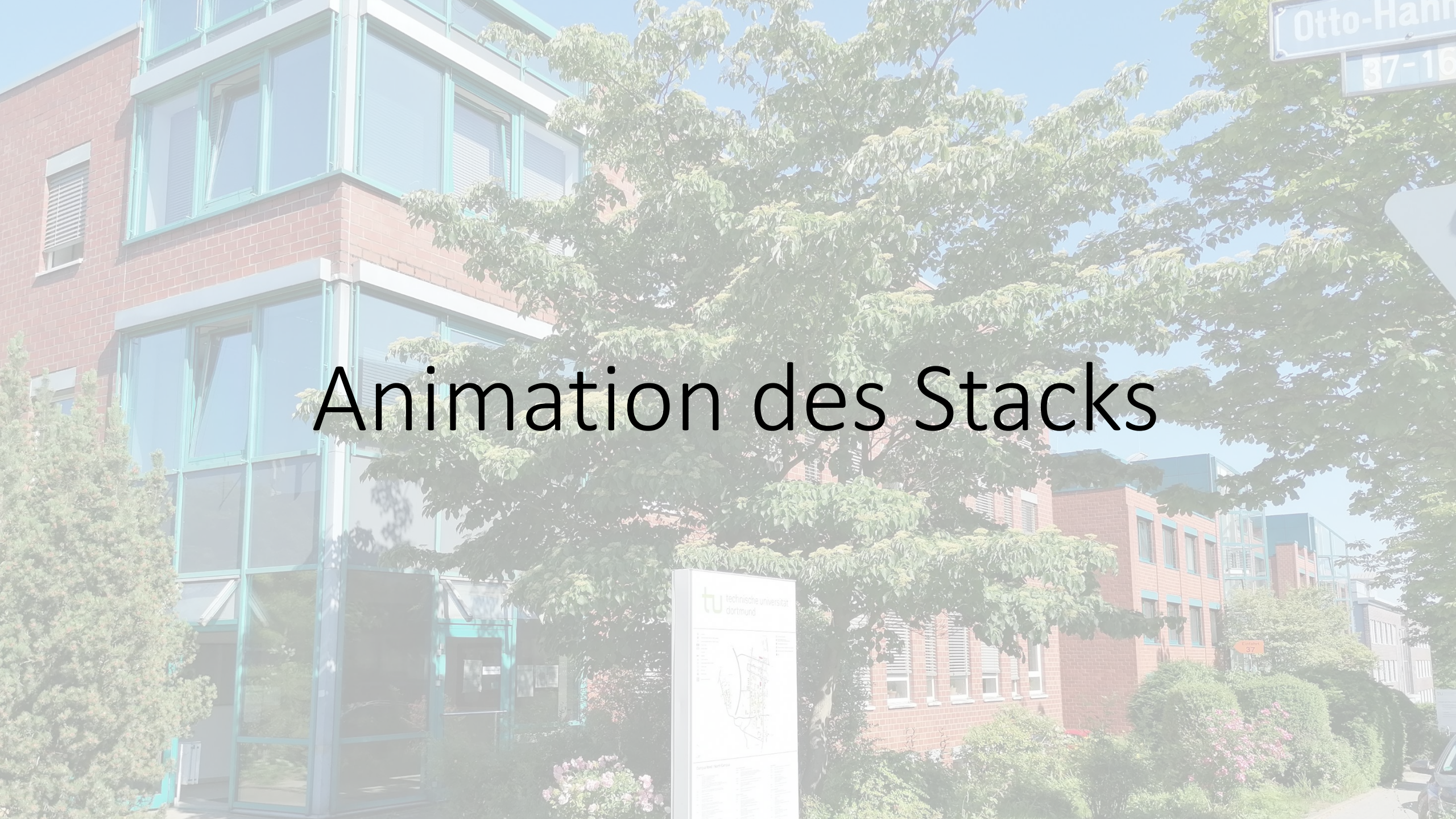
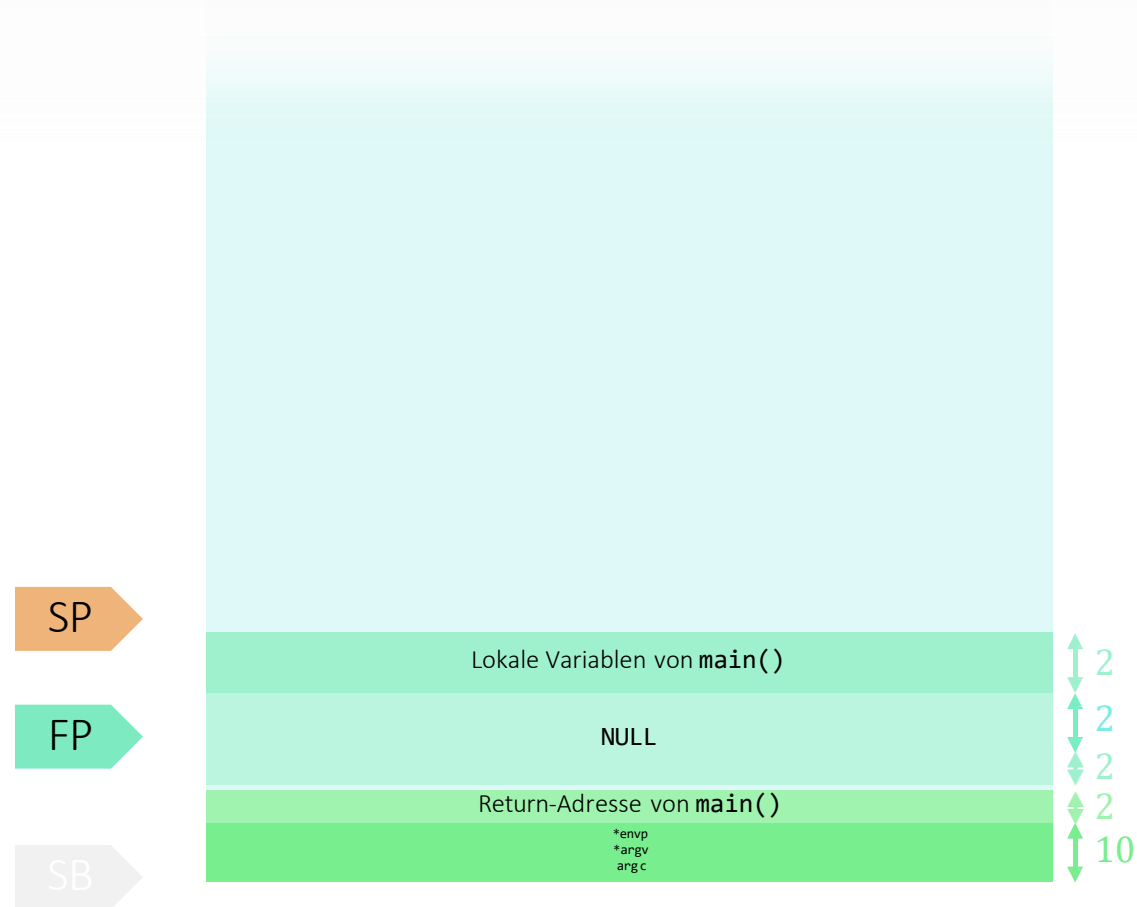


Animation des Stacks

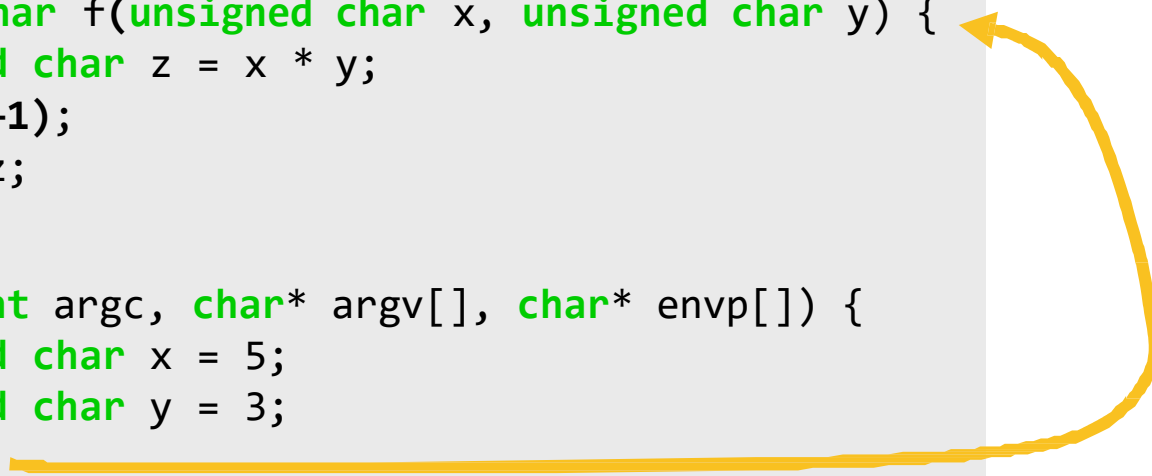


