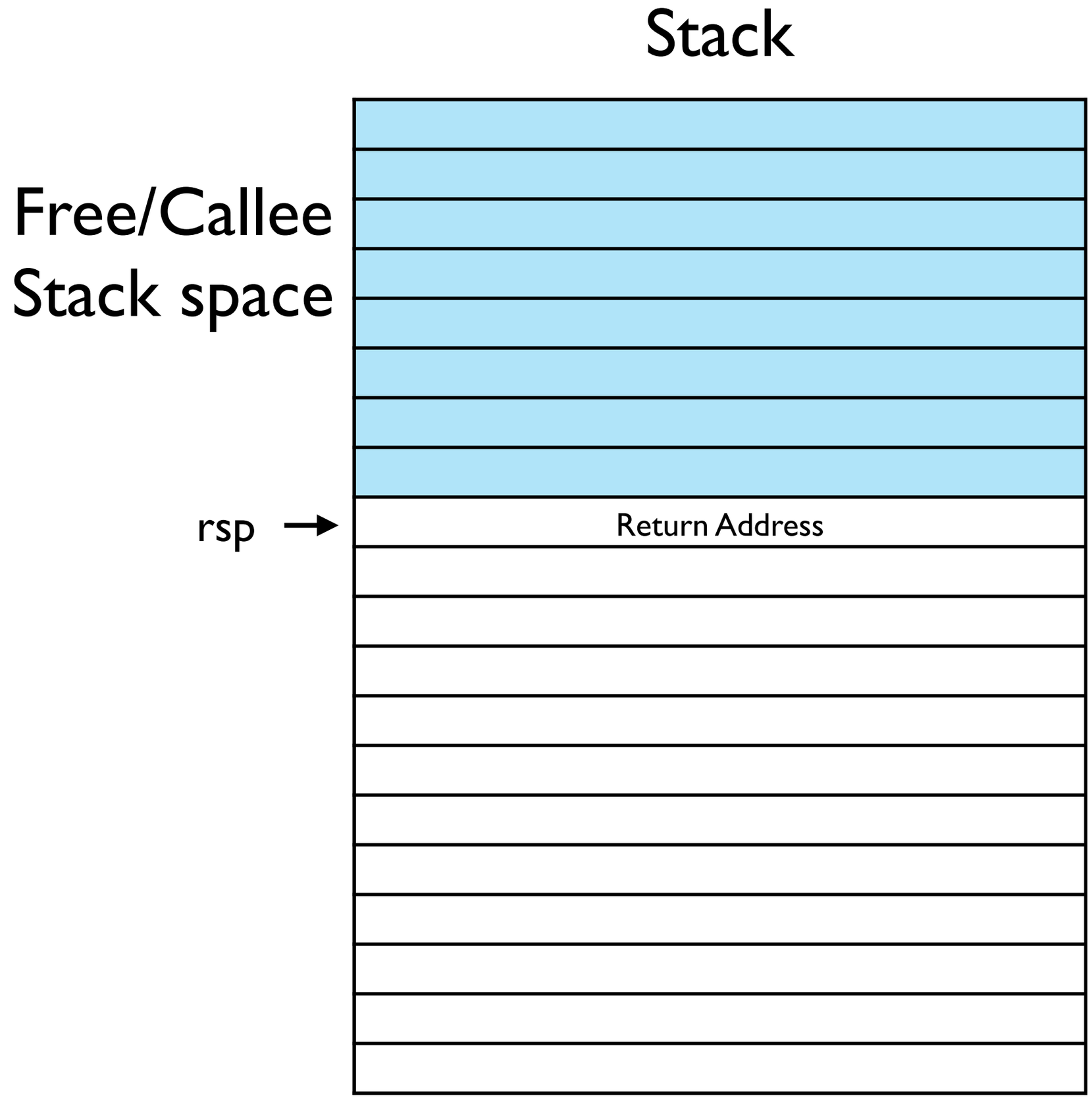
Stack



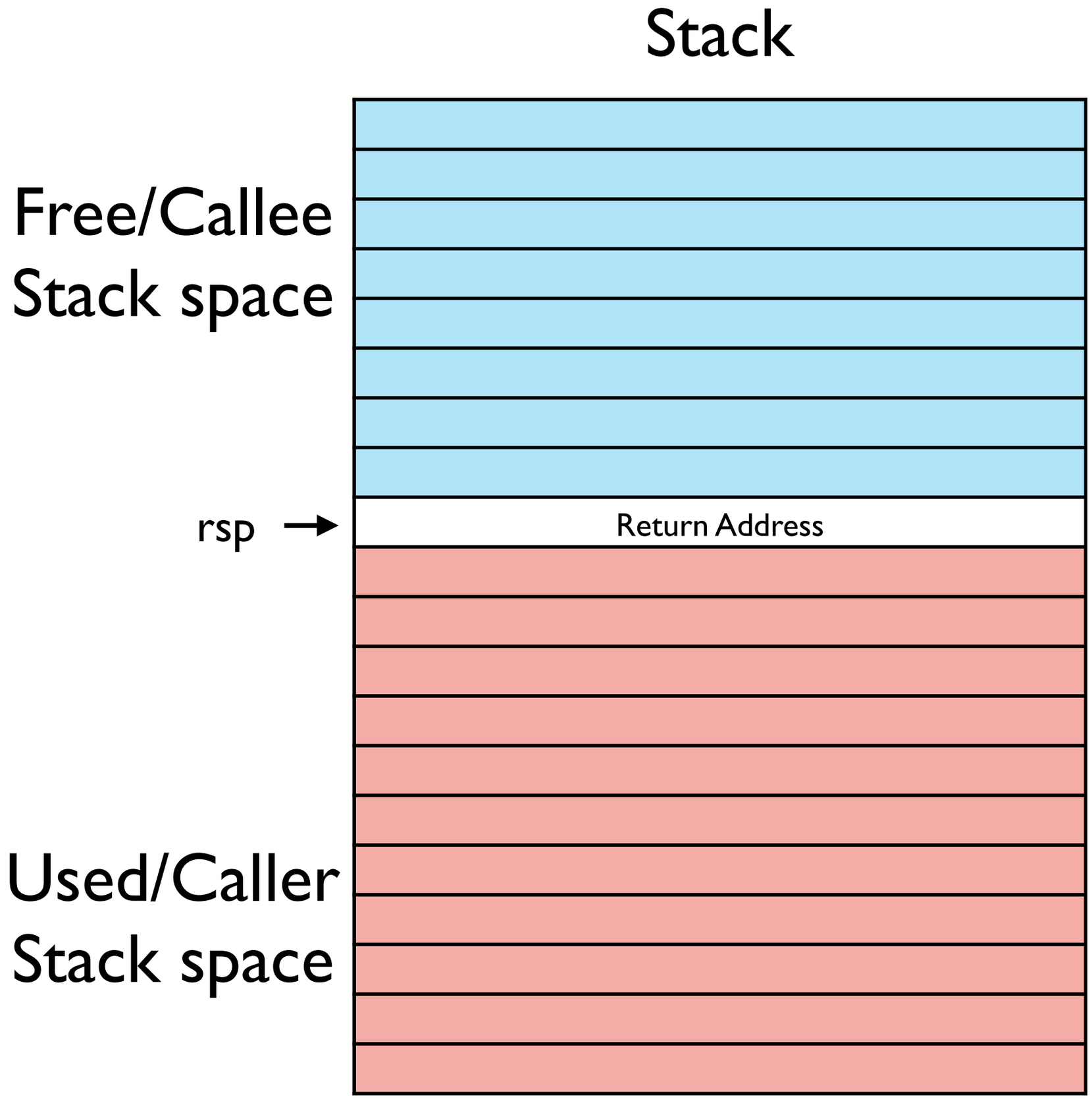
Upon entry to a function

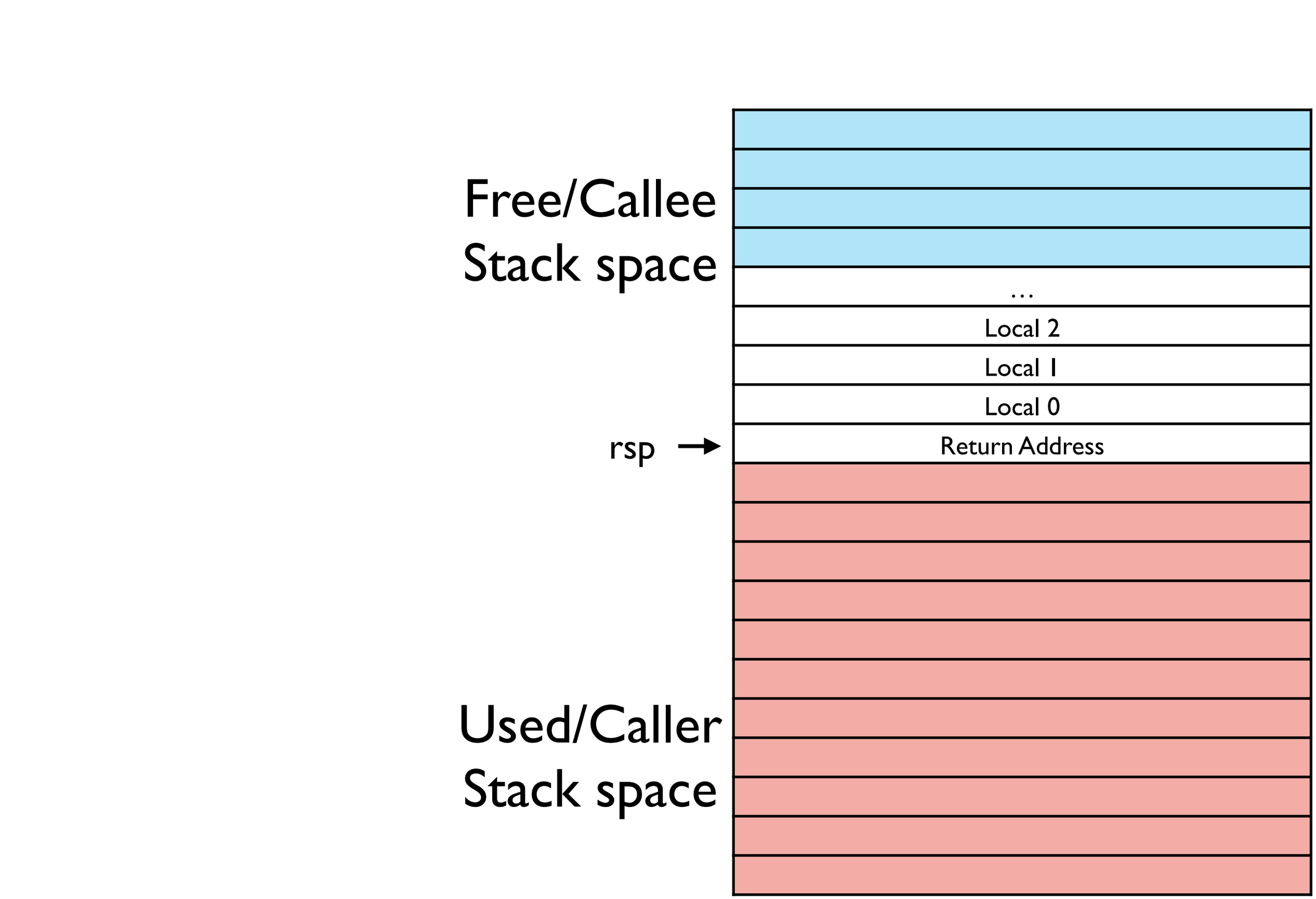


Upon entry
to a function

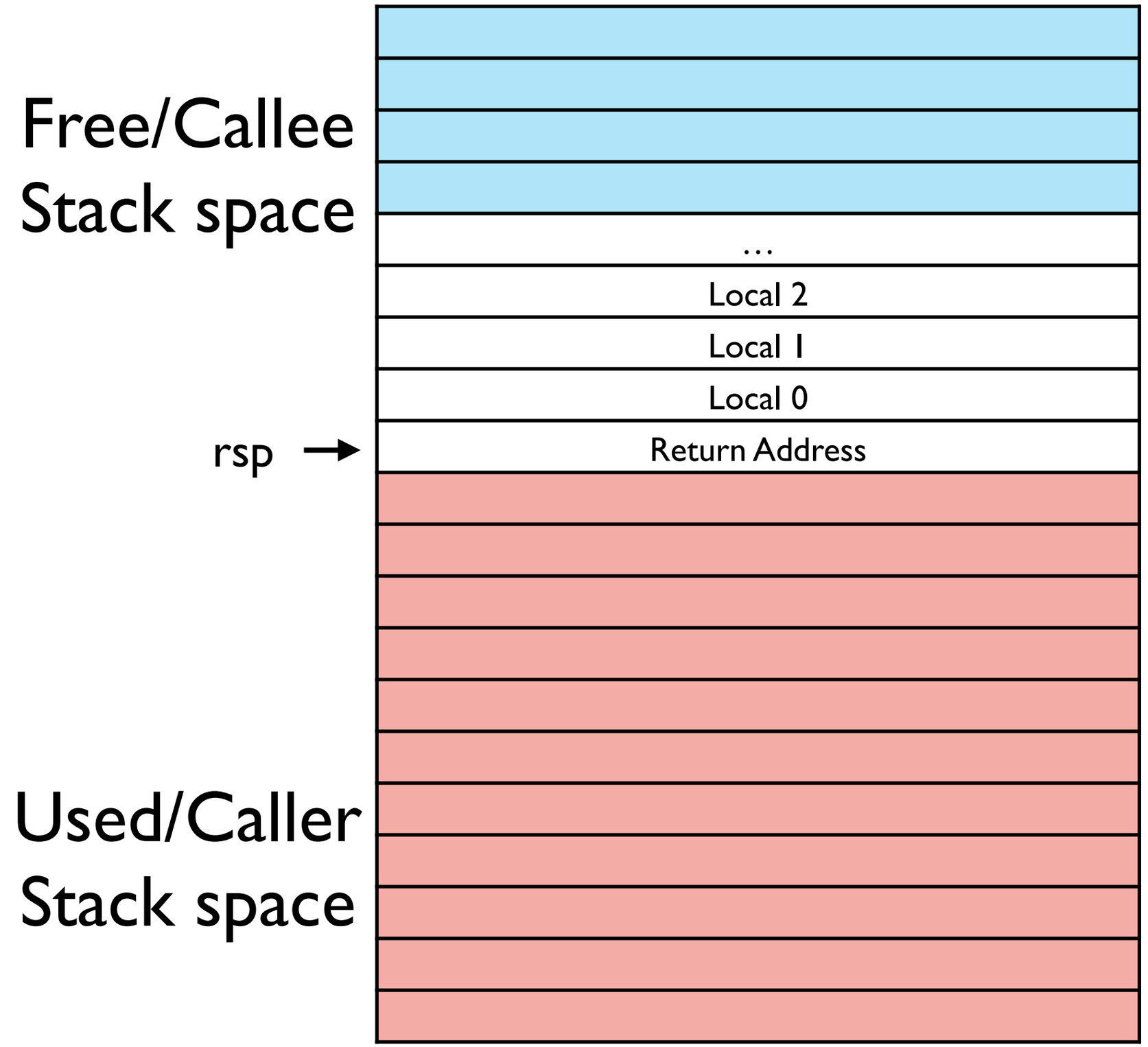


Upon entry
to a function




Free/Callee
Stack space
…
Local 2
Local 1
Local 0
rsp
Return Address
Used/Caller
Stack space

What about function arguments?

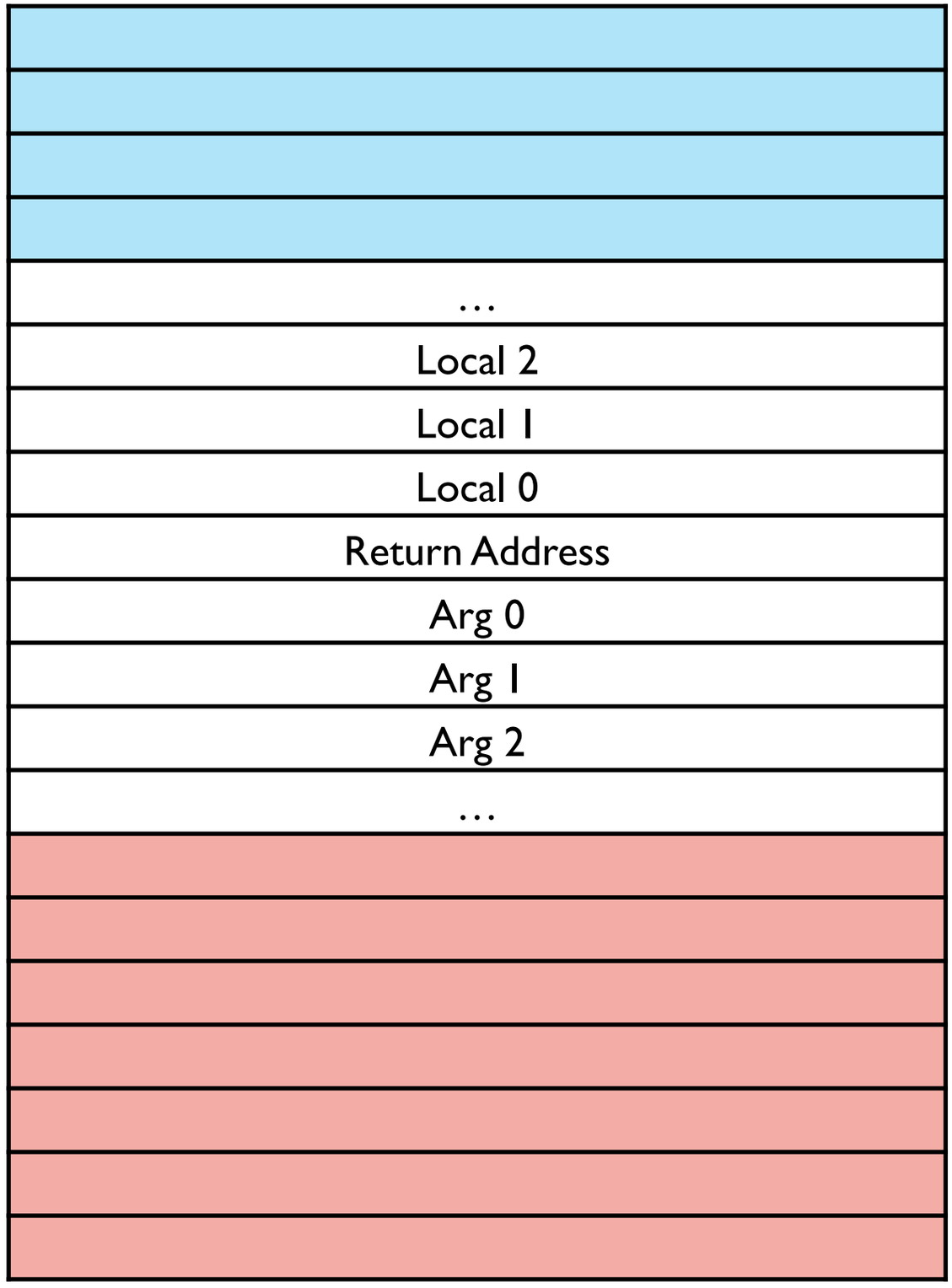


What about function arguments?