Stack nach der Initialisierung



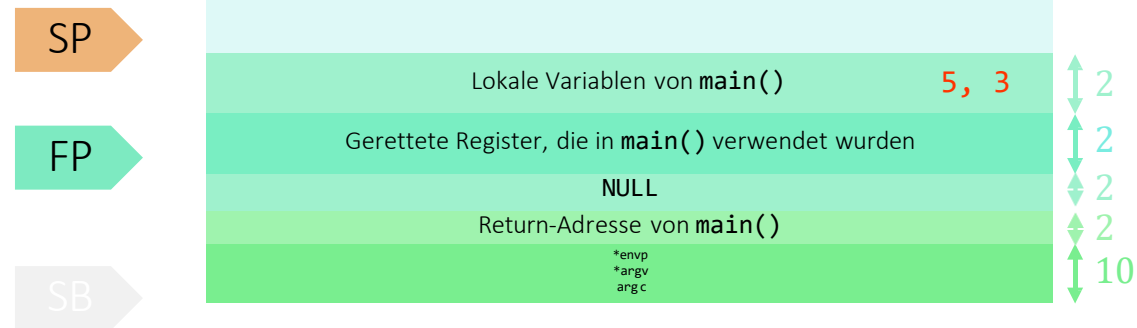
Aufruf einer Funktion

```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
    unsigned char y = 3;  