below rsp

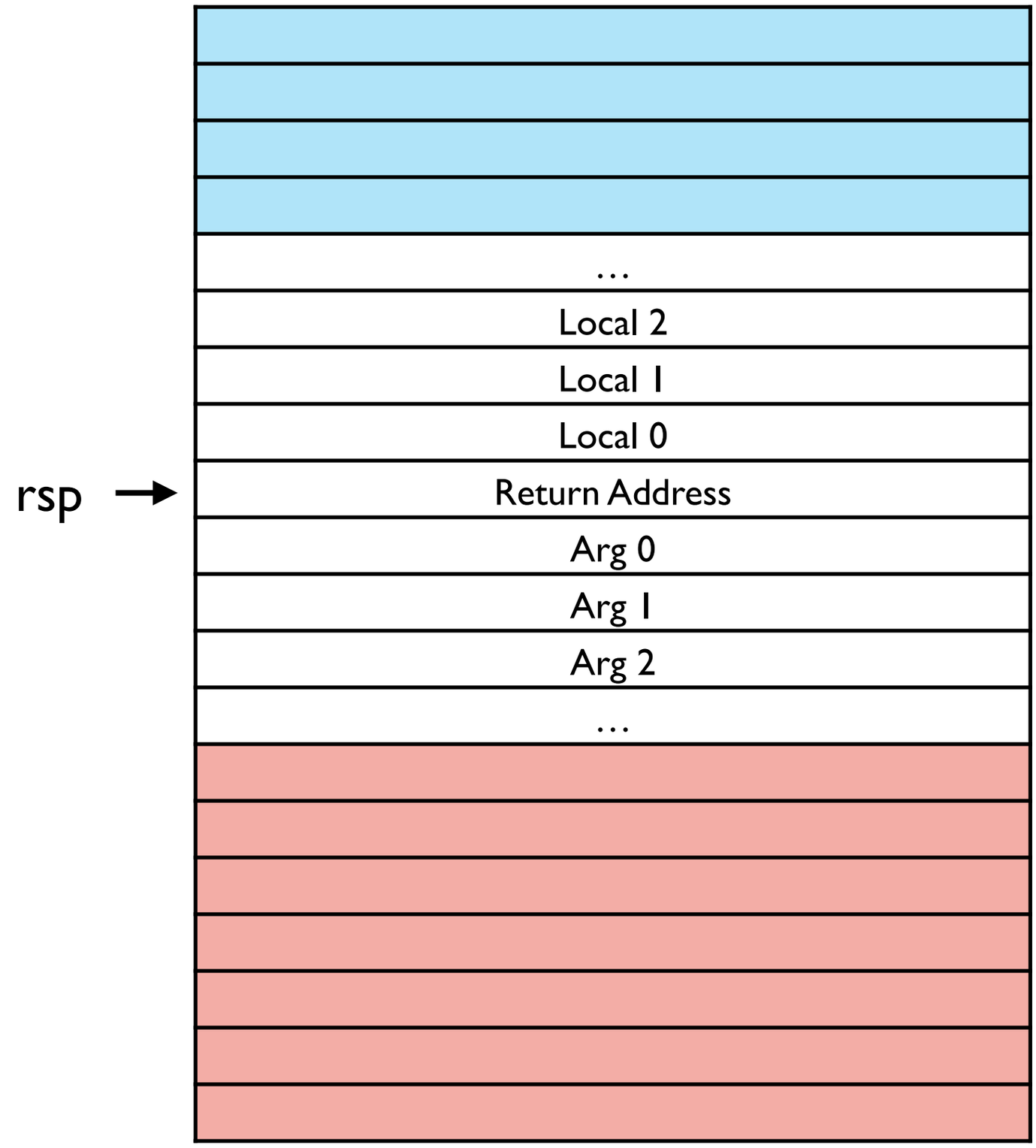
Free/Callee Stack space

rsp →



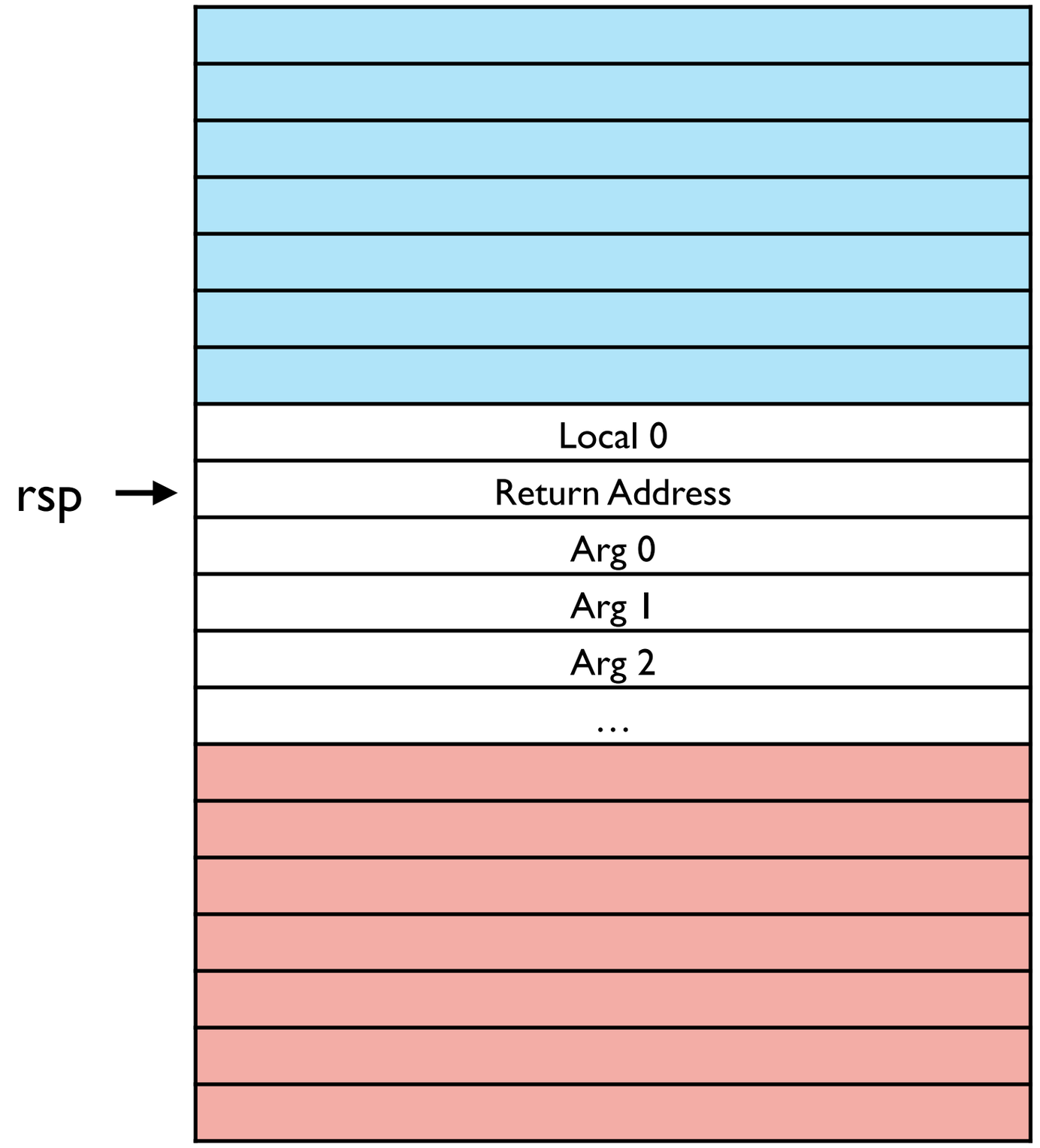
Used/Caller Stack space

- rsp points to the return address
- return value goes in rax
- above rsp is free
- below rsp is used by the caller
- arguments go below rsp



32-bit
C Calling Convention

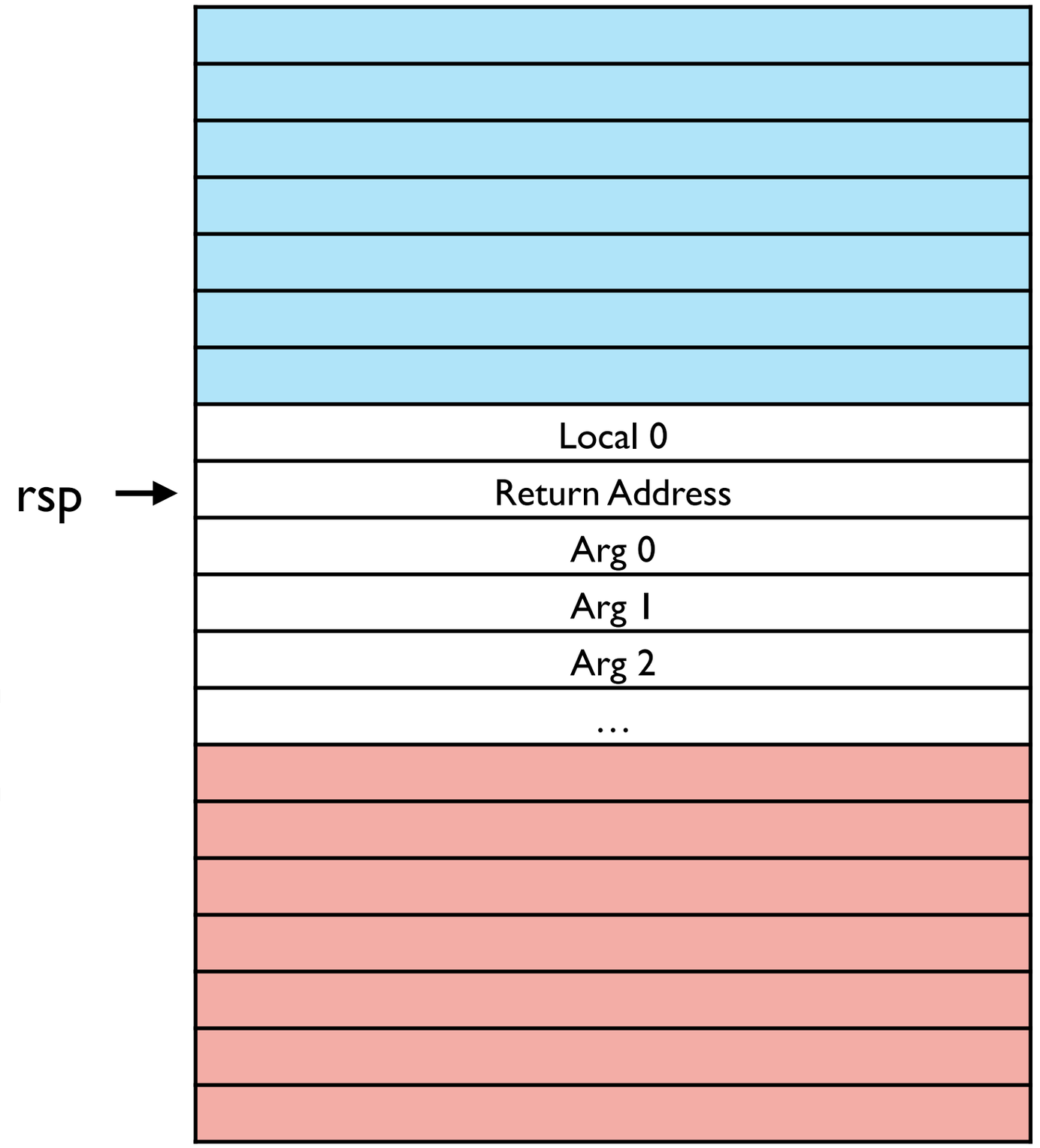
- example
  - 1 local variable
  - call function of 2 arguments



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

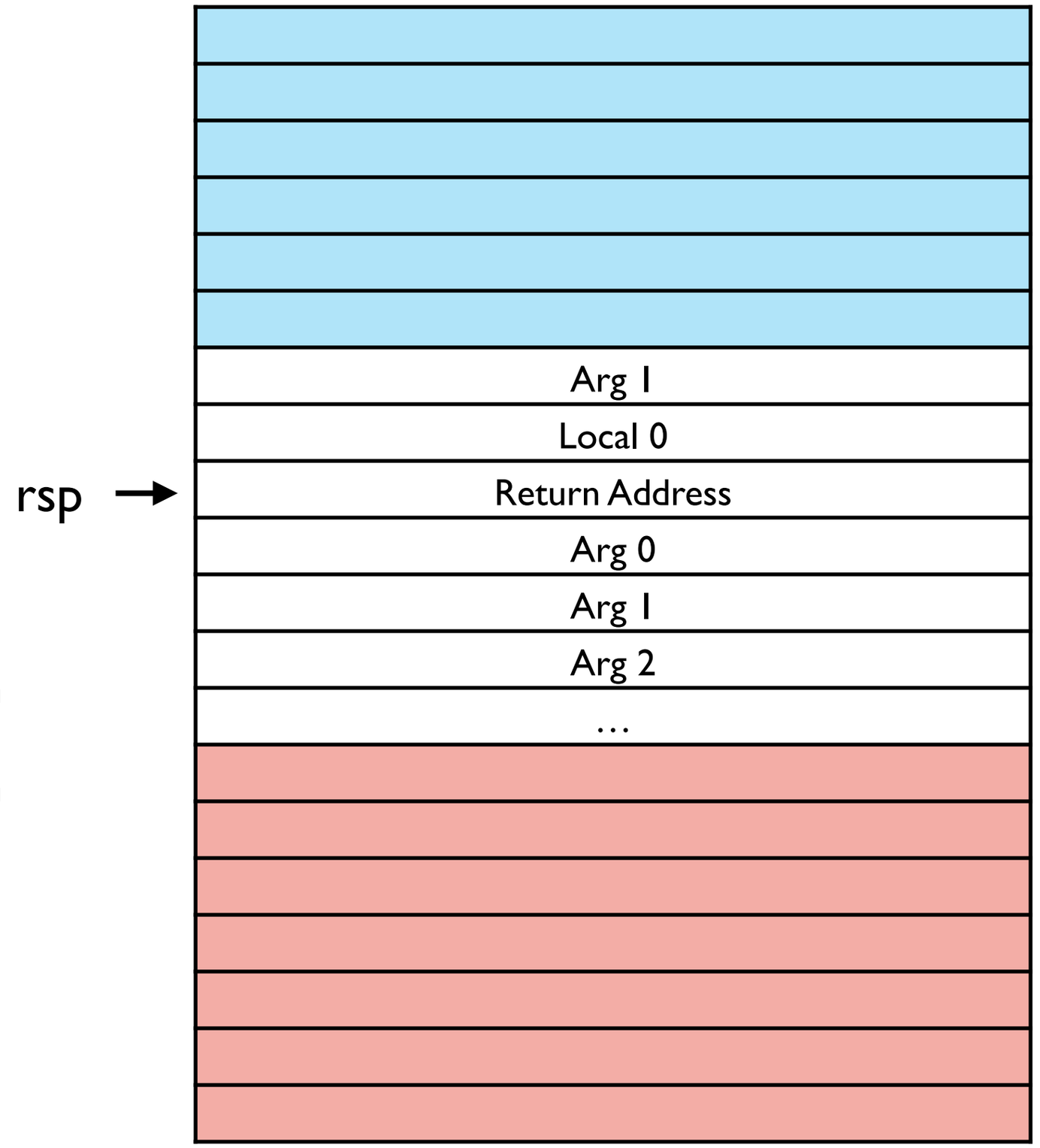
```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

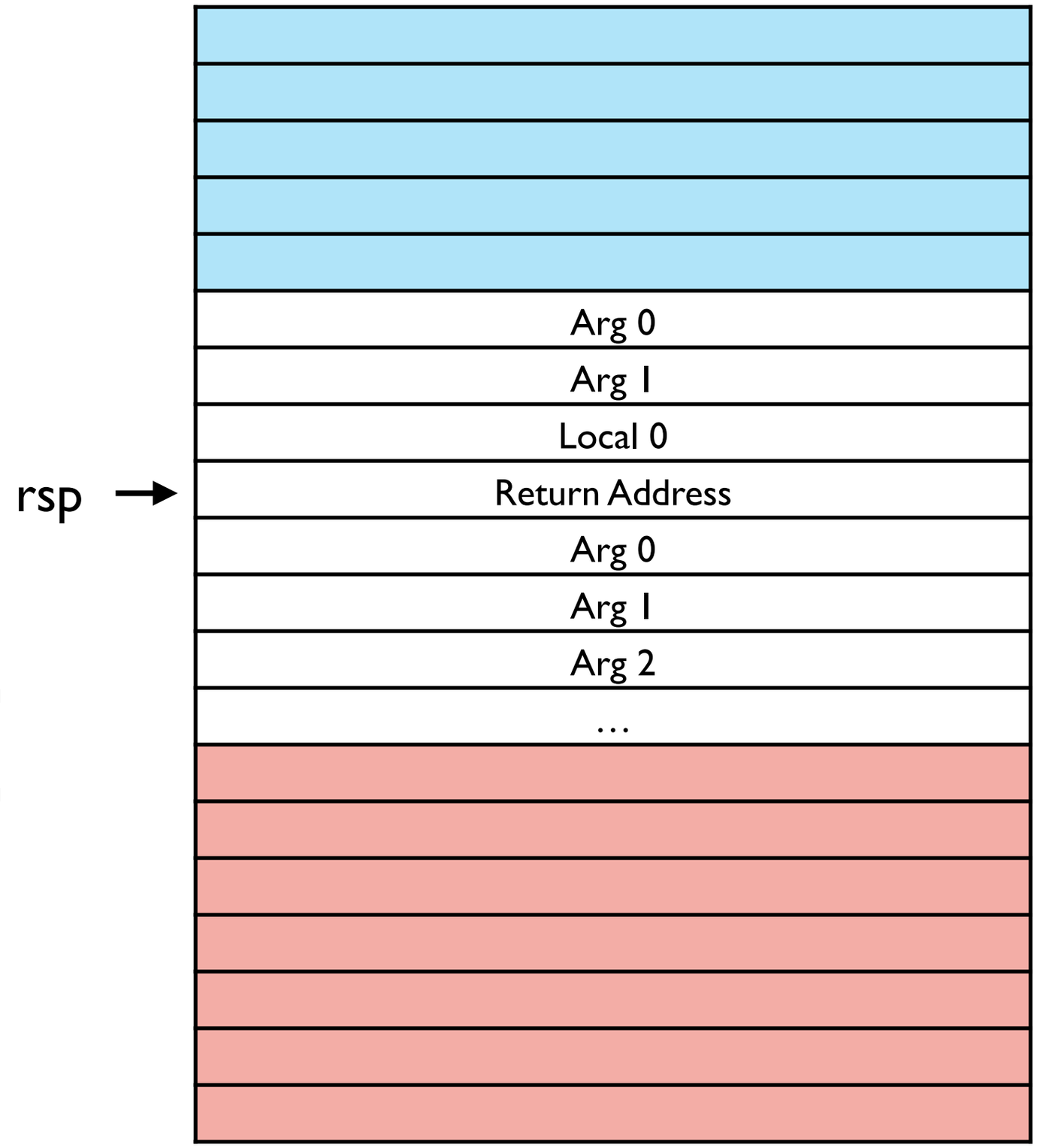
```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

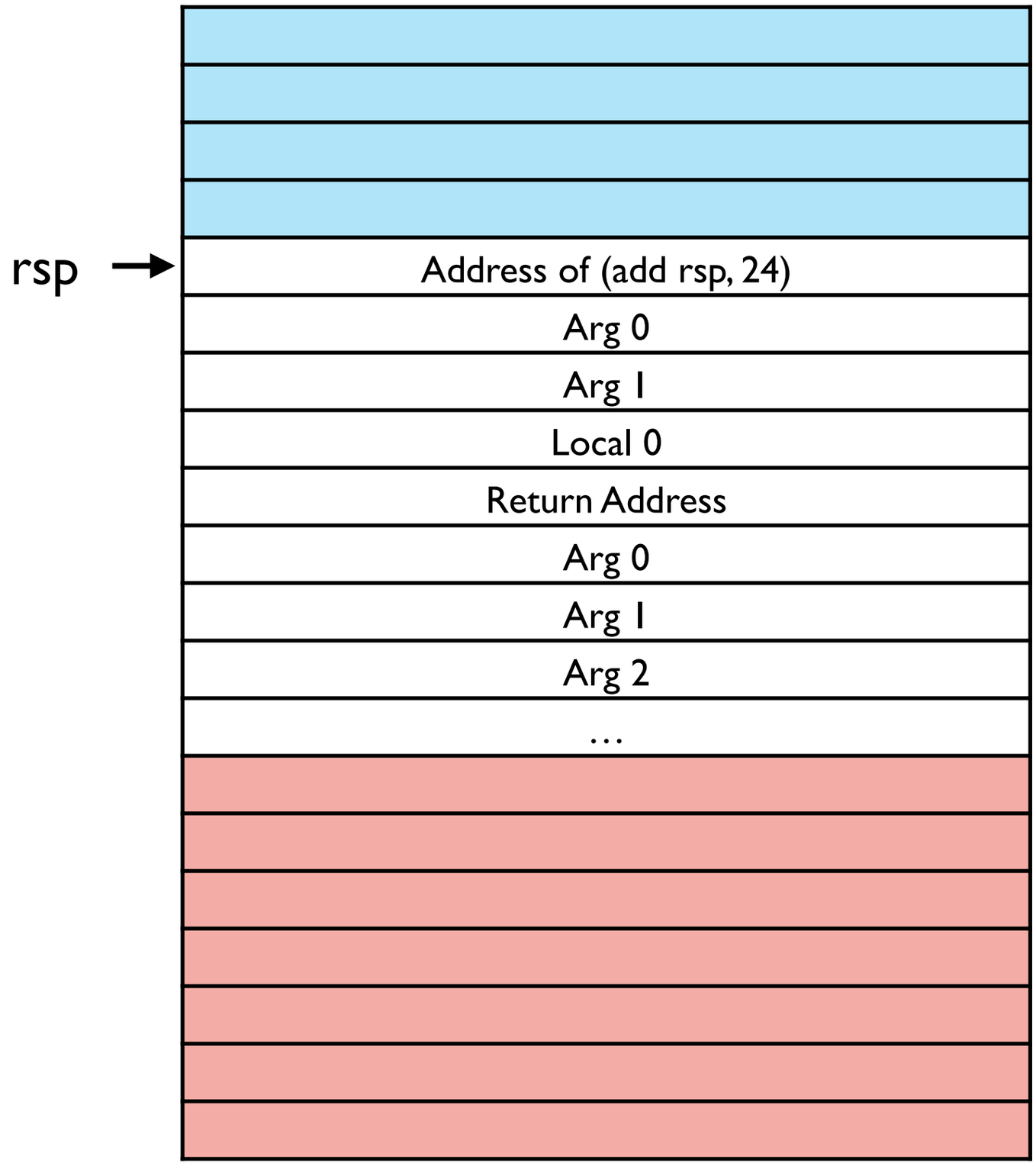
- example
  - 1 local variable
  - call function of 2 arguments

```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```

32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

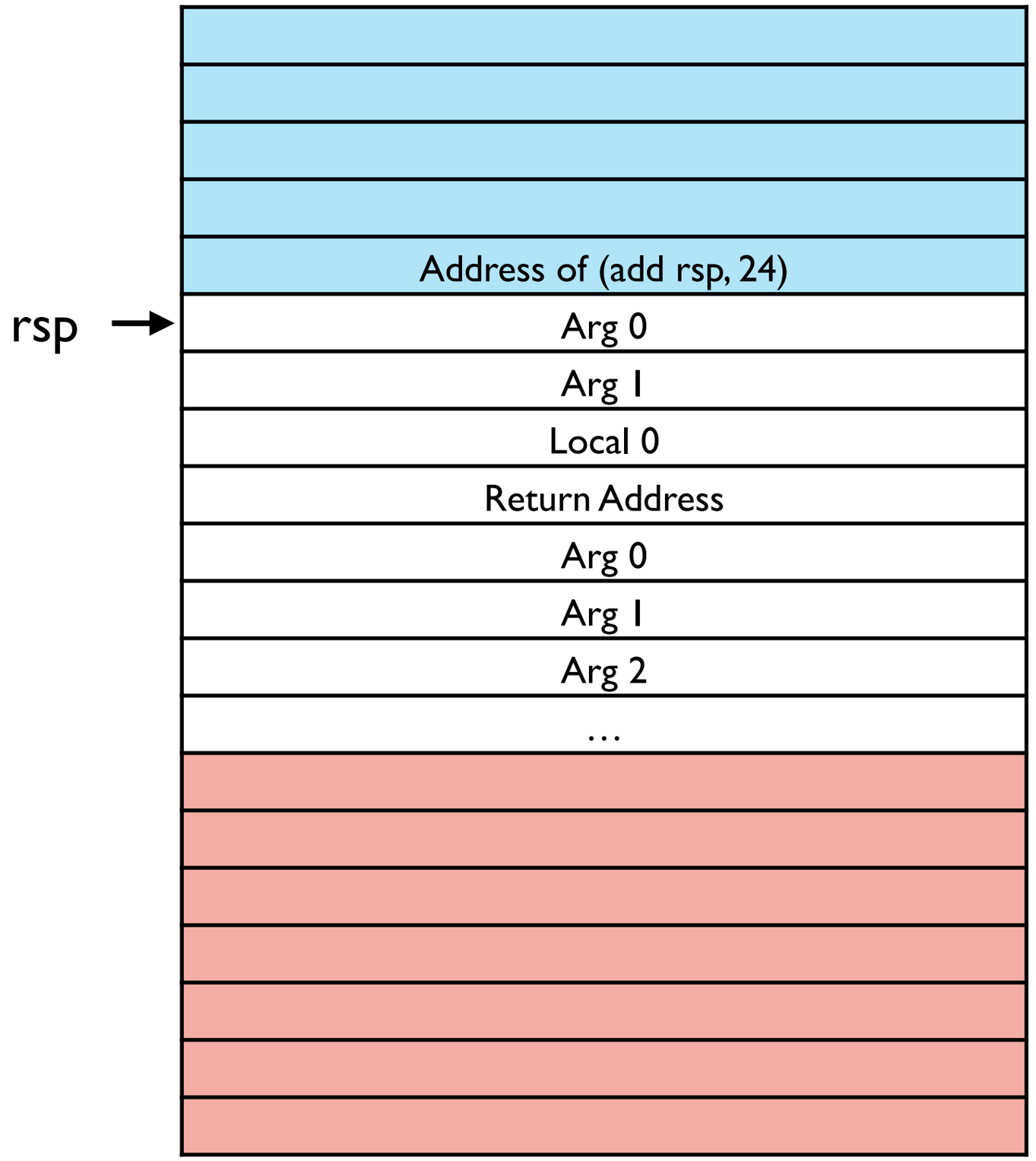
```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