    f(x, y)  
    return 0;  
}
```

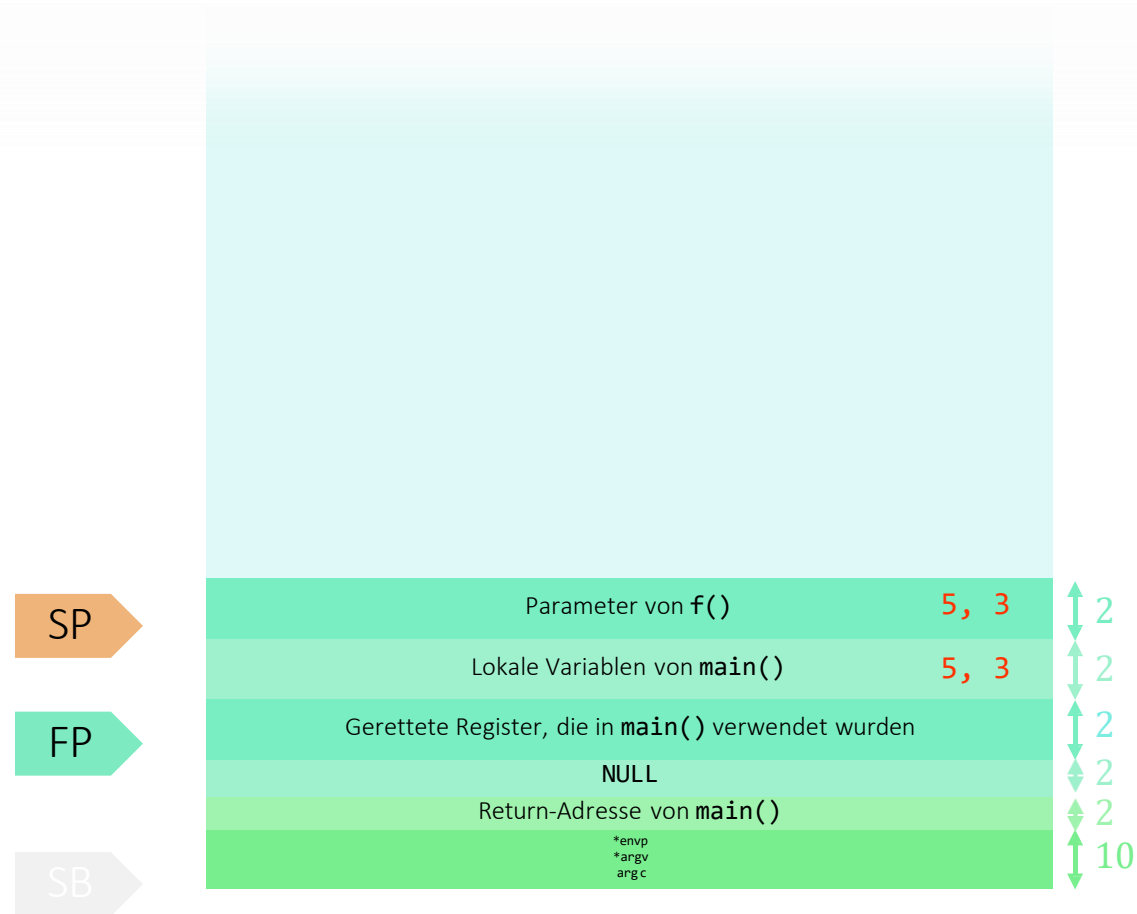


The diagram consists of two yellow arrows. One arrow starts at the function call `f(x, y)` in the `main` function and points to the opening curly brace of the `f` function definition. The second arrow starts at the call `g(z+1)` inside the `f` function and points to the opening curly brace of the `g` function definition, illustrating the sequence of function calls.

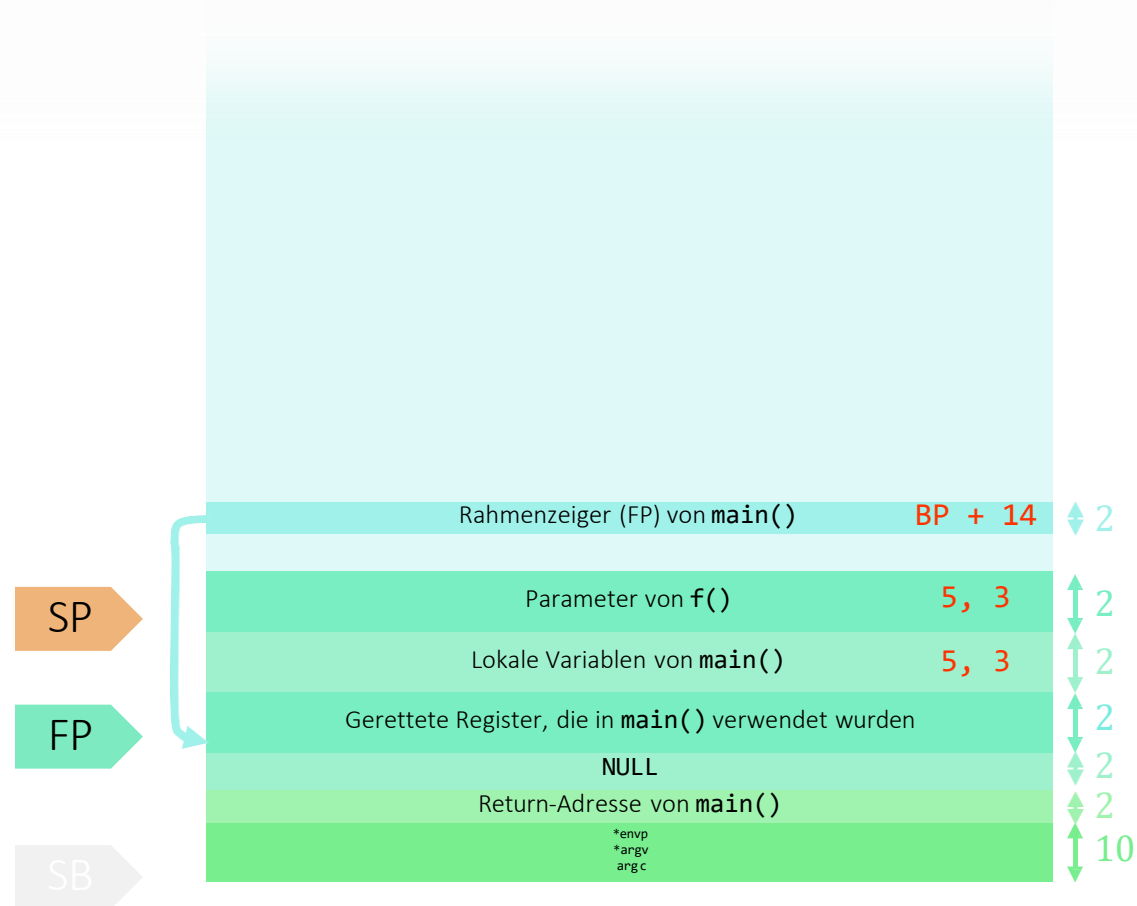