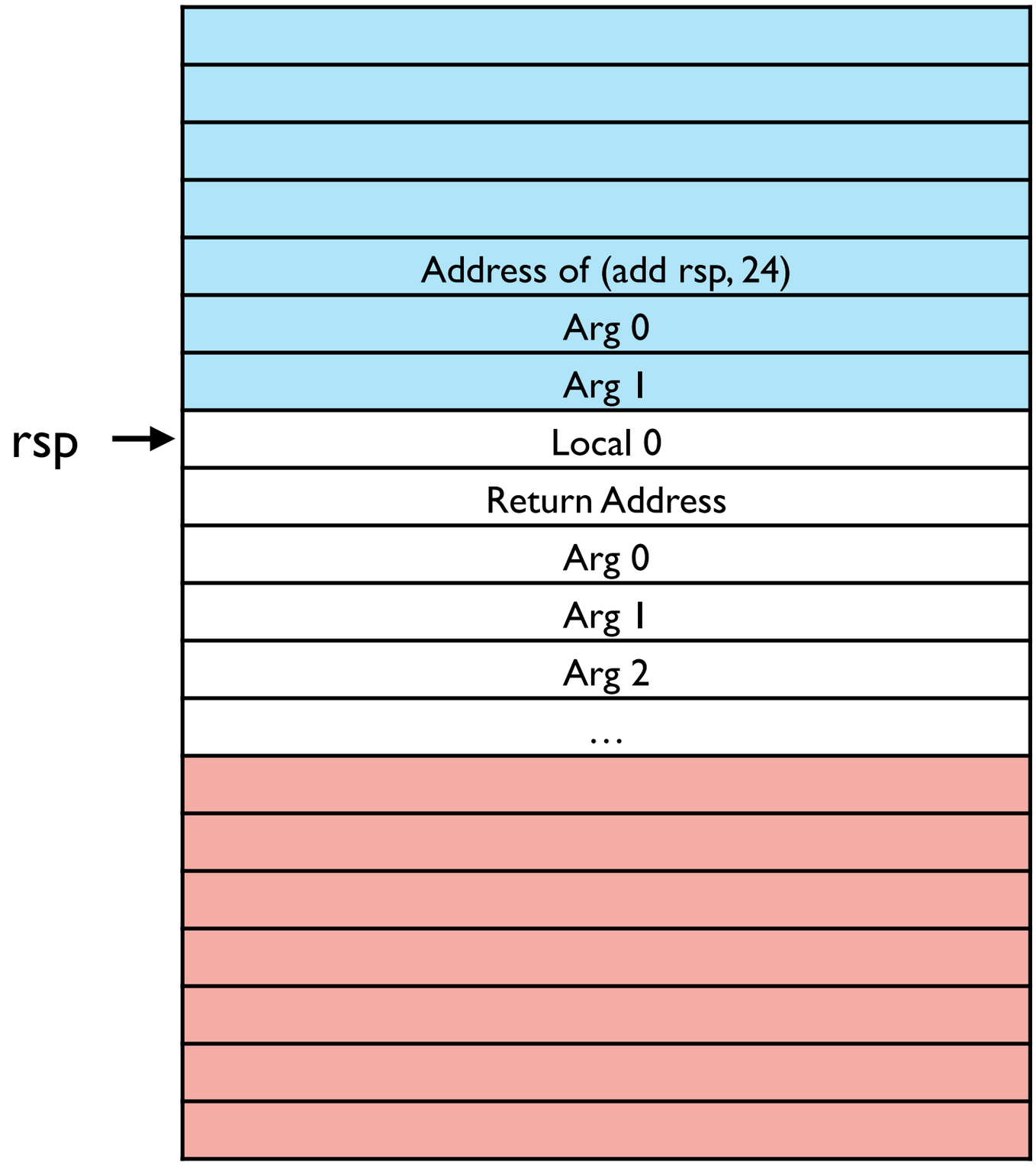
```
mov [rsp - 16], ..
mov [rsp - 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

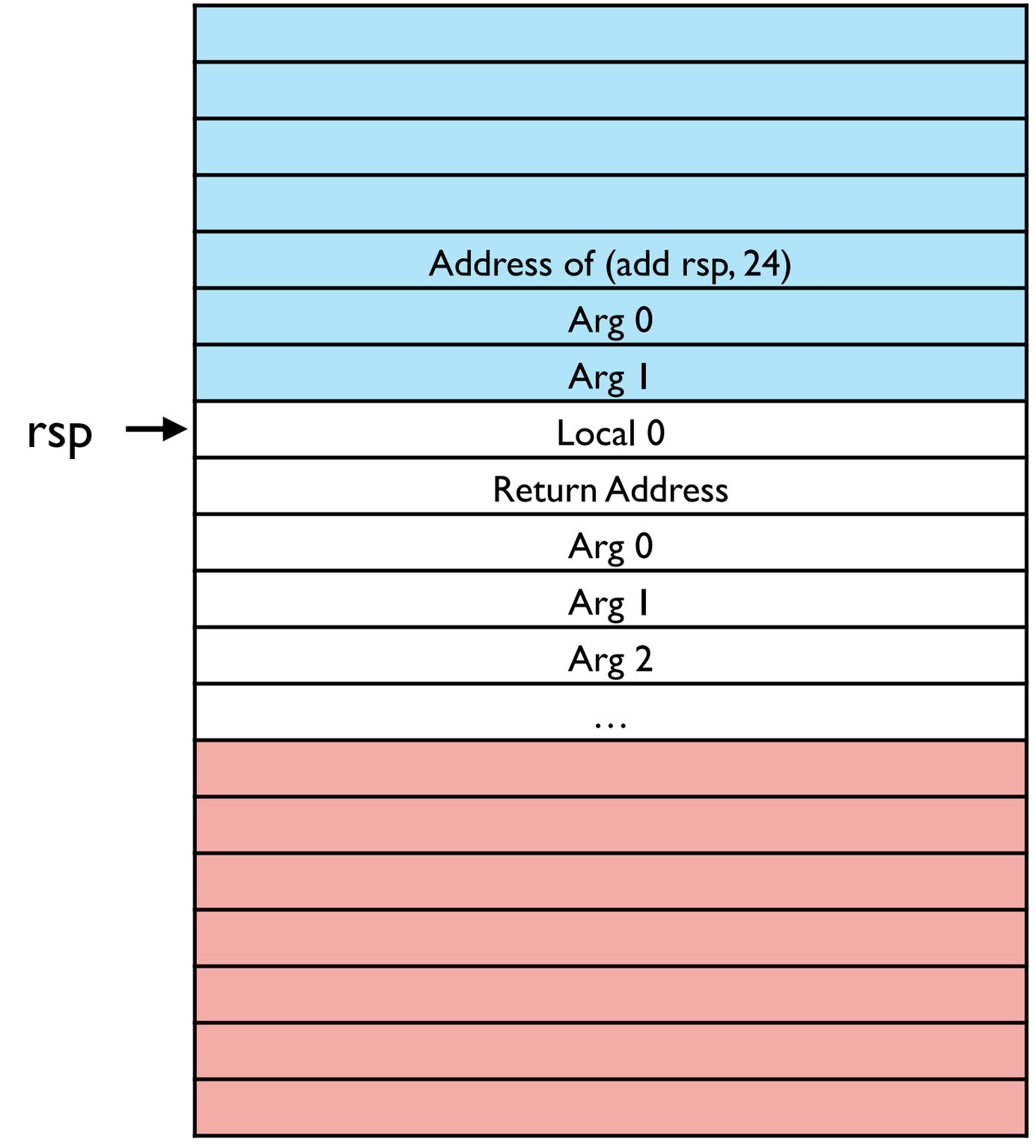
- example
  - 1 local variable
  - call function of 2 arguments

```
mov [rsp – 16], ..
mov [rsp – 24], ..
sub rsp, 24
call fun
add rsp, 24
```



32-bit
C Calling Convention

- Benefit of this approach:
  - just use one register for stack management
- Downside:
  - Can't tell where one stack "frame" begins and the next ends
  - stack trace, debugger, garbage collector need that info

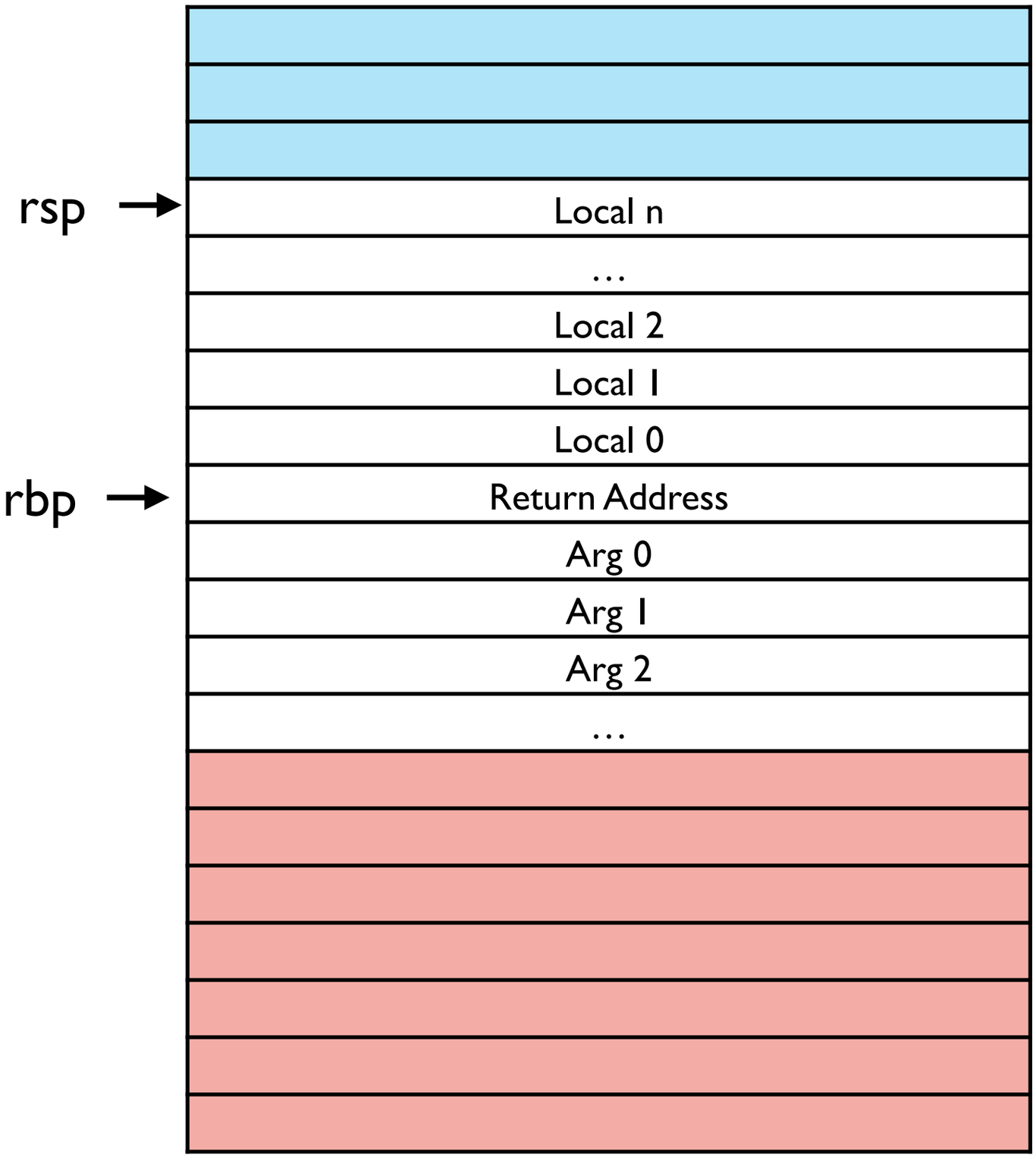


32-bit
C Calling Convention

- rbp: "base"/"frame" pointer: bottom of our portion of the stack

- rsp: stack pointer: top of our portion of the stack

- `push` instruction for arguments



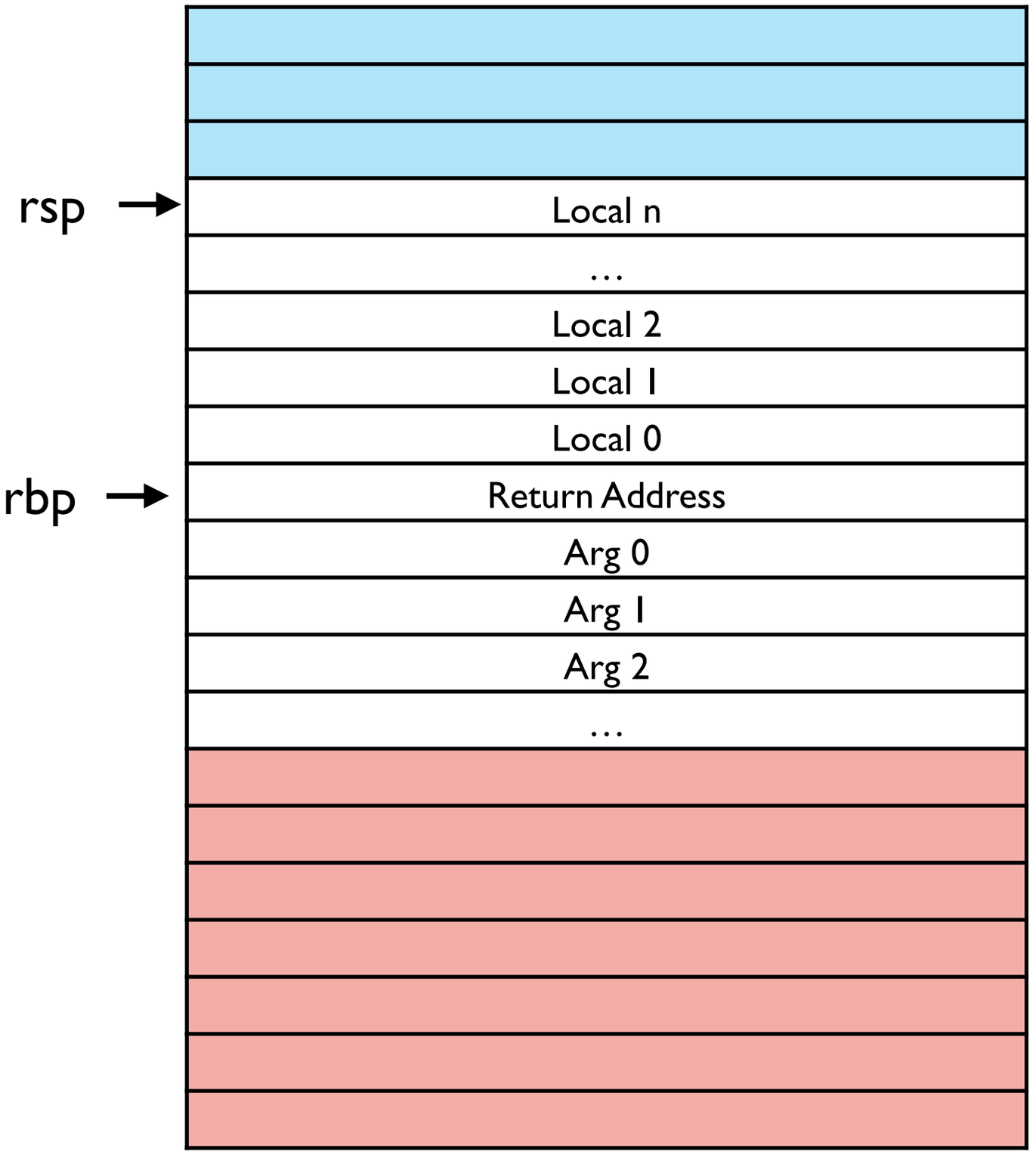
32-bit
C Calling Convention

- rbp: "base"/"frame" pointer: bottom of our portion of the stack

- rsp: stack pointer: top of our portion of the stack

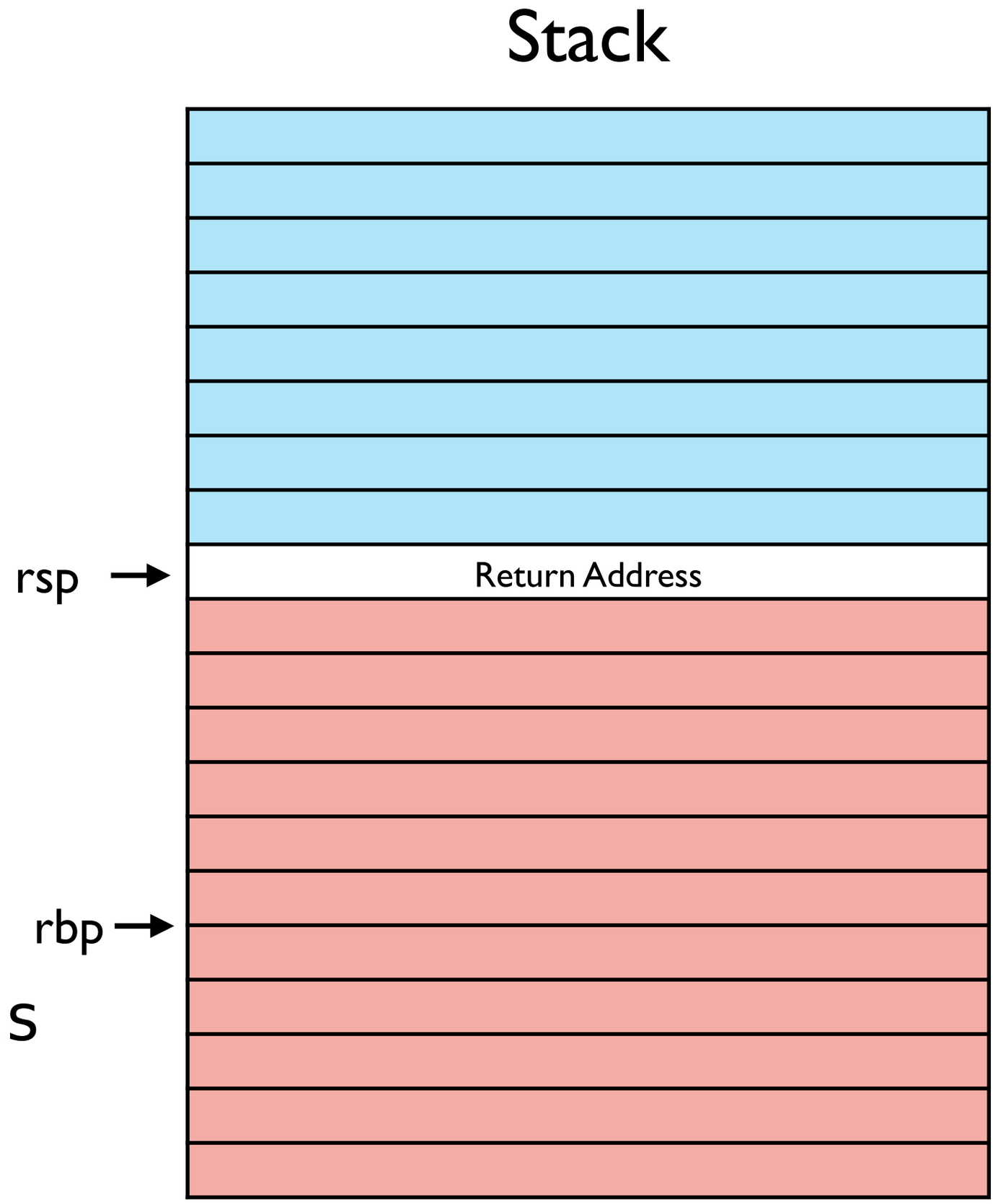
- `push` instruction for arguments

```
start_here:
mov rbp, rsp
mov [rbp – 8], 3
```



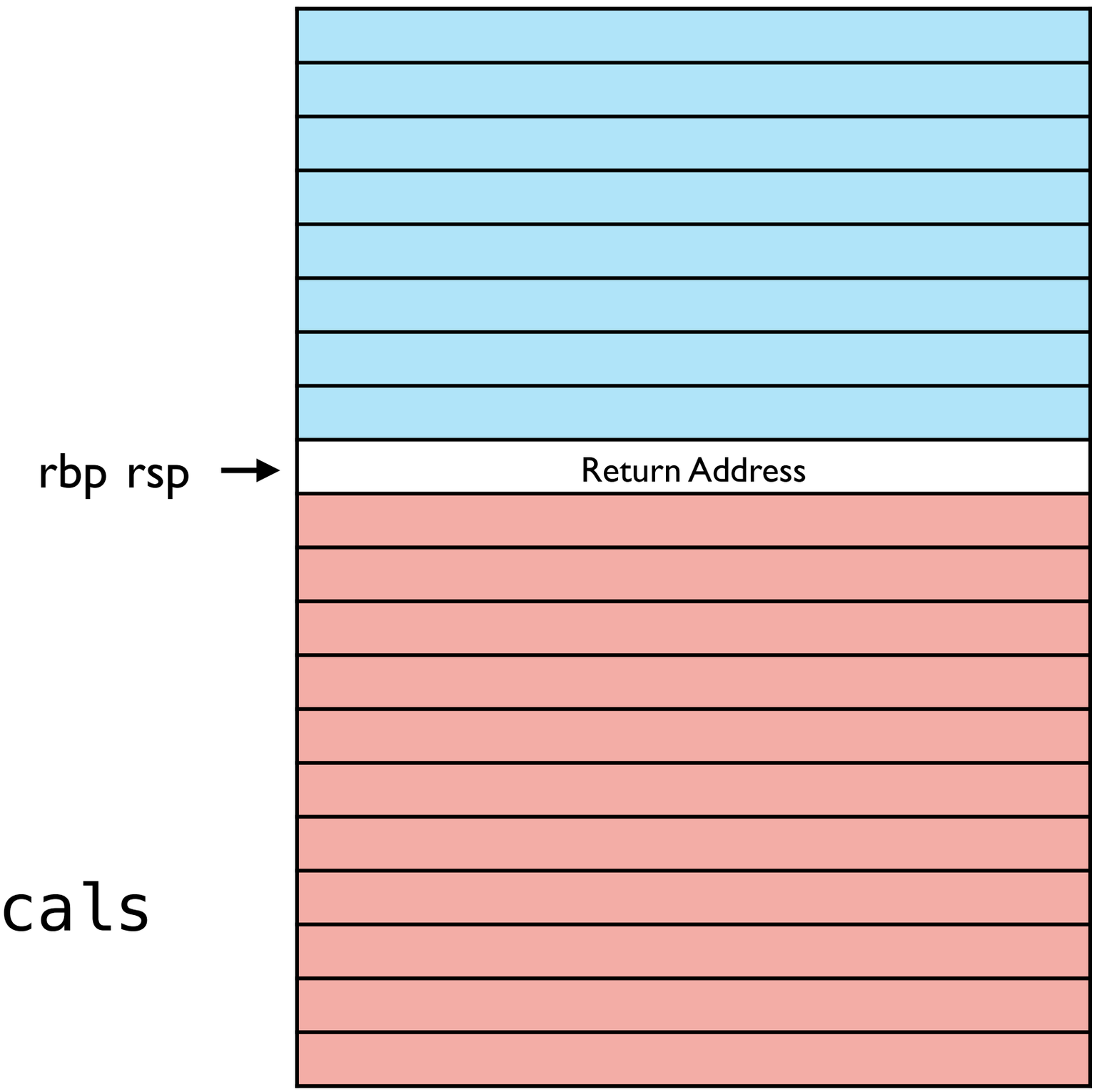
Upon entry to a function

```
start_here:
mov rbp, rsp
sub rsp, 8 * locals
```



Upon entry to a function

```
start_here:
mov rbp, rsp
sub rsp, 8 * locals
```

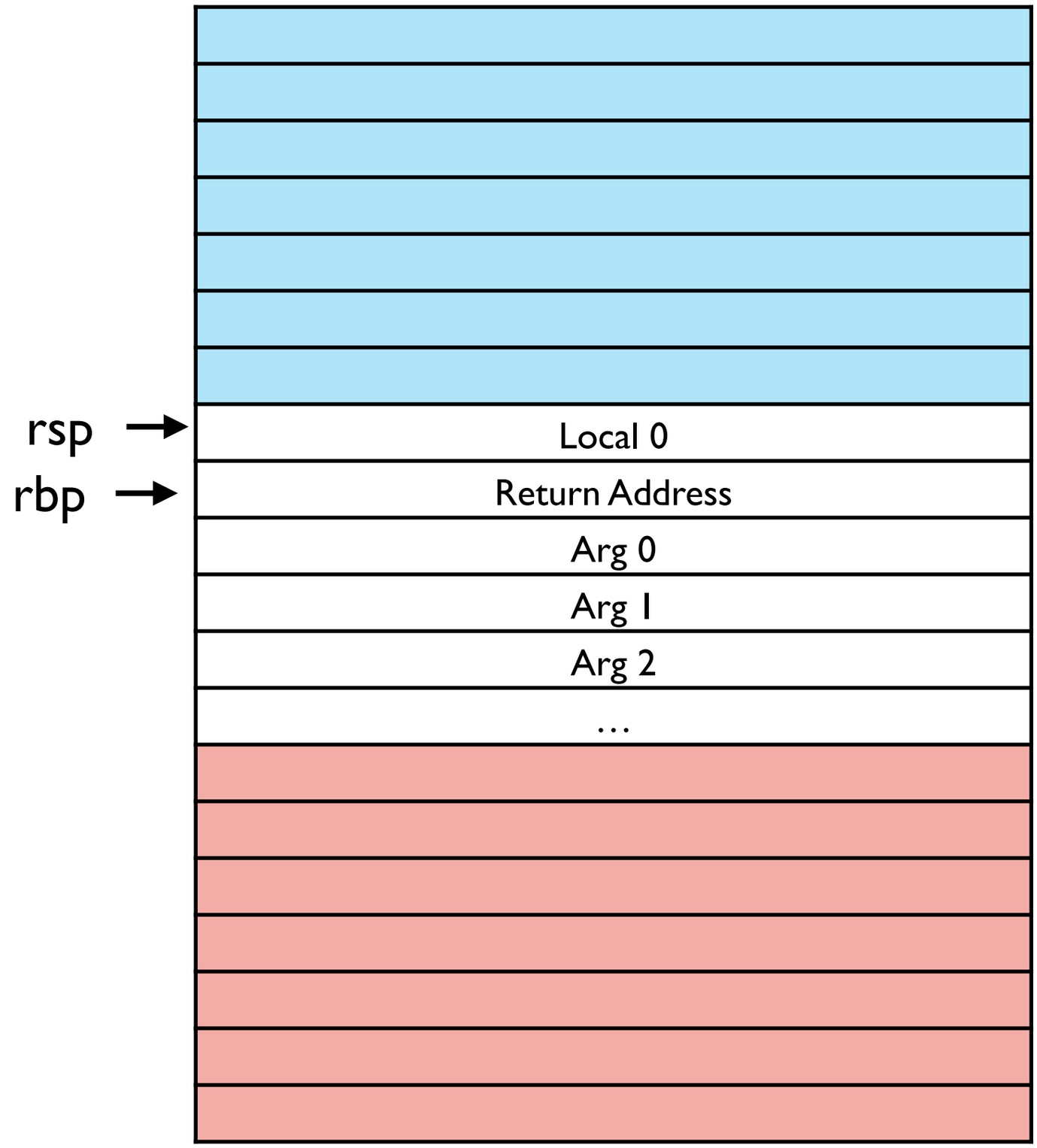


Upon entry to a function

```
start_here:
mov rbp, rsp
sub rsp, 8 * locals
```

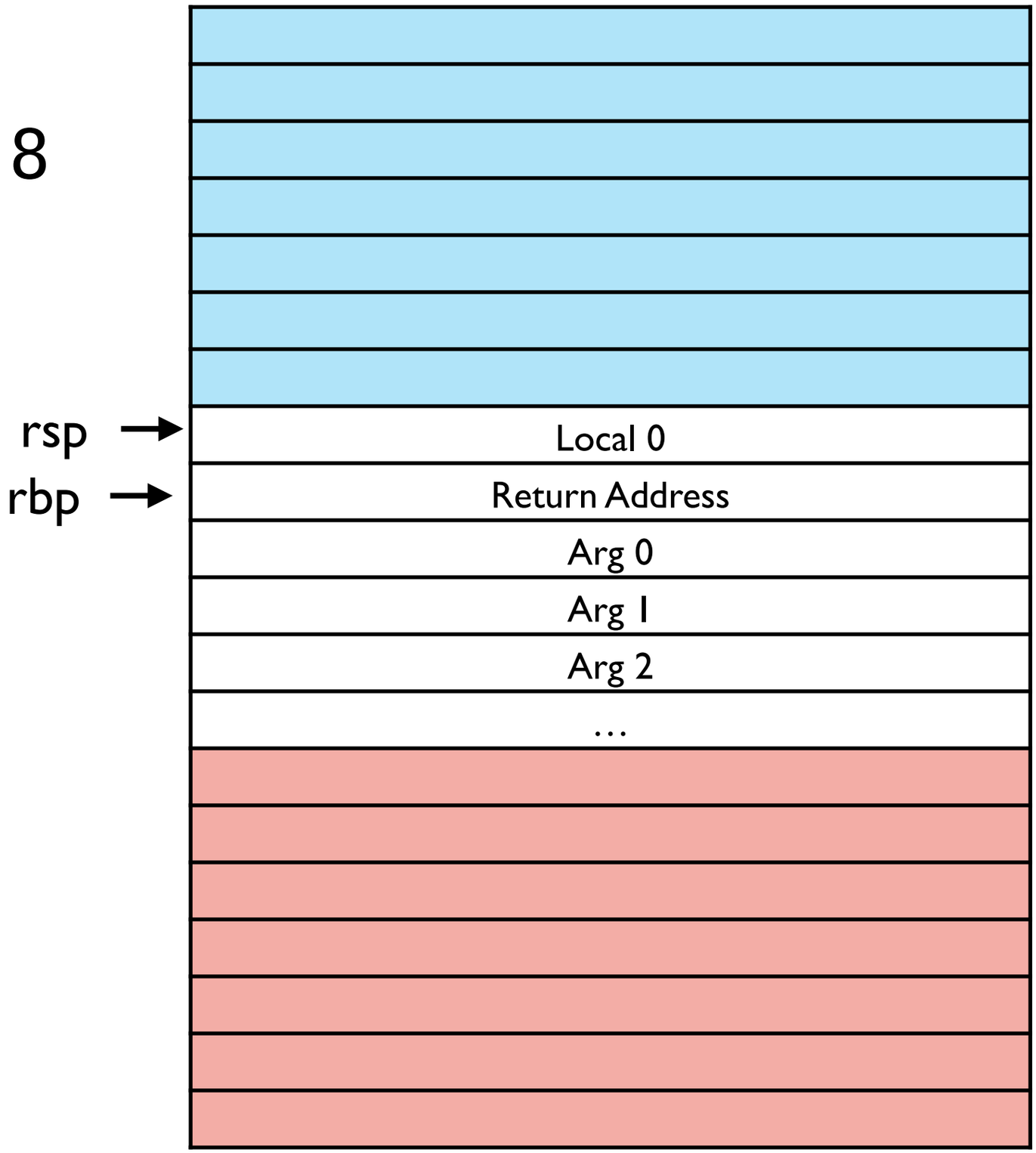

32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments



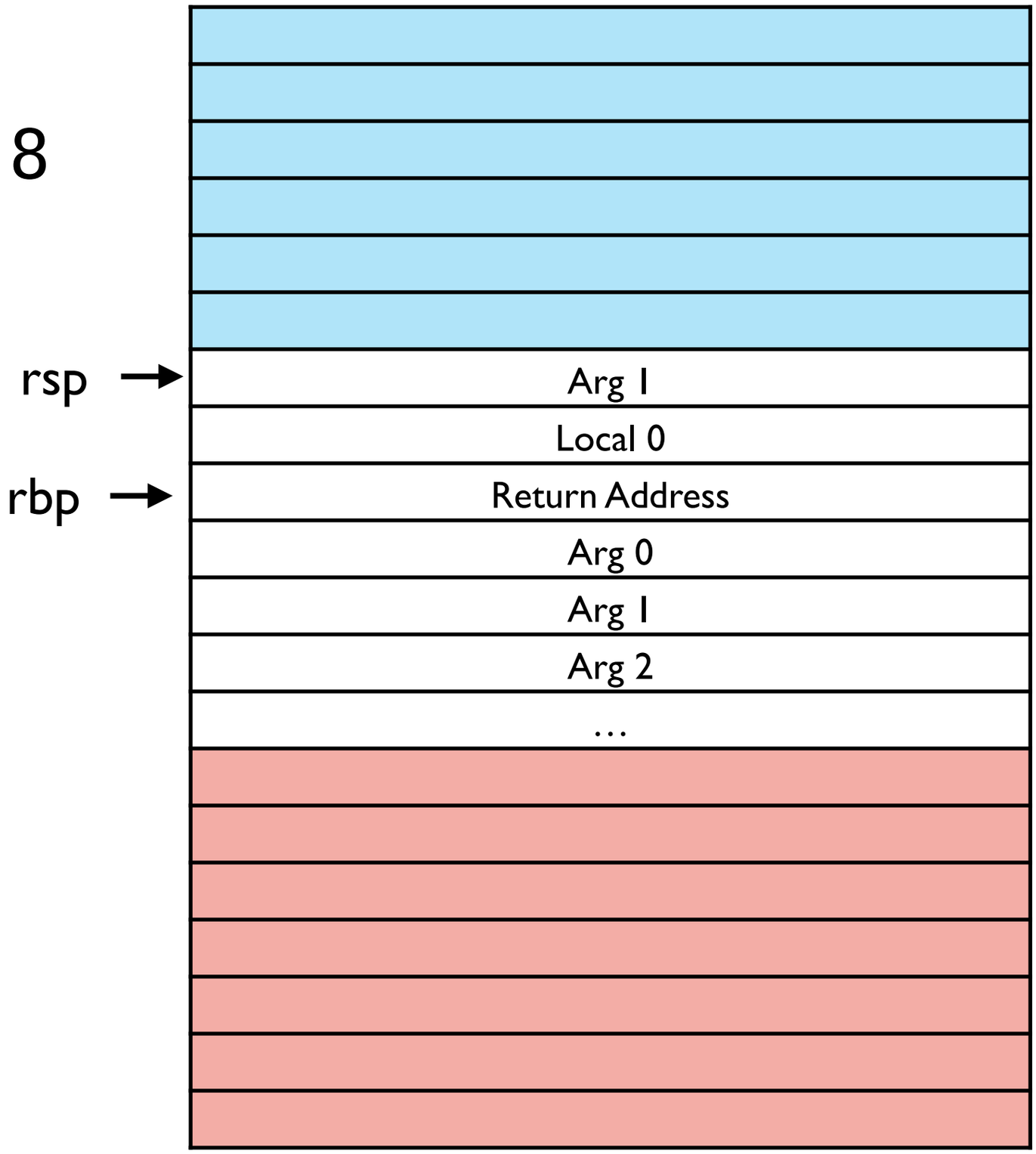
```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



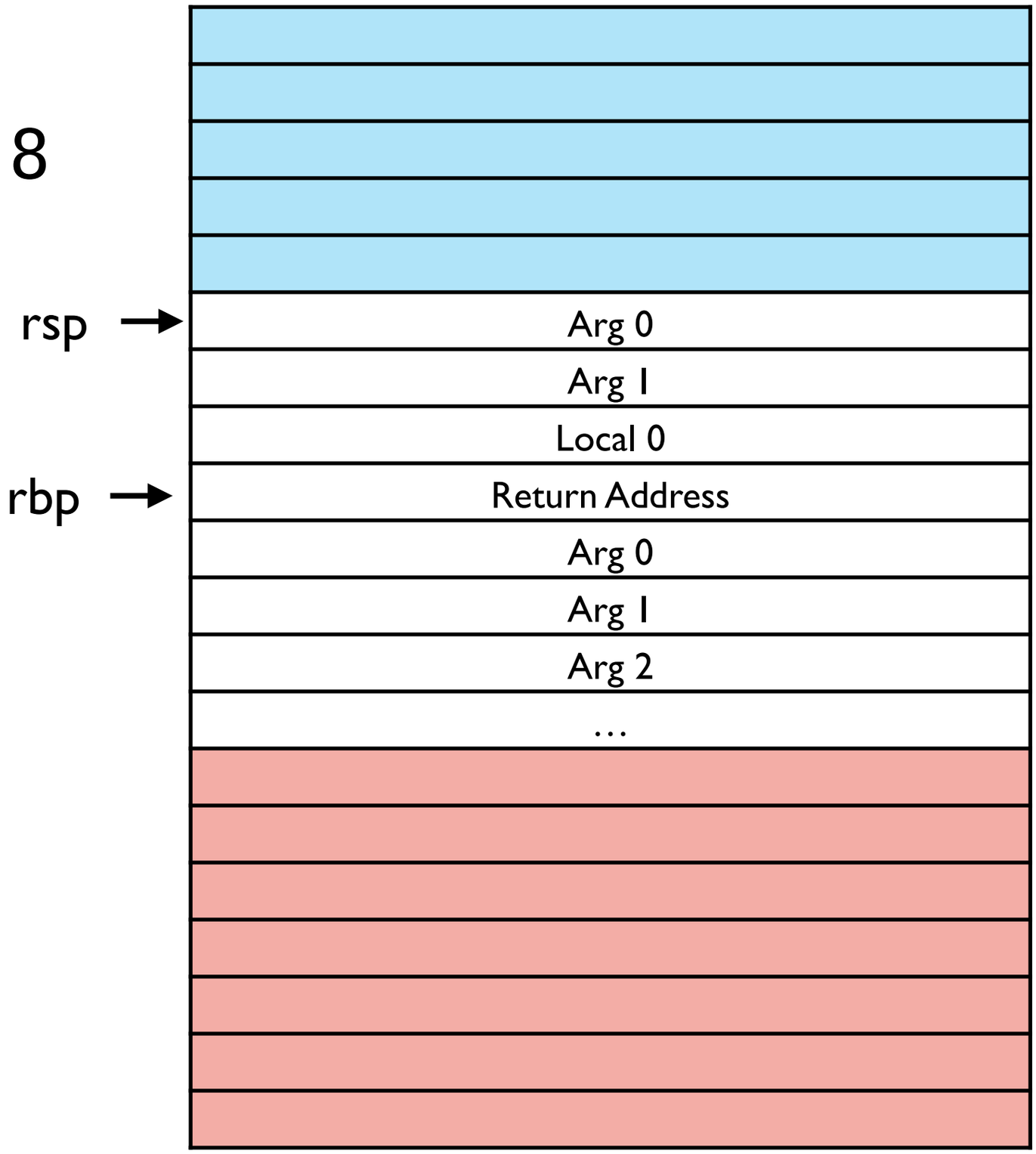
```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
    push arg1
    push arg0
    call fun
    add rsp, 16
```

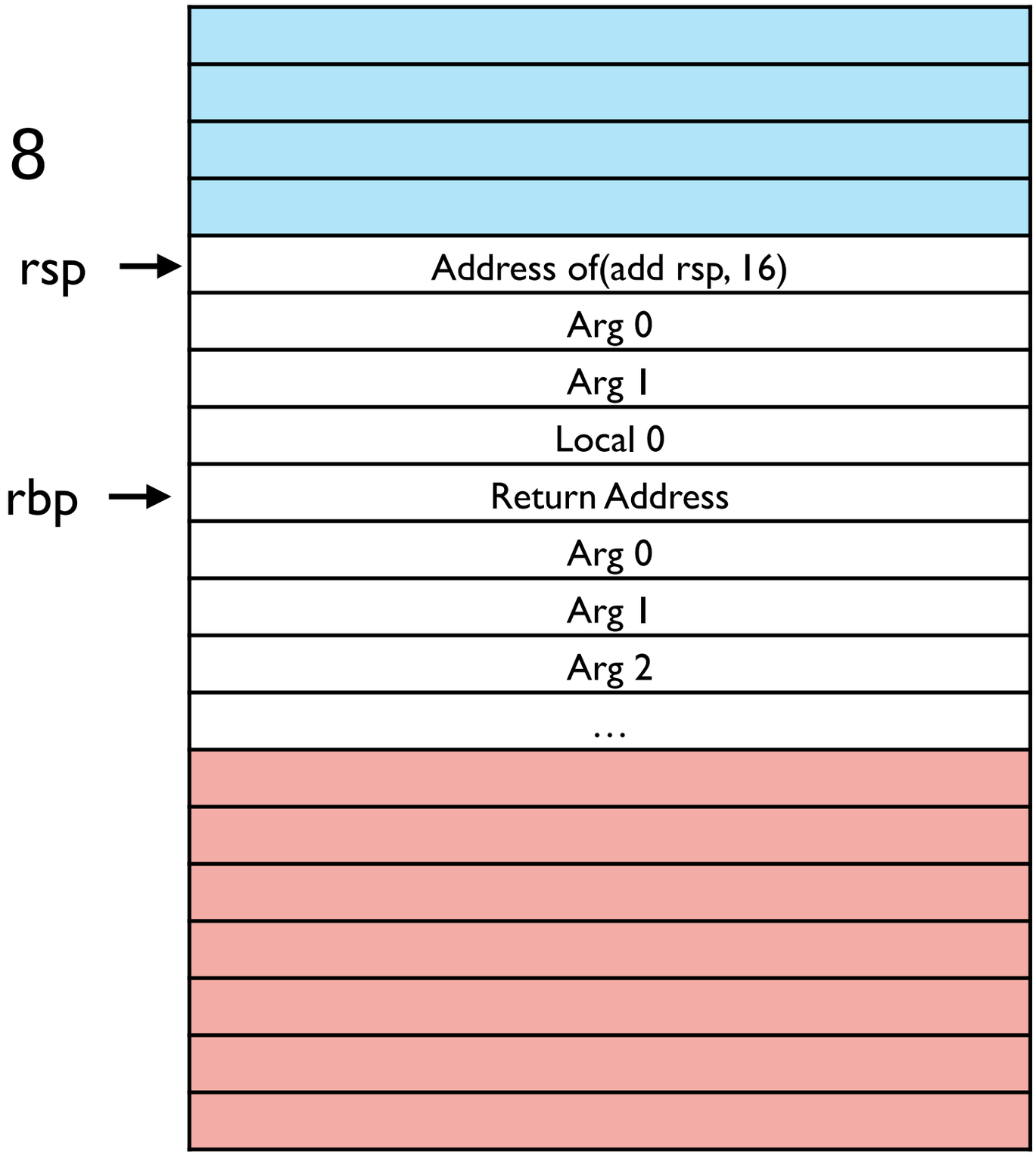
```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
    push arg1
    push arg0
    call fun
    add rsp, 16
```



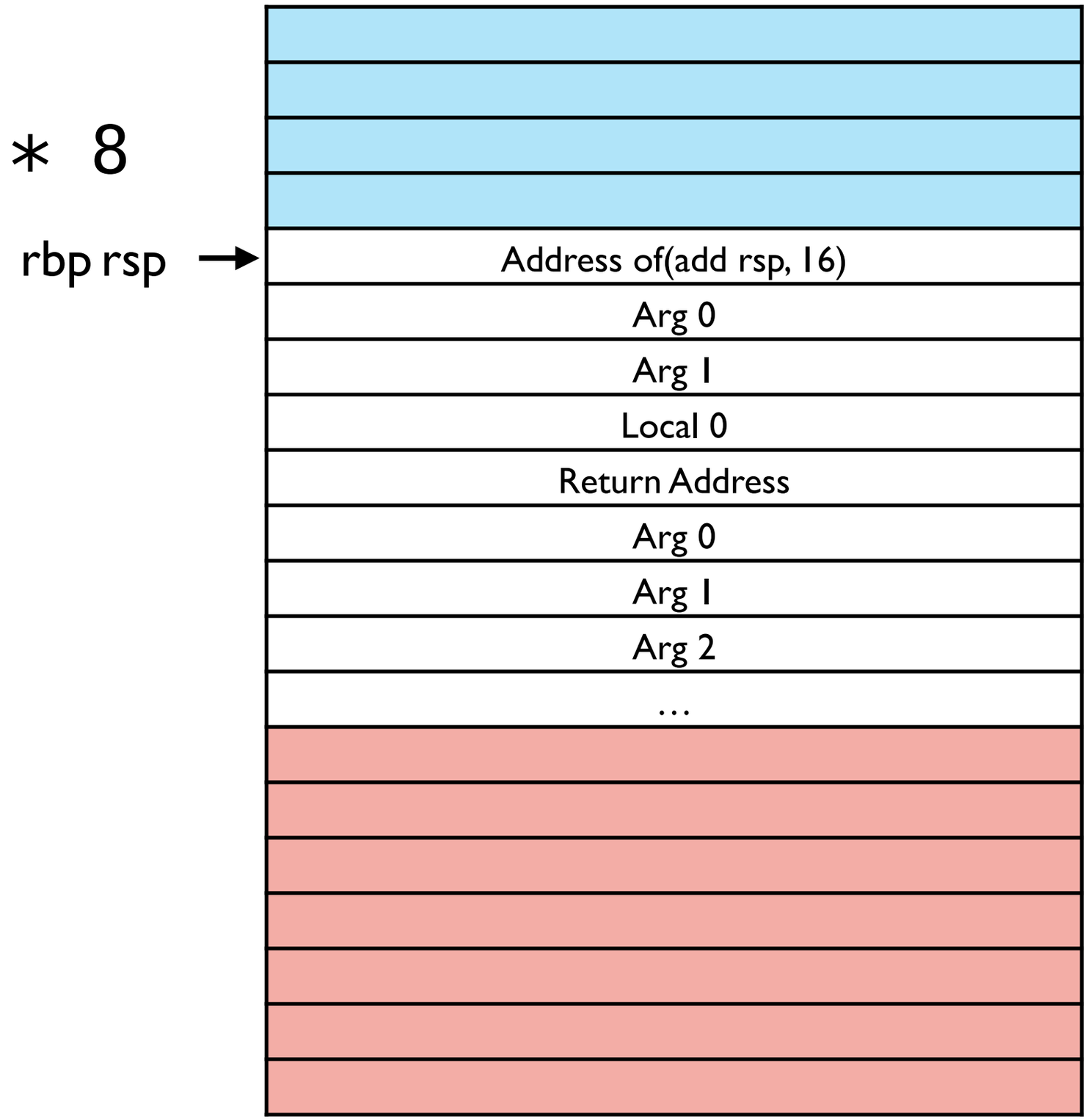
```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

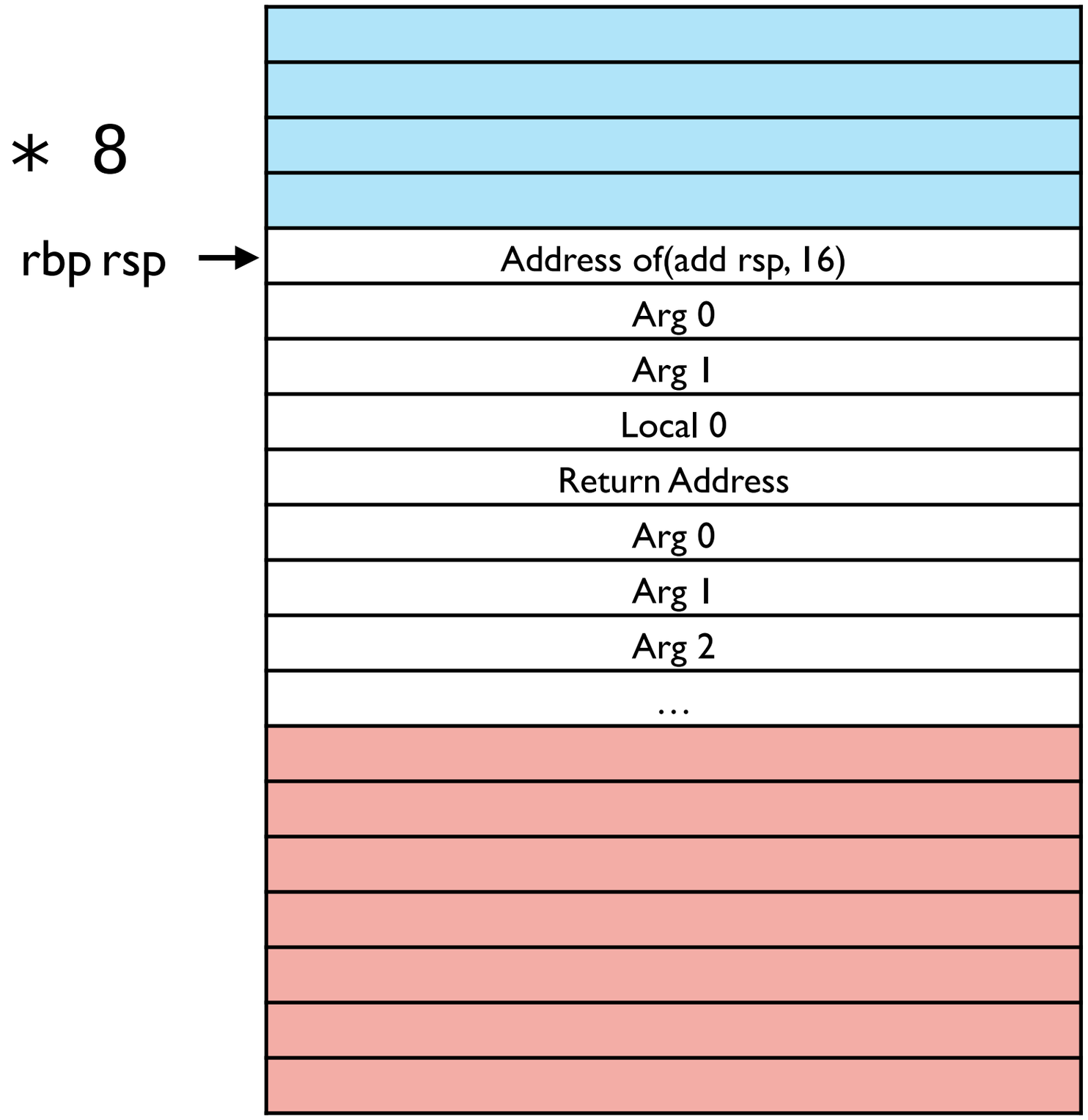
```
push arg1
push arg0
call fun
add rsp, 16
```

```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```

```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



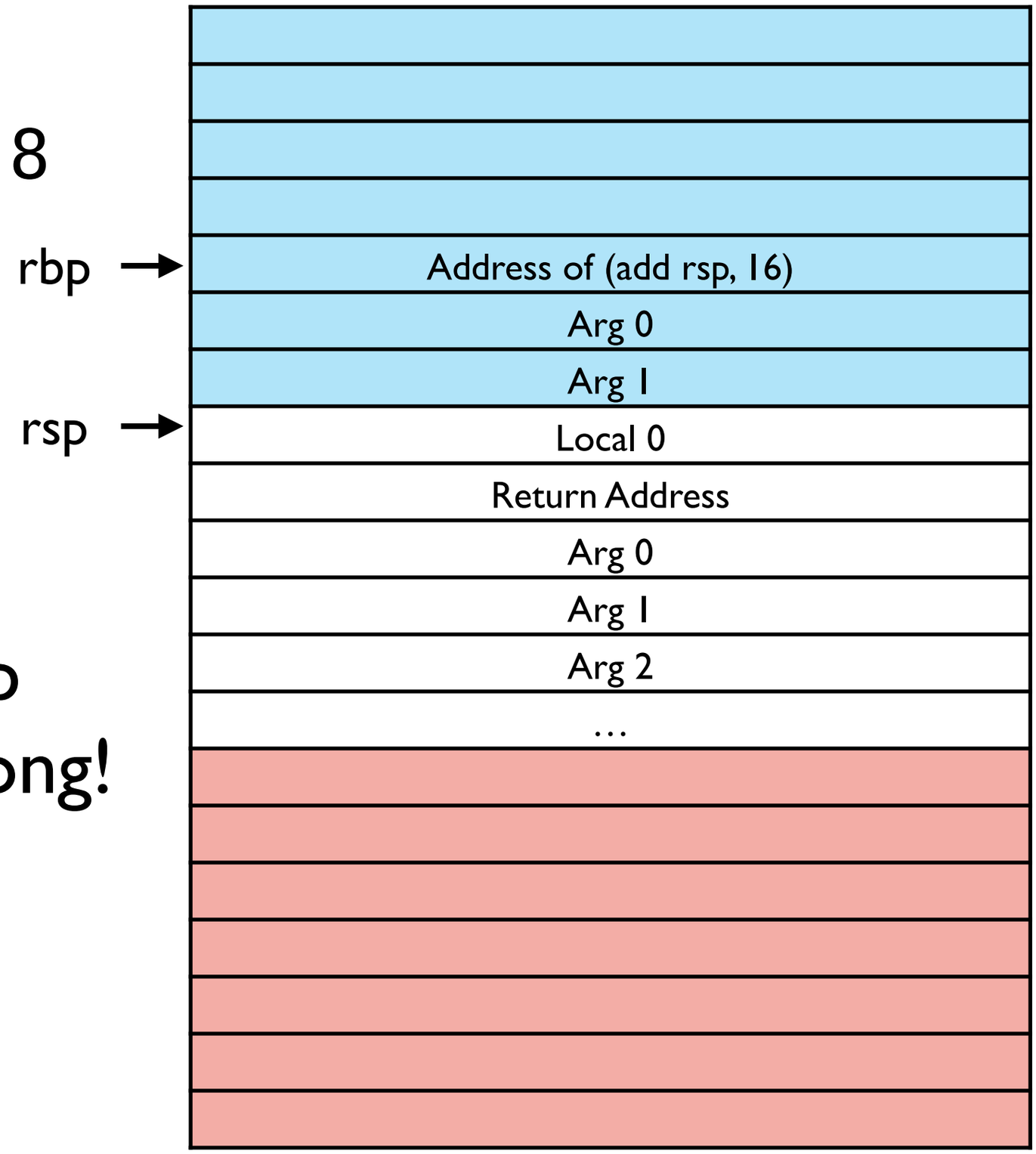
```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

```
    push arg1
    push arg0
    call fun
    add rsp, 16
```


```
fun:
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
ret
```

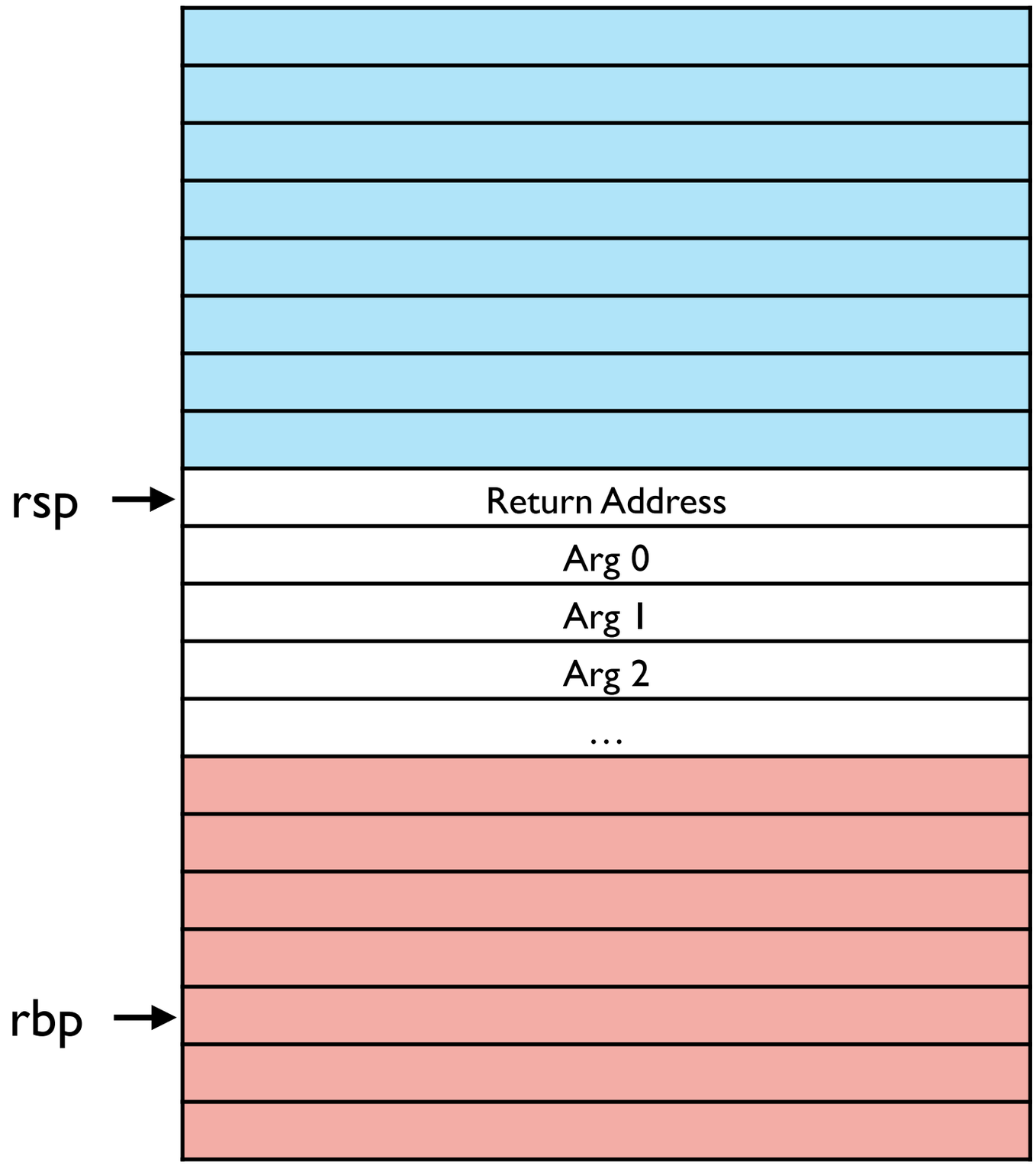
```
push arg1
push arg0
call fun
add rsp, 16
```

rbp is wrong!



32-bit
C Calling Convention

- rbp is a *callee-save* register, meaning the callee is responsible for ensuring the value is restored before returning

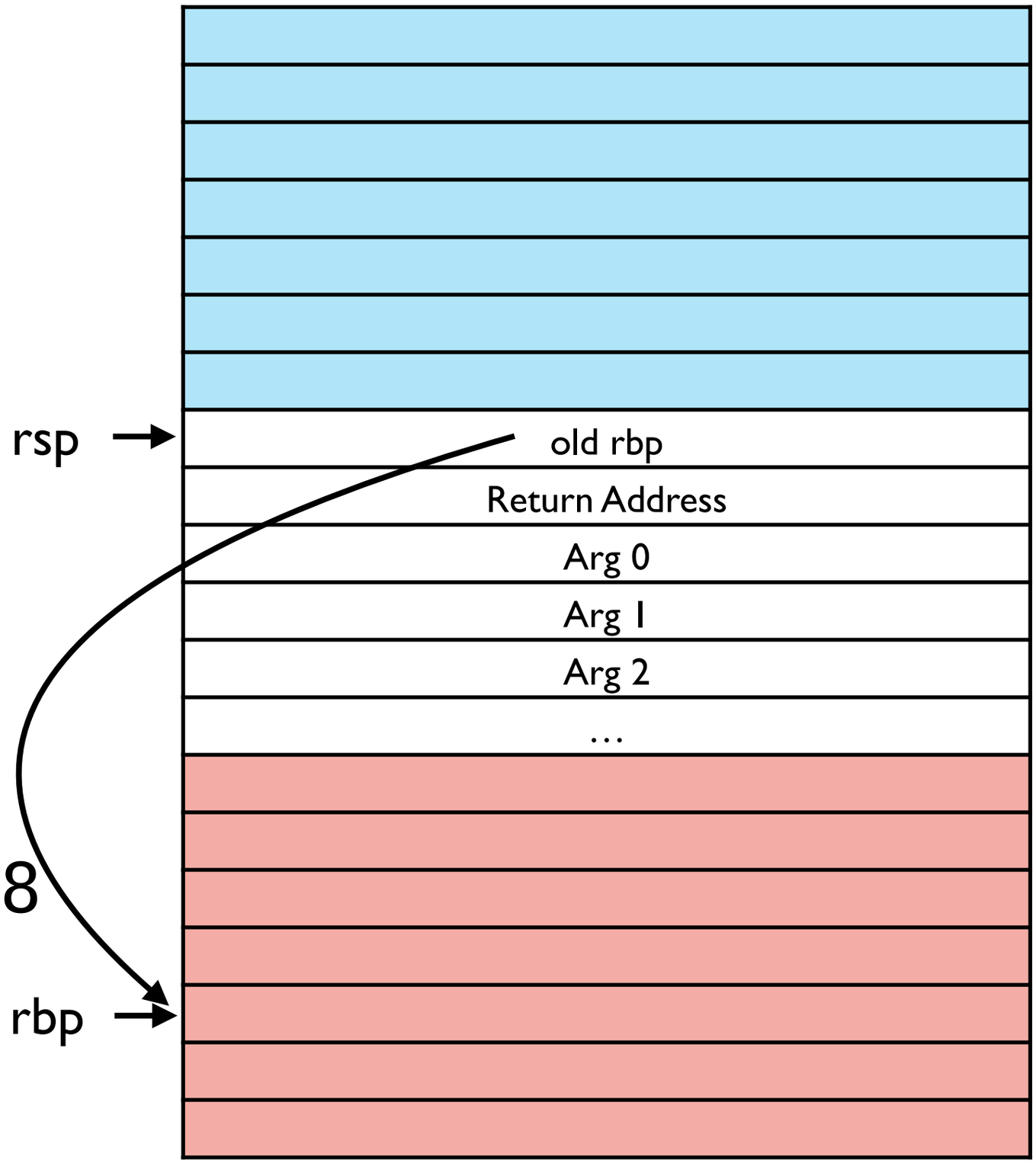


32-bit
C Calling Convention

```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```

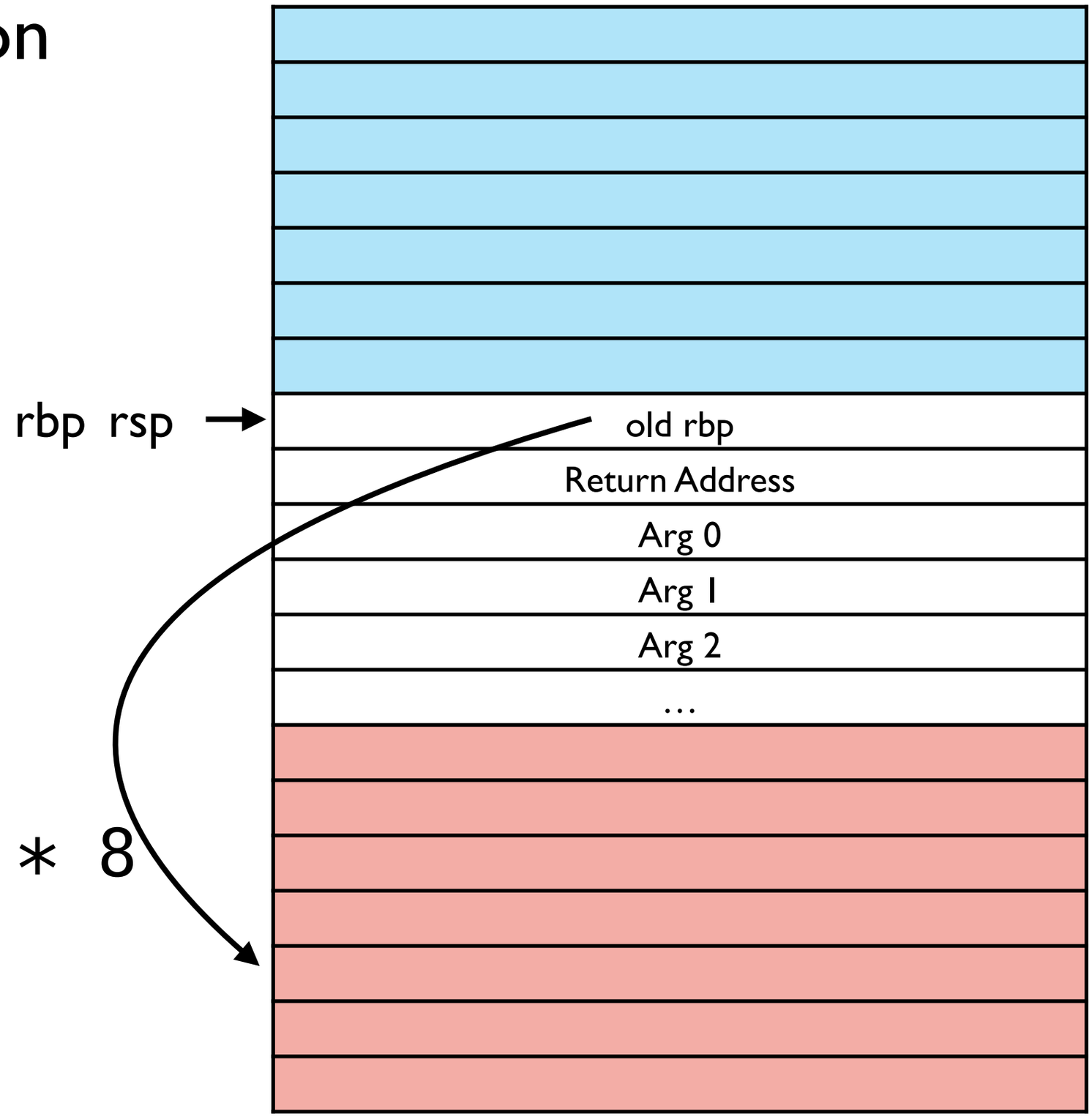
32-bit
C Calling Convention

```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```



32-bit
C Calling Convention

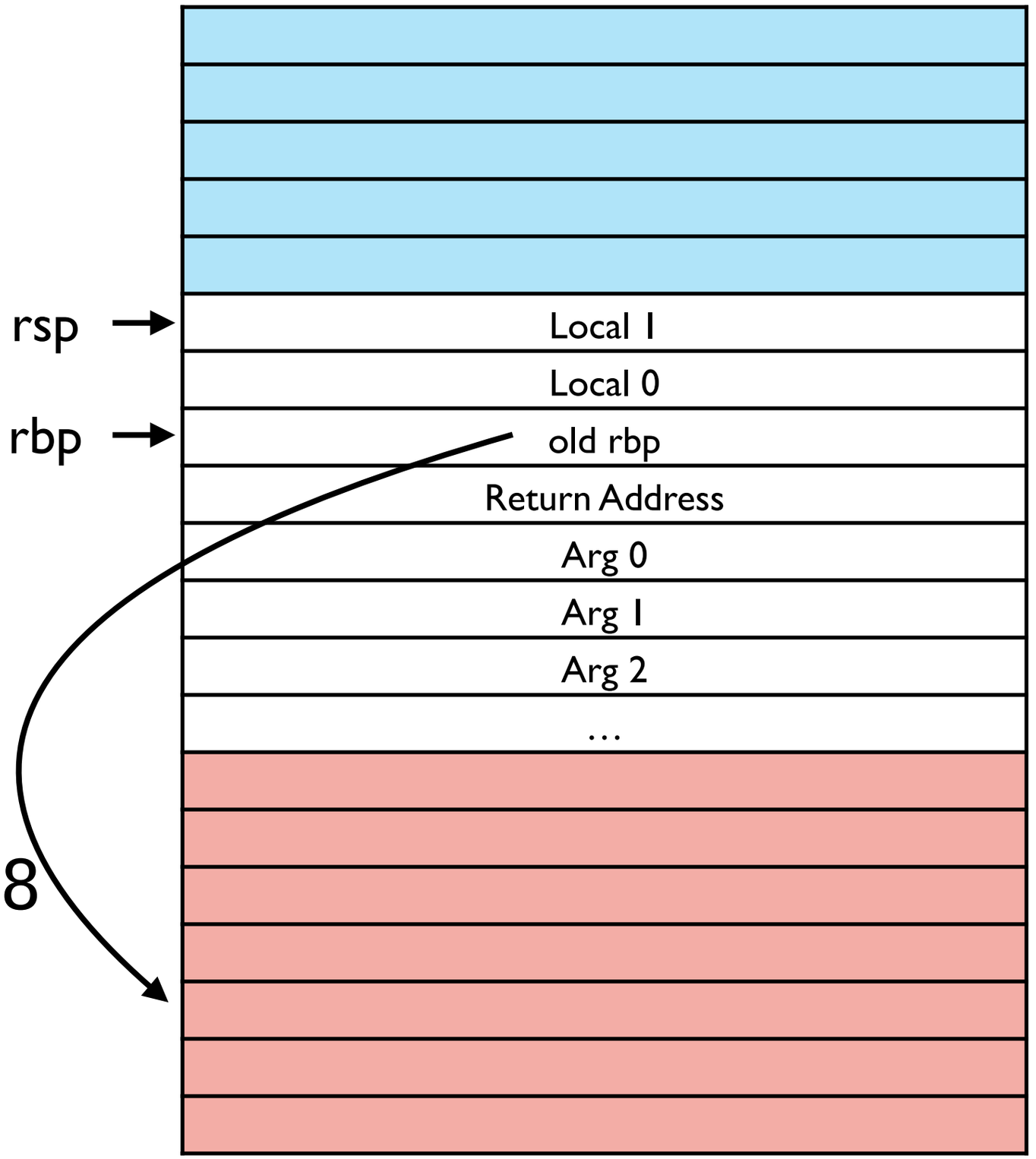
```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

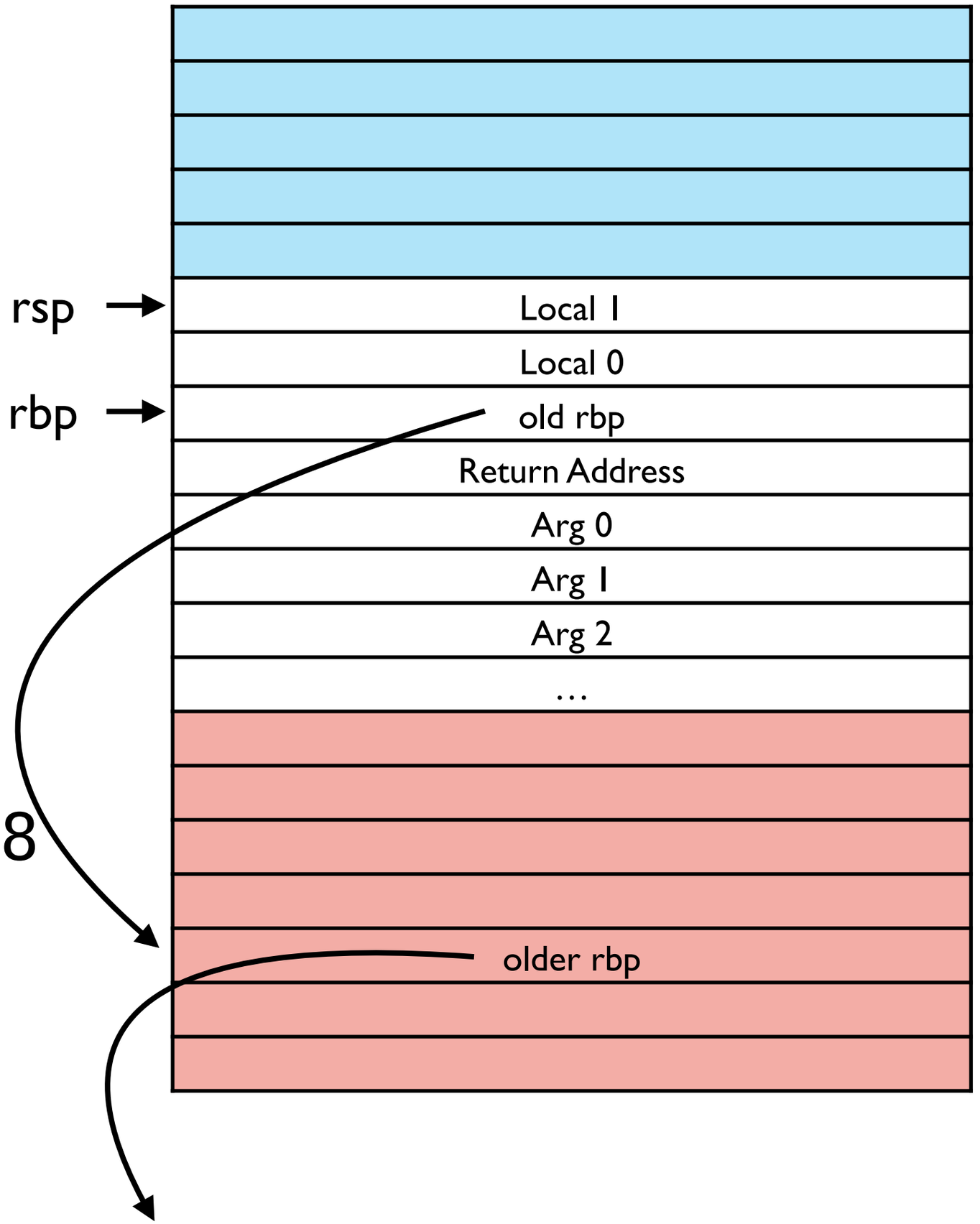
```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```



32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments

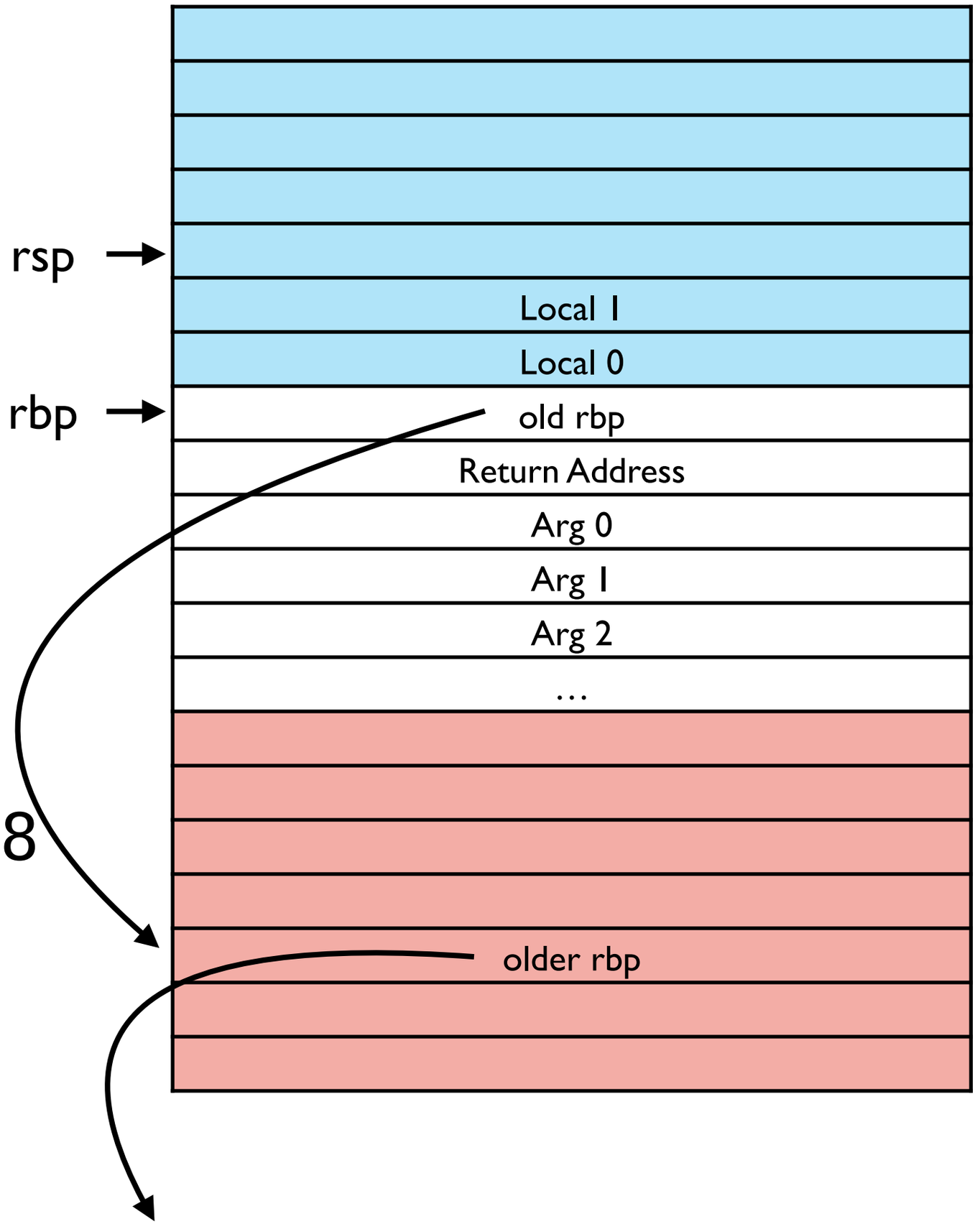
```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```

32-bit
C Calling Convention

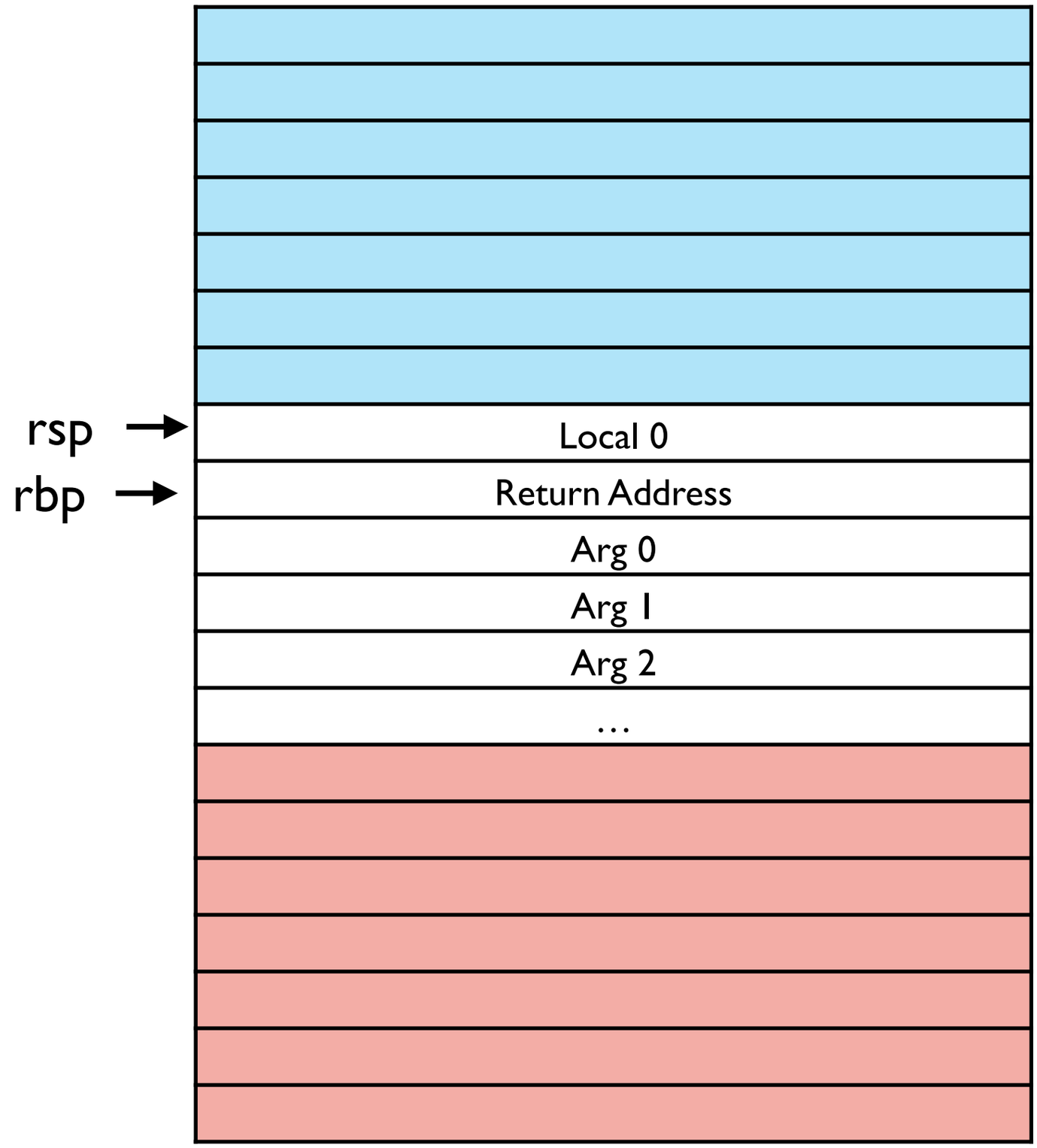
- rbp values form a linked list of stack frames

```
push rbp
mov rbp, rsp
sub rsp, locals * 8
```



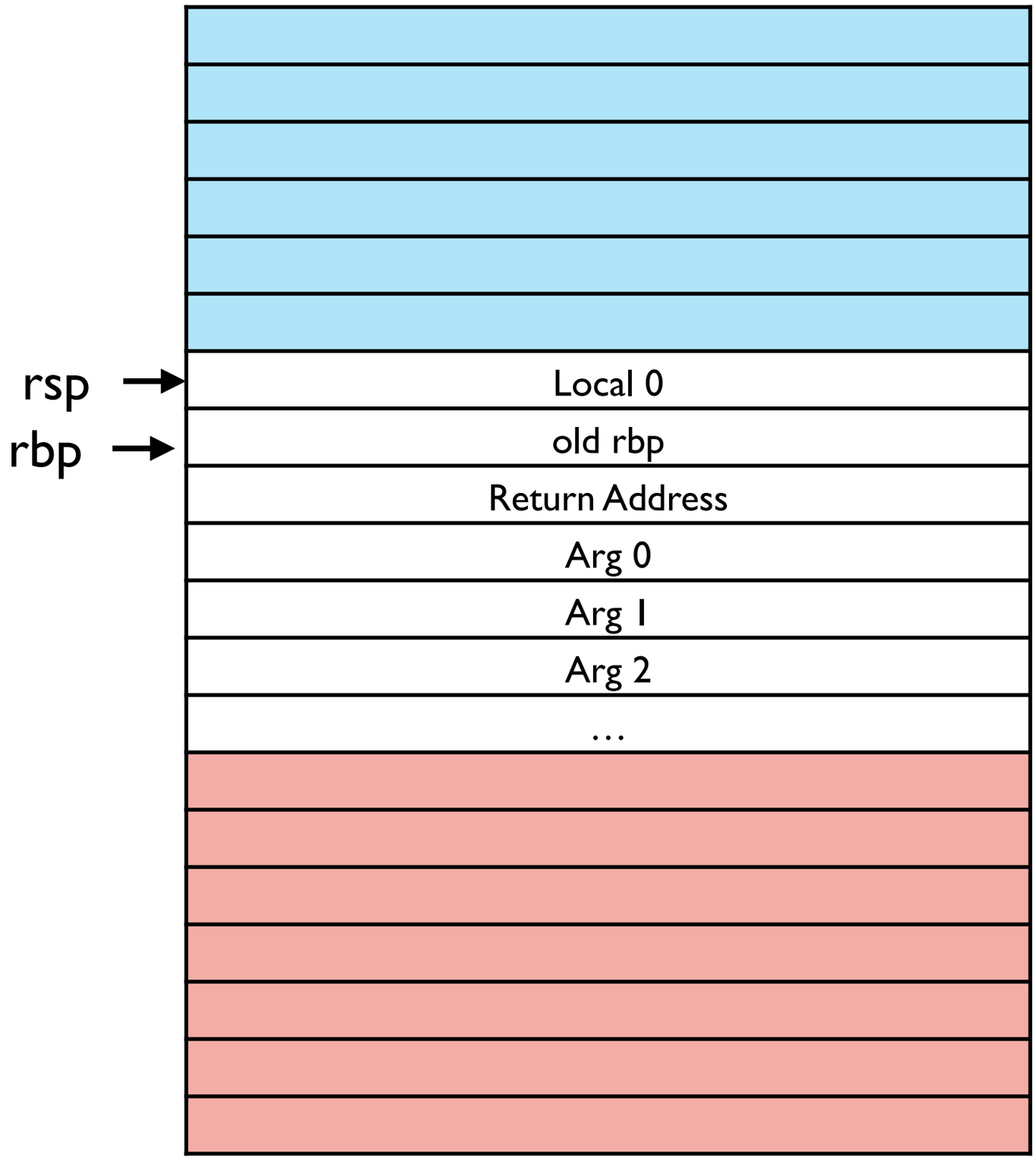
32-bit
C Calling Convention

- example
  - 1 local variable
  - call function of 2 arguments



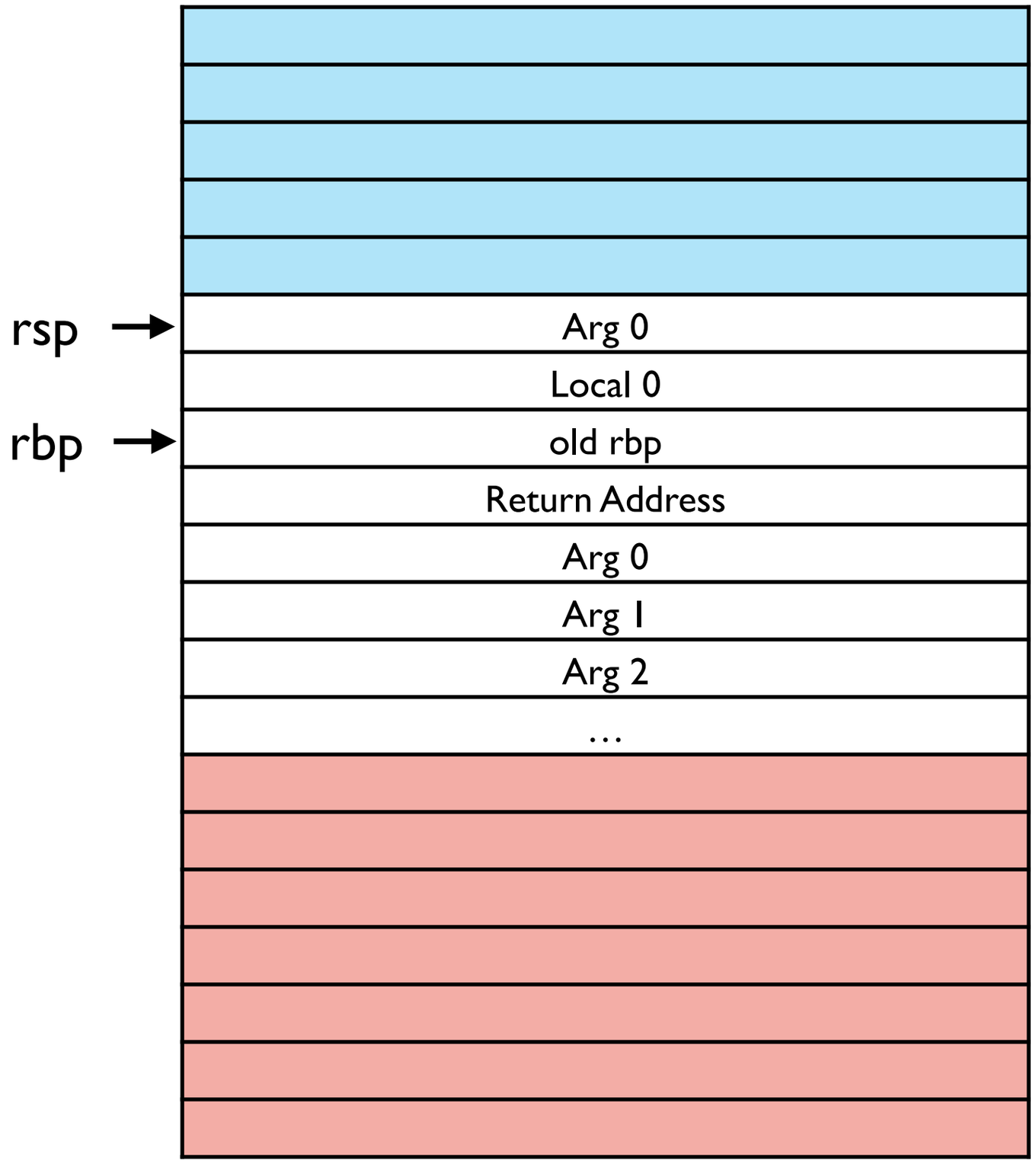
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret

push arg1
push arg0
call fun
add rsp, 16
```



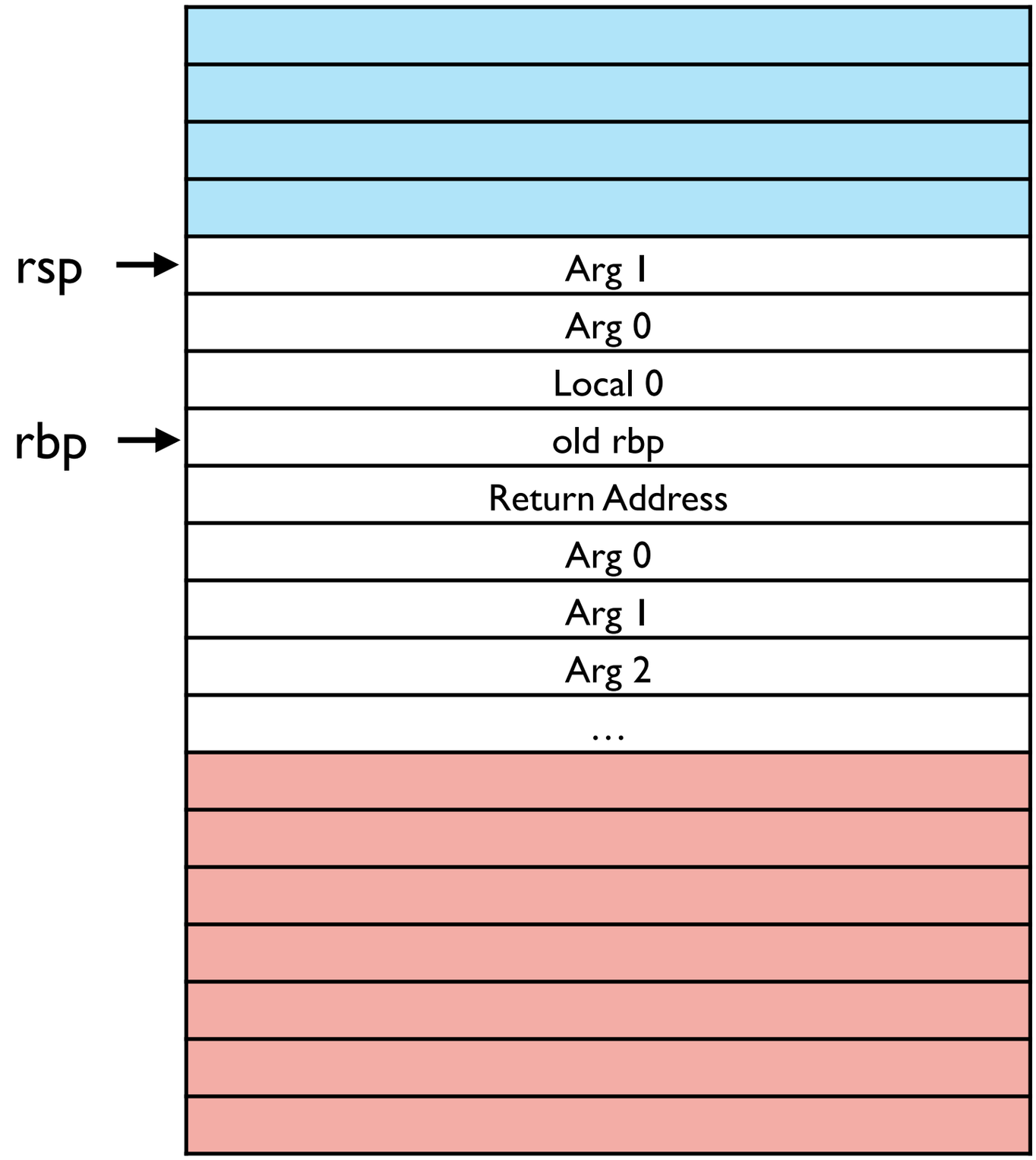
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret


  push arg1
  push arg0
  call fun
  add rsp, 16
```



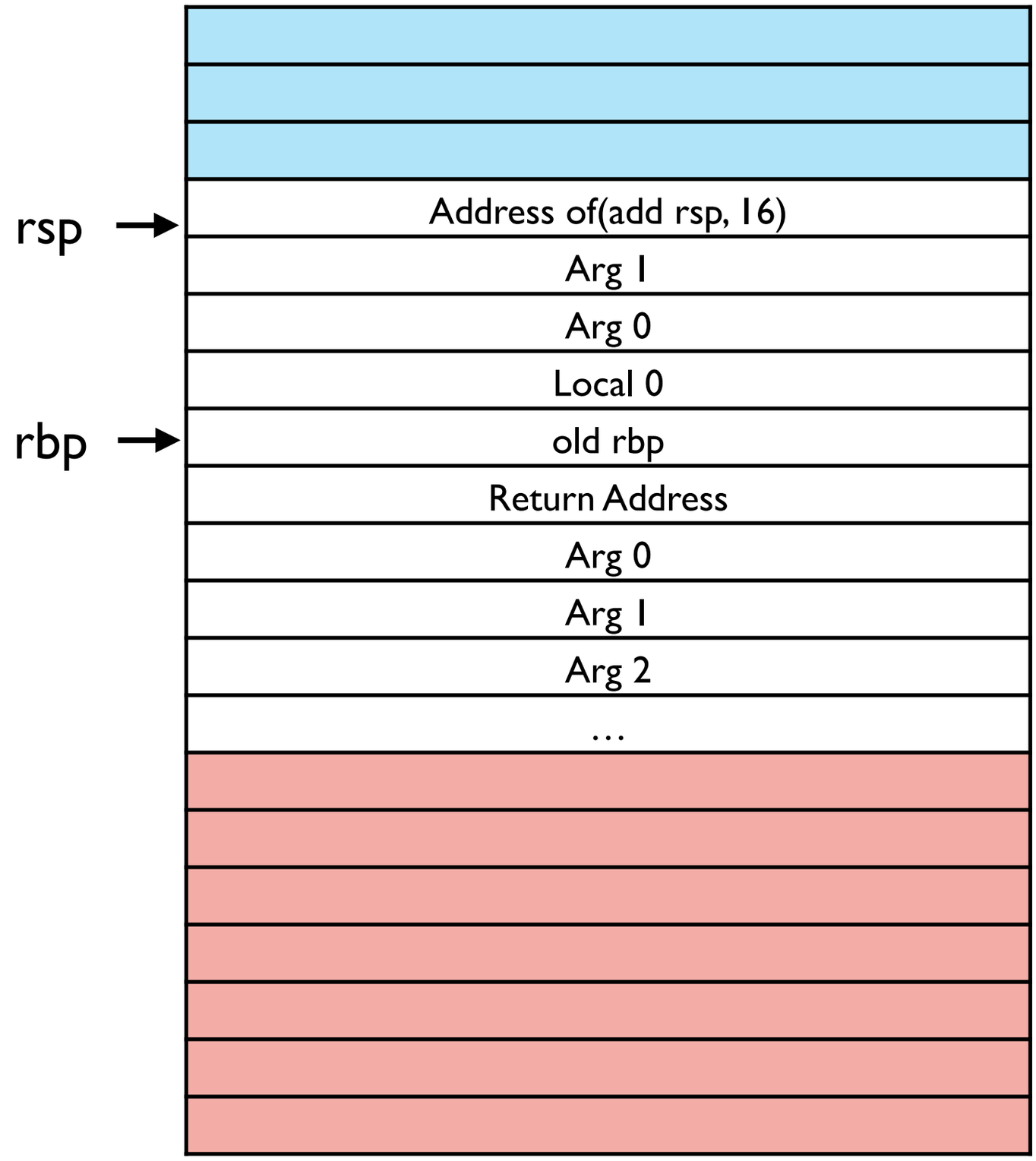
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret


  push arg1
  push arg0
  call fun
  add rsp, 16
```



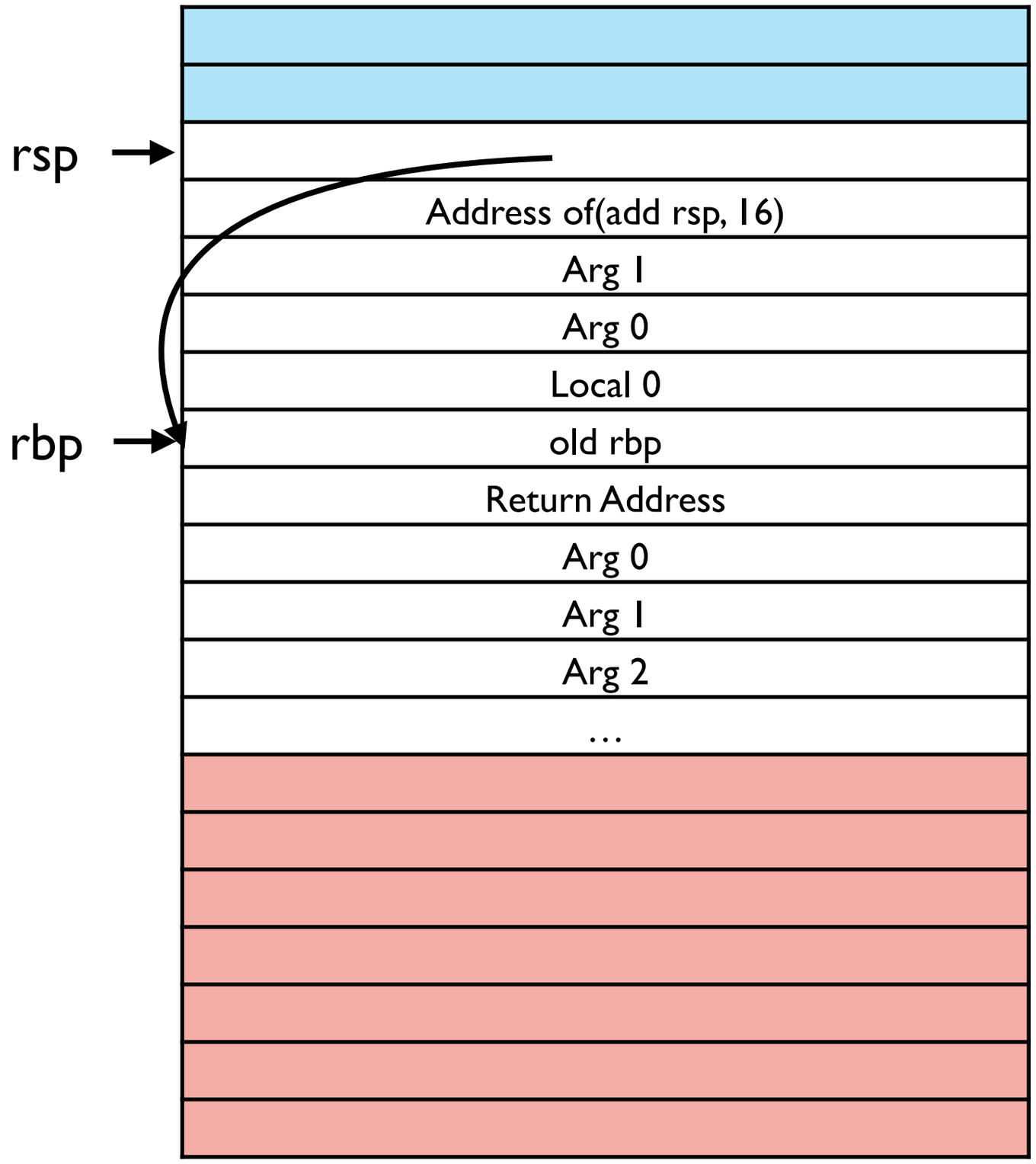
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



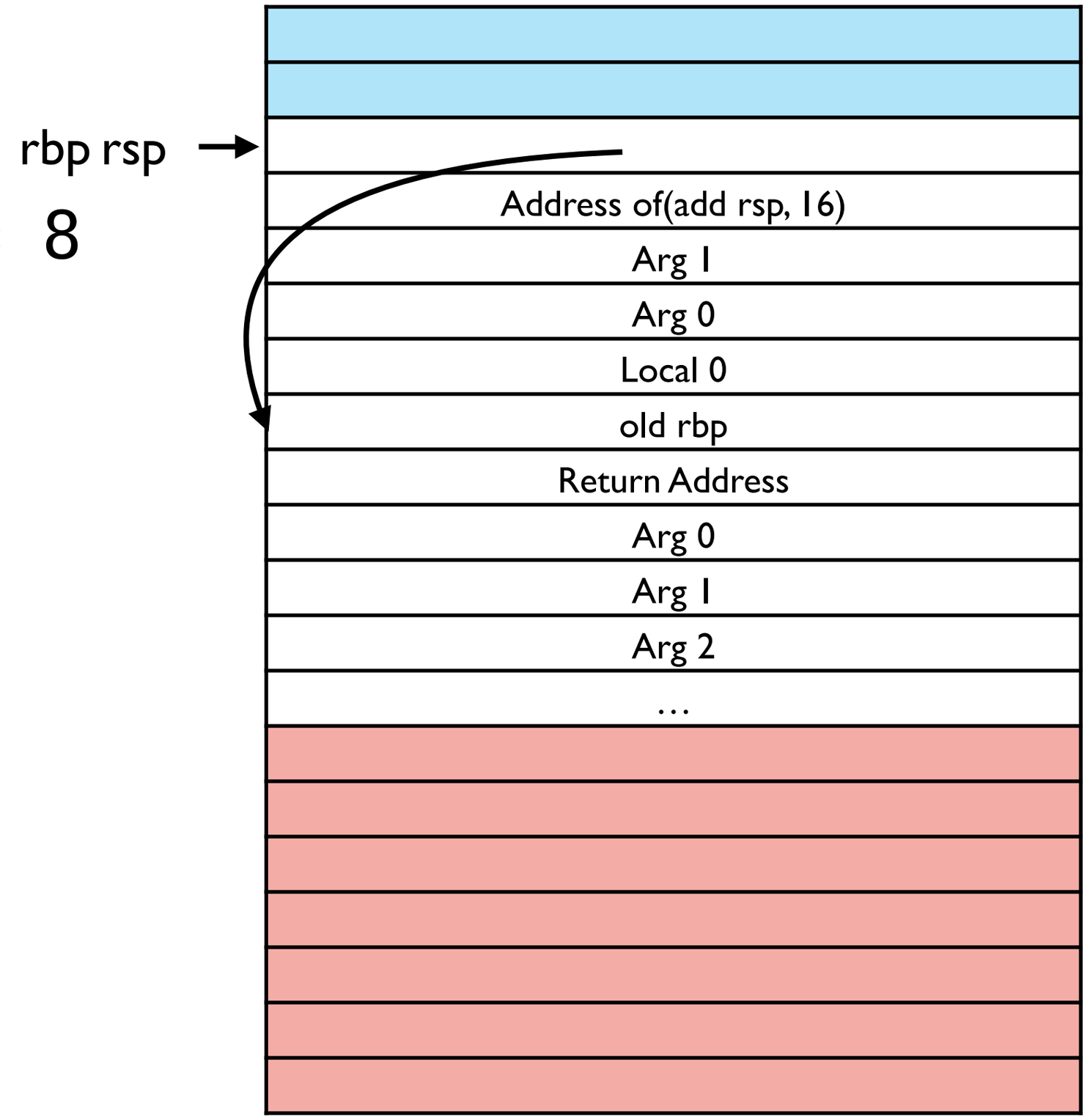
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



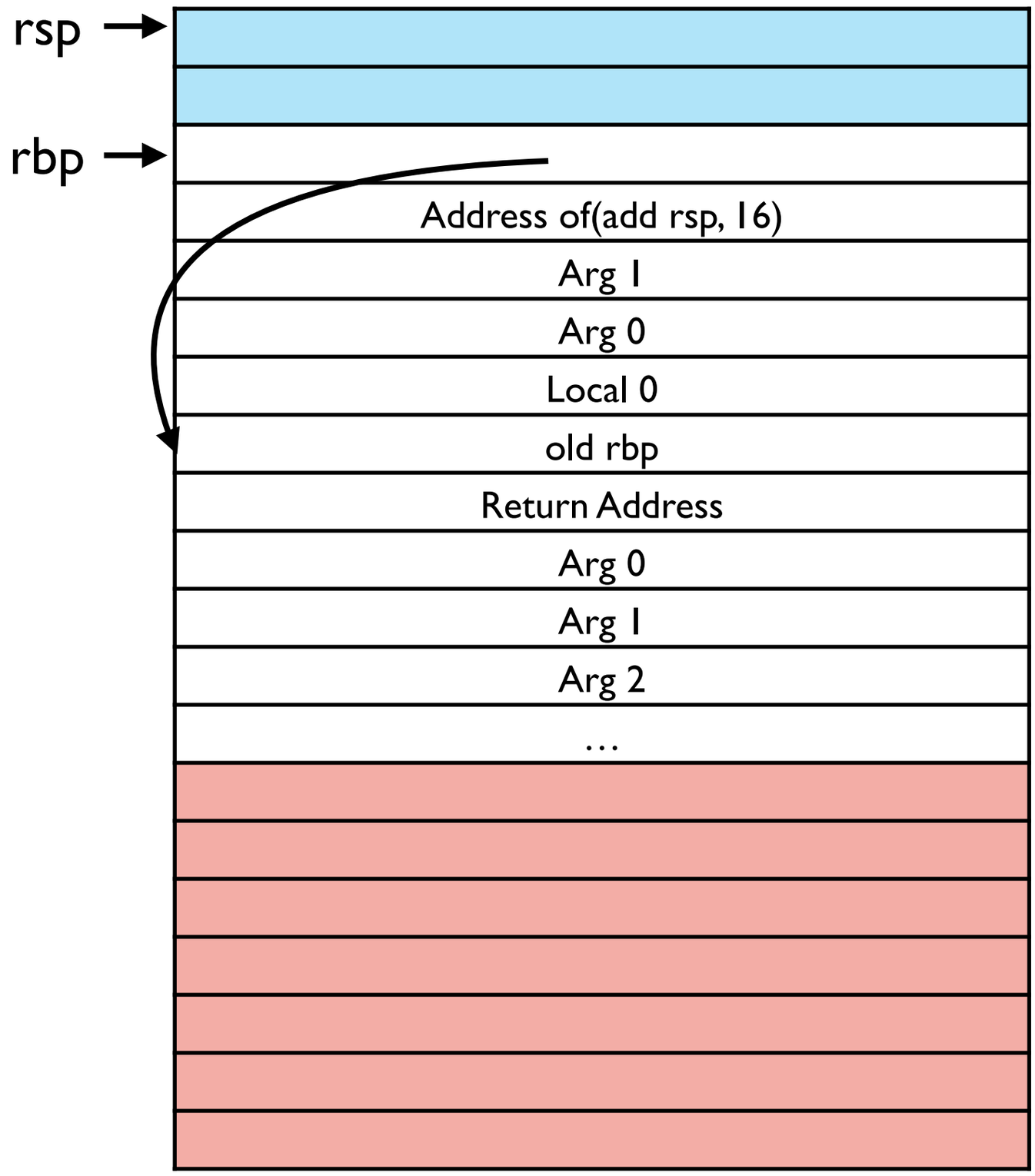
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



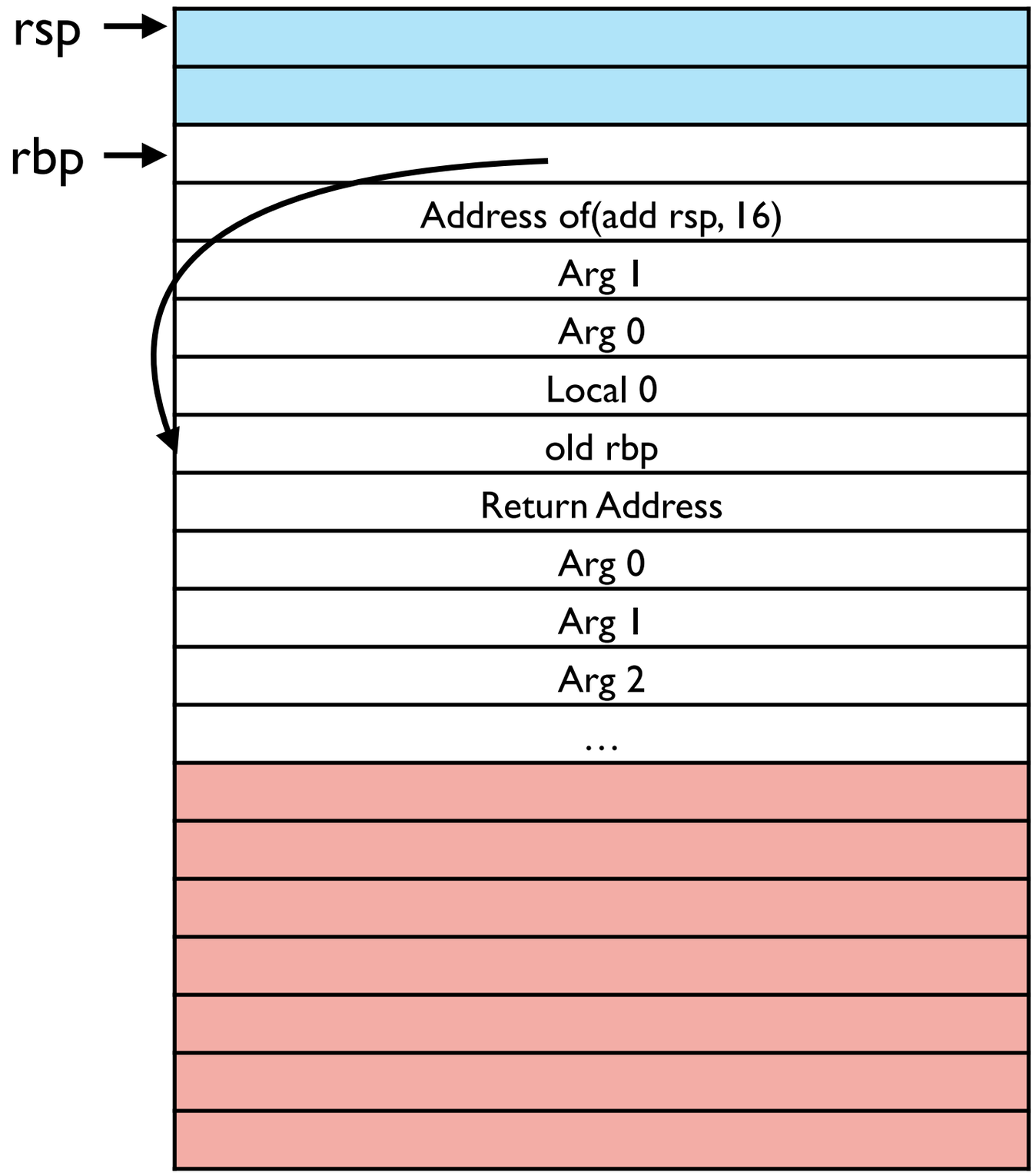
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
  push arg1
  push arg0
  call fun
  add rsp, 16
```



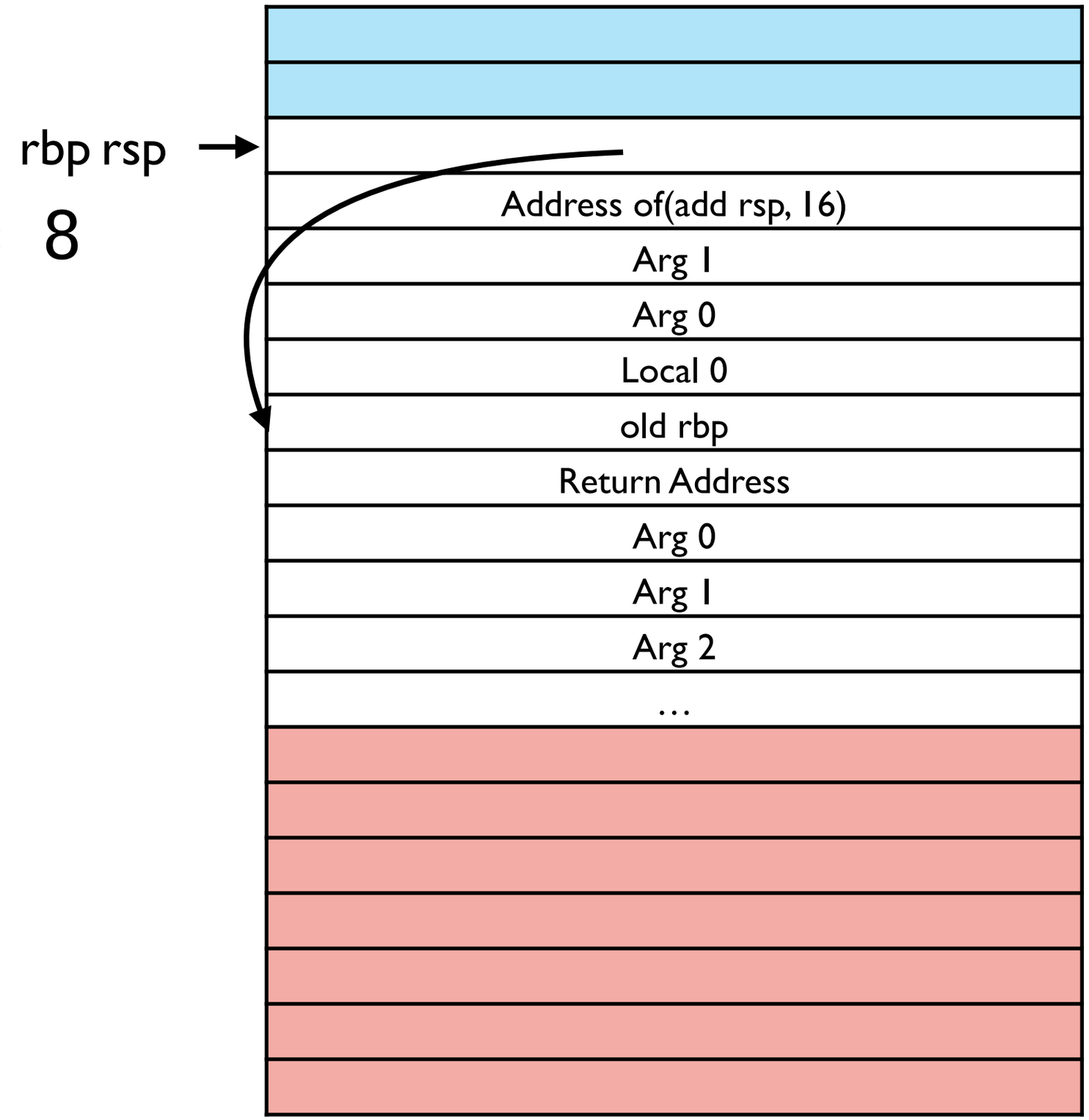
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
  push arg1
  push arg0
  call fun
  add rsp, 16
```