Register retten



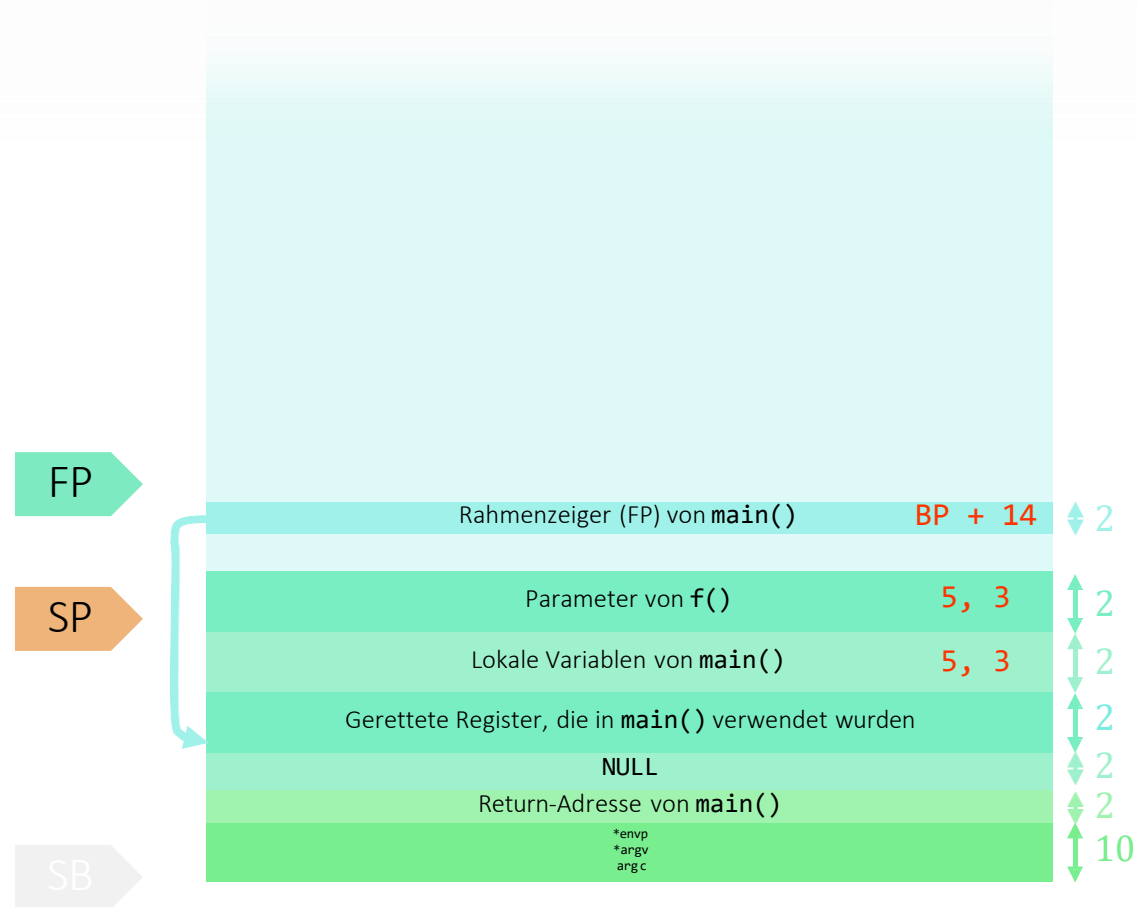
Parameter auf den Stapel legen



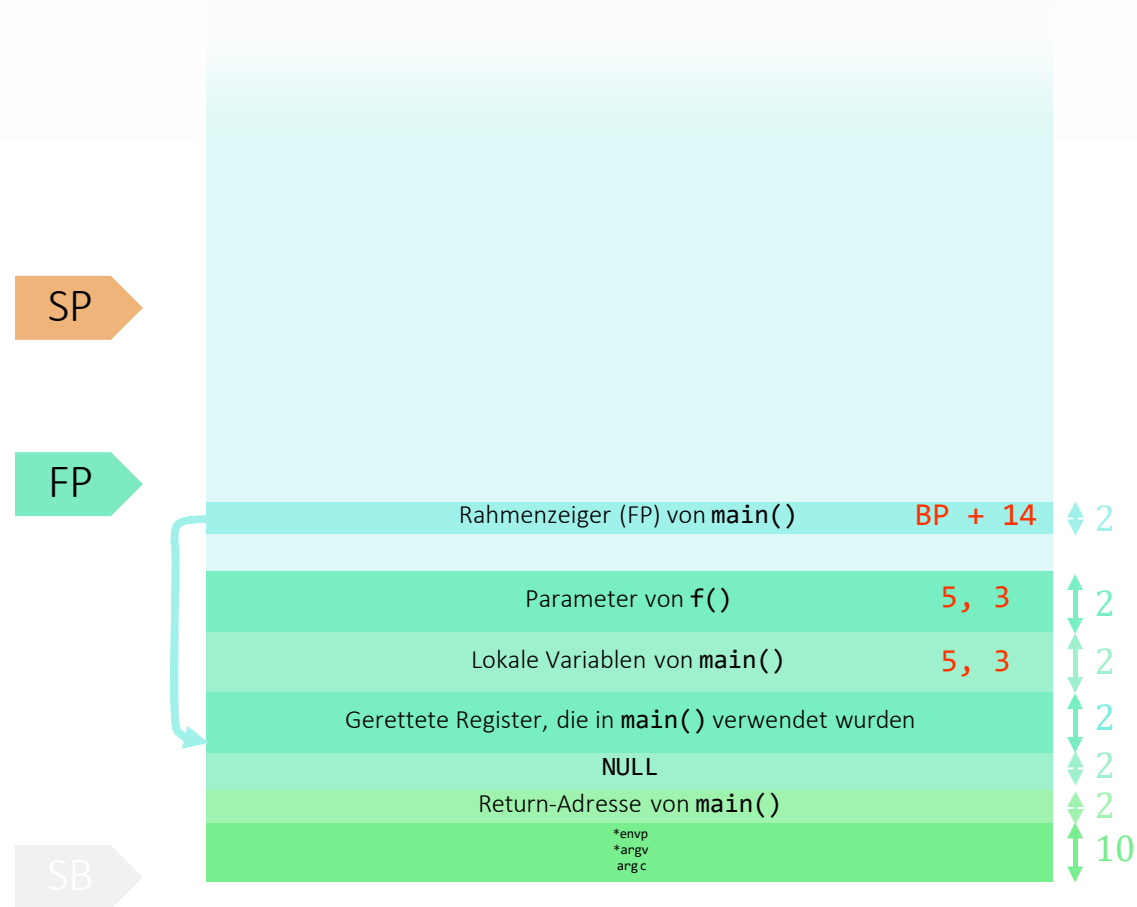
Rahmenzeiger retten