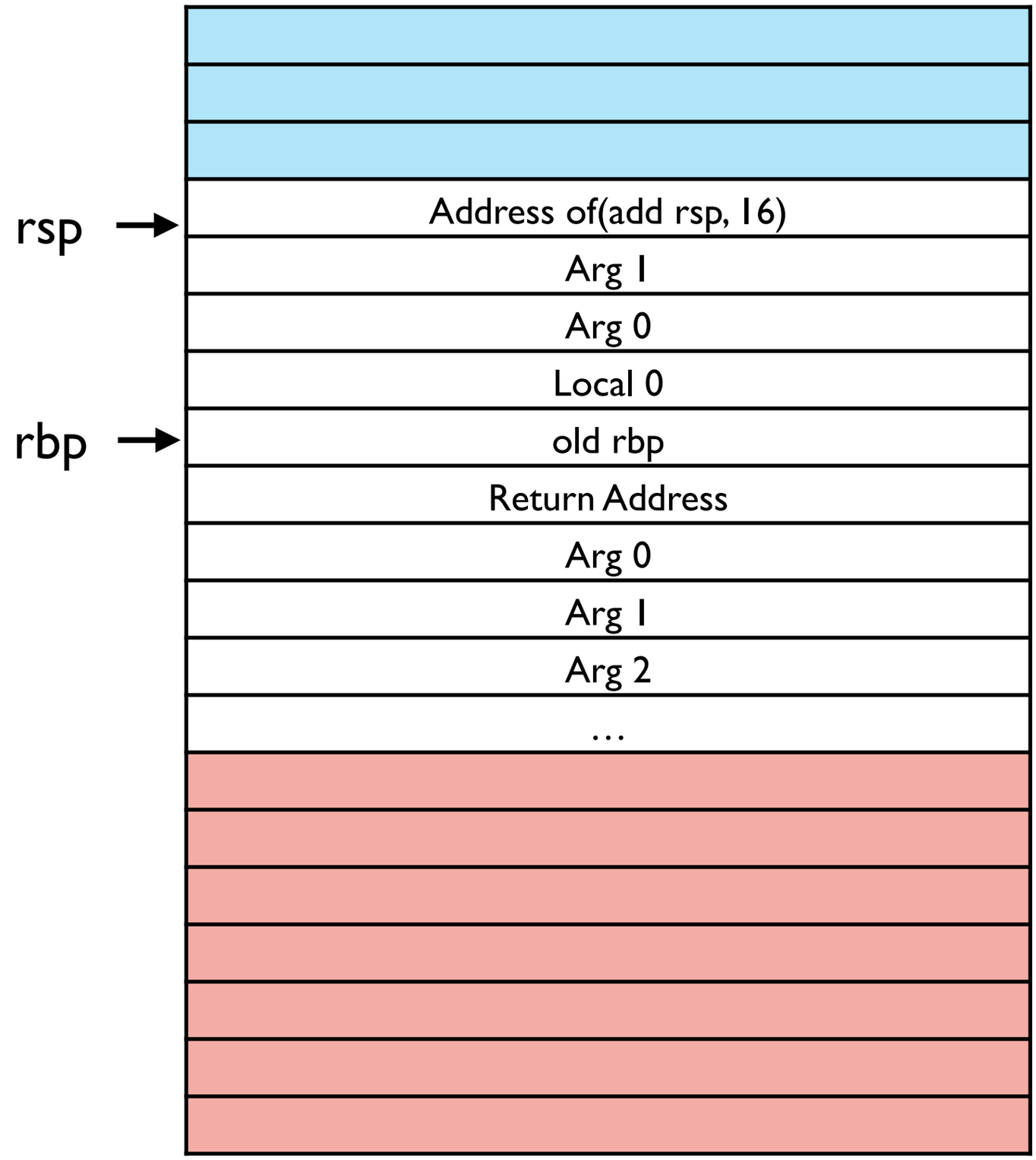
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



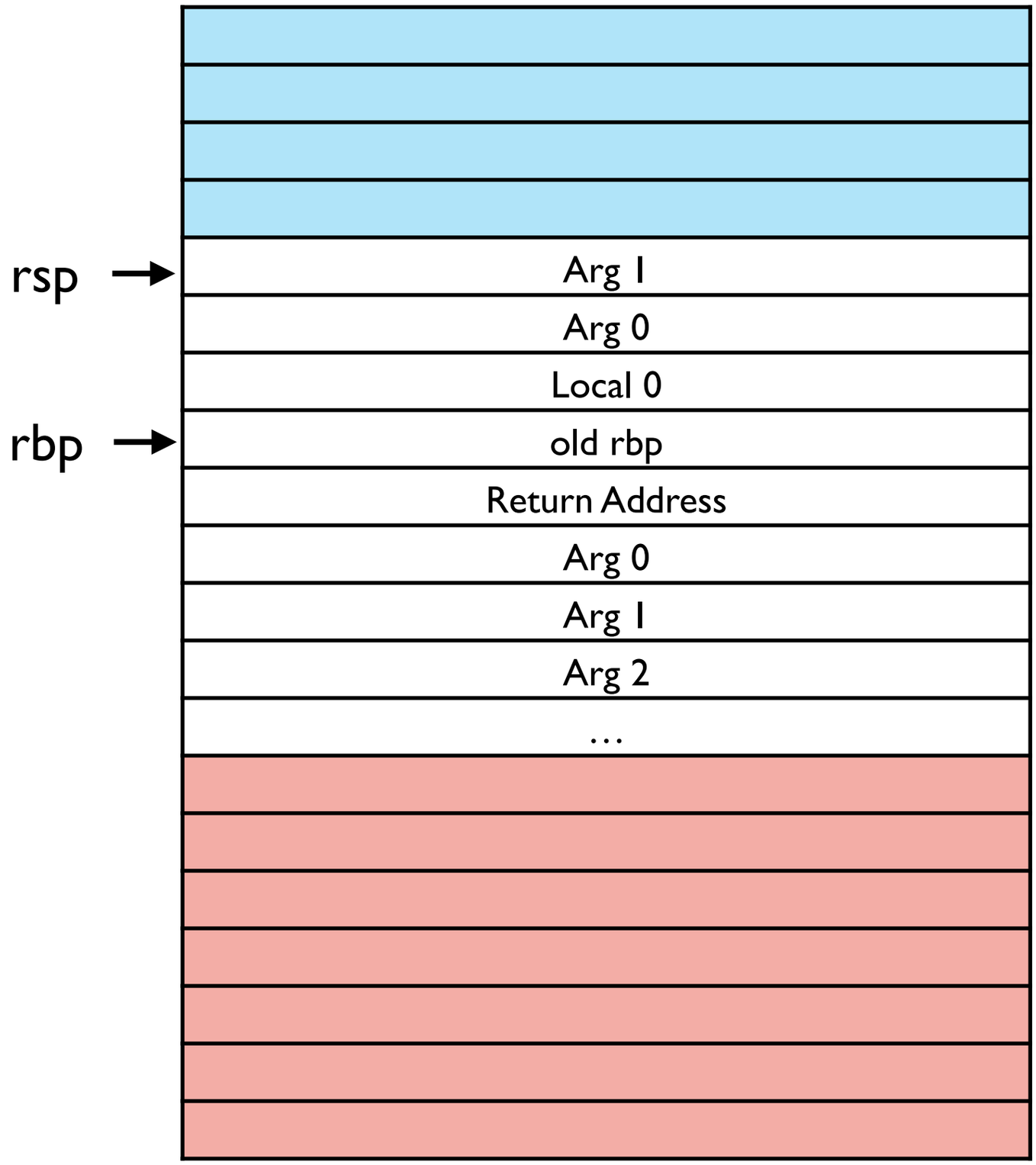
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
push arg1
push arg0
call fun
add rsp, 16
```



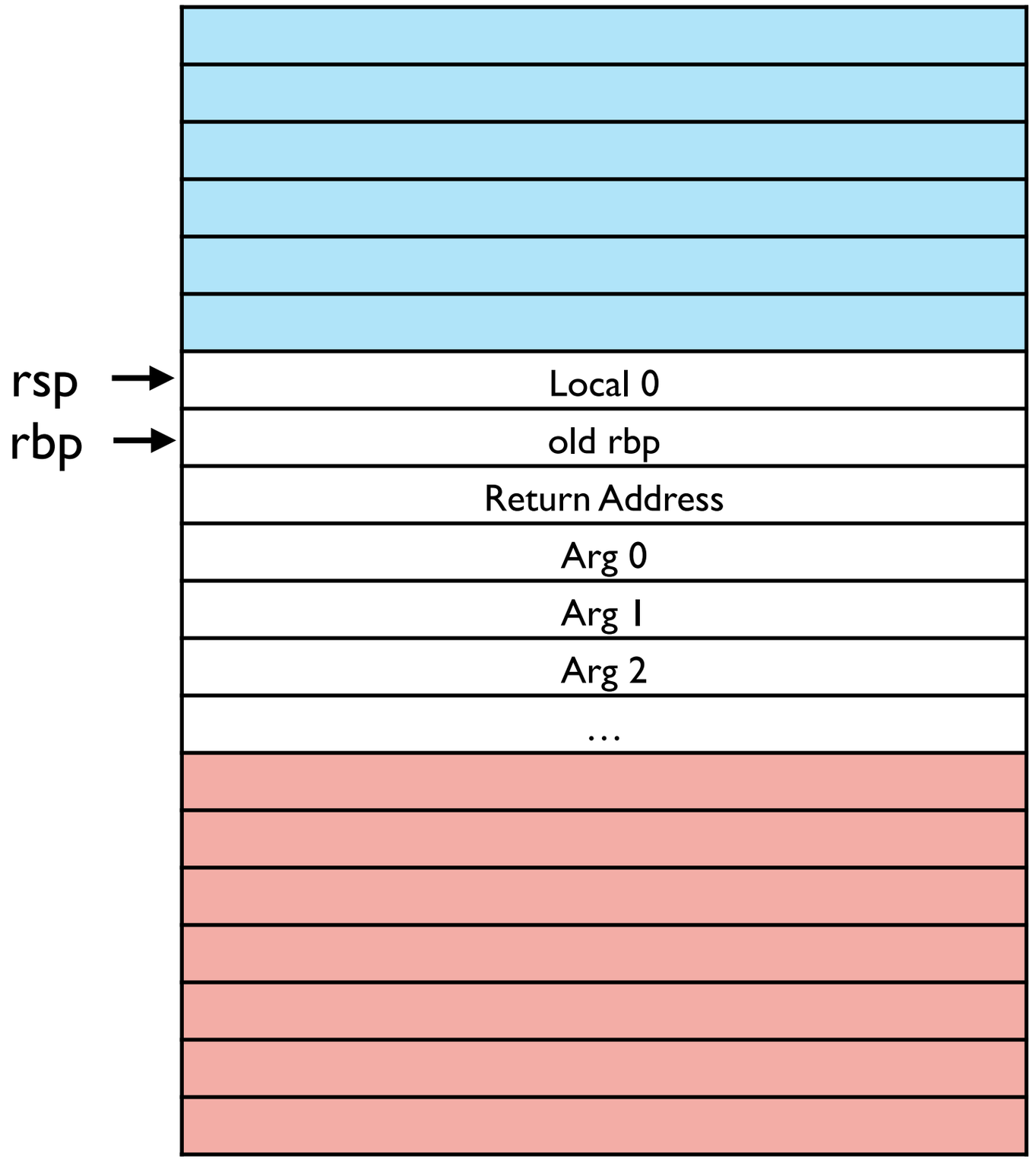
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret


  push arg1
  push arg0
  call fun
  add rsp, 16
```



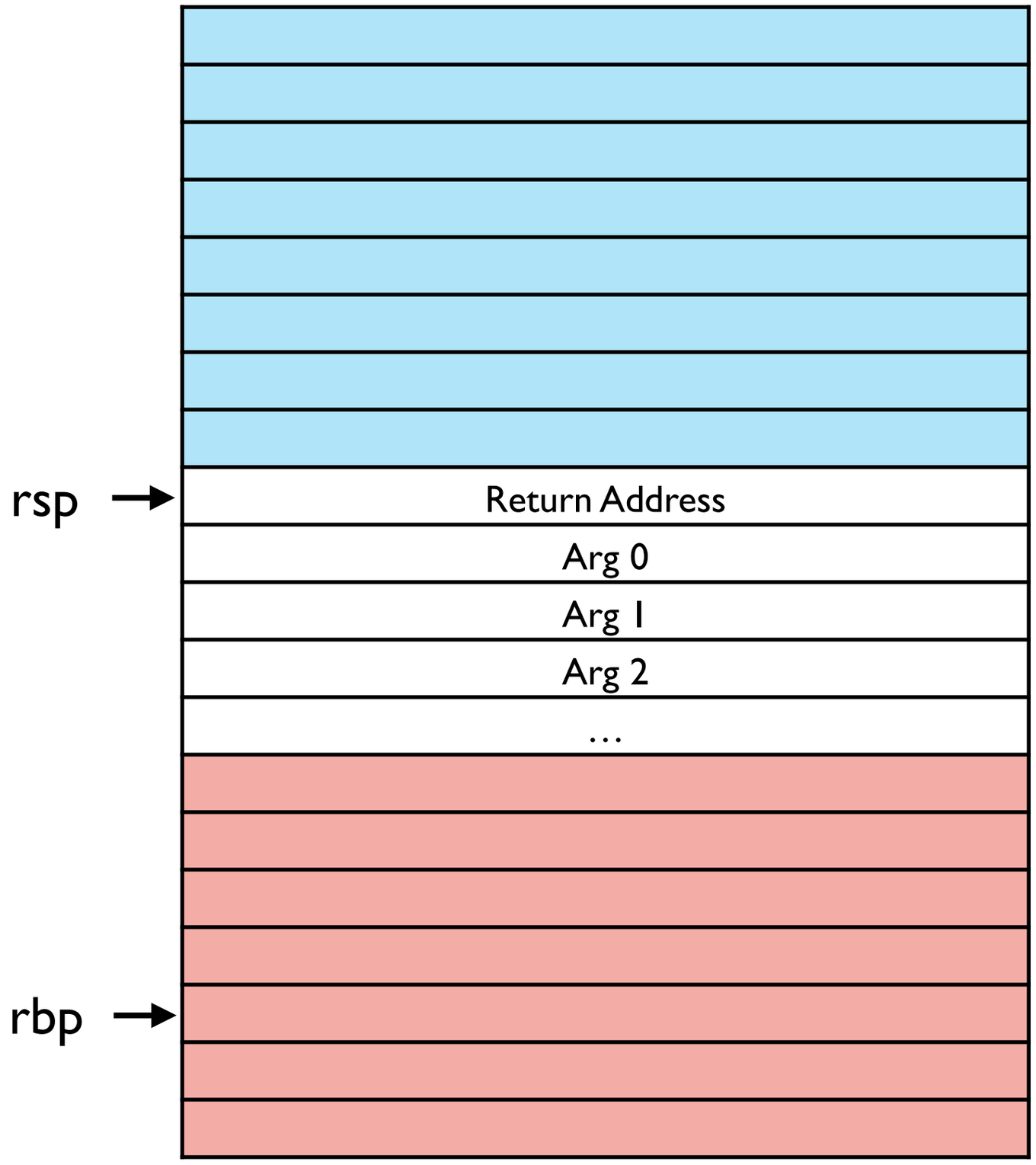
```
fun:
push rbp
mov rbp, rsp
sub rsp, locals * 8
…
mov rsp, rbp
pop rbp
ret
```

```
  push arg1
  push arg0
  call fun
  add rsp, 16
```



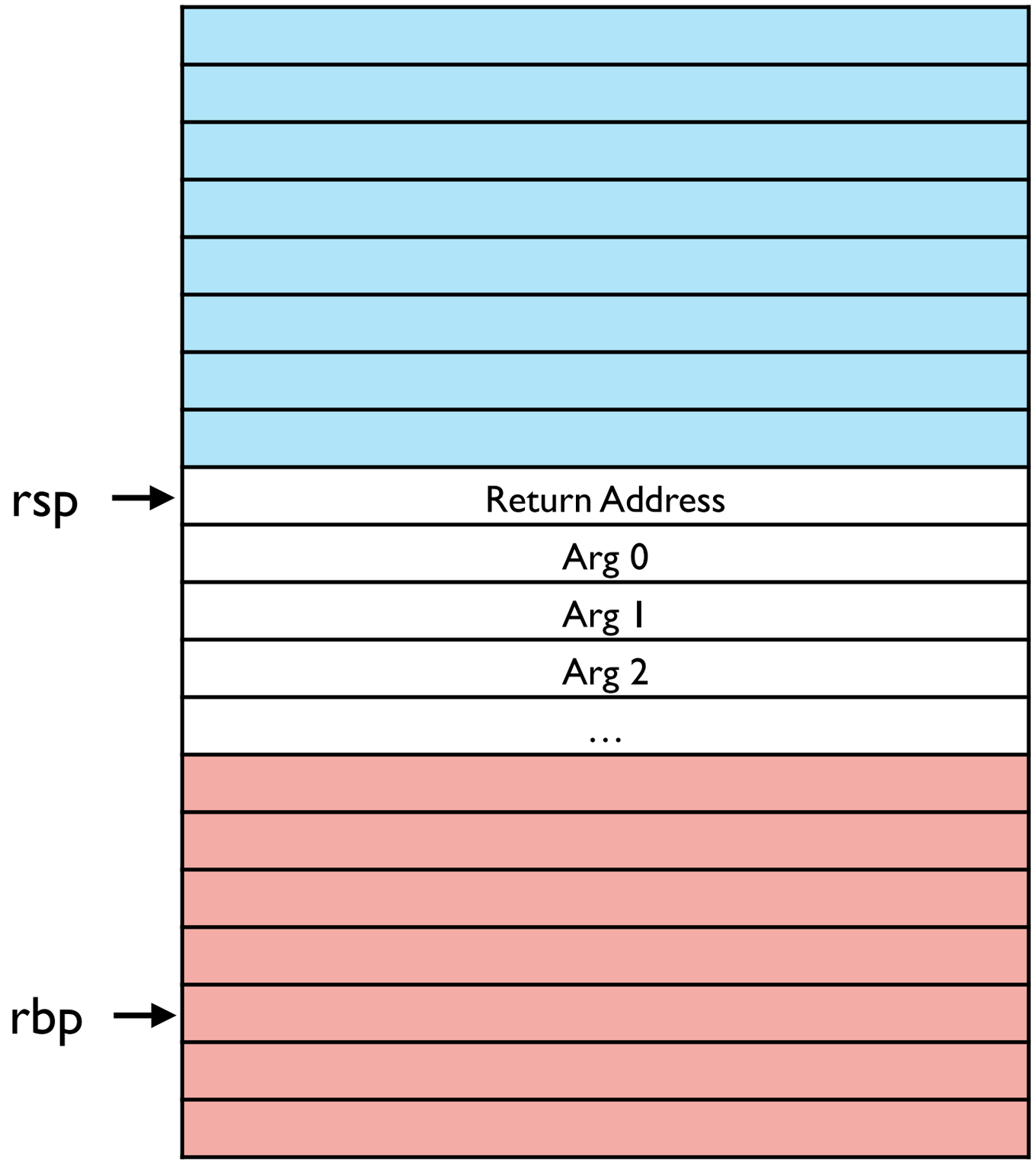
32-bit
C Calling Convention

- rsp points to the return address
- arguments are below (higher addresses)
  return address
- return value goes in rax
- rbp, rbx, r12-r15 are callee-save



64-bit
C Calling Convention

- rsp points to the return address
- ~~arguments are below (higher addresses)~~
  ~~return address~~
- return value goes in rax
- rbp, rbx, r12-r15 are callee-save



64-bit

C Calling Convention

- rsp points to the return address

- **arguments**
  - **rdi, rsi, rdx, rcx, r8, r9**
  - **more on stack (if needed)**

- return value goes in rax
- rbp, rbx, r12-r15 are callee-save

- **before a call, rsp % 16 == 0**

Calling Conventions

- When calling rust code, we need to use the 64 bit calling convention
- When calling "snake" functions, we will use the 32-bit calling convention
  - Makes it easier to test passing arguments on the stack without making huge functions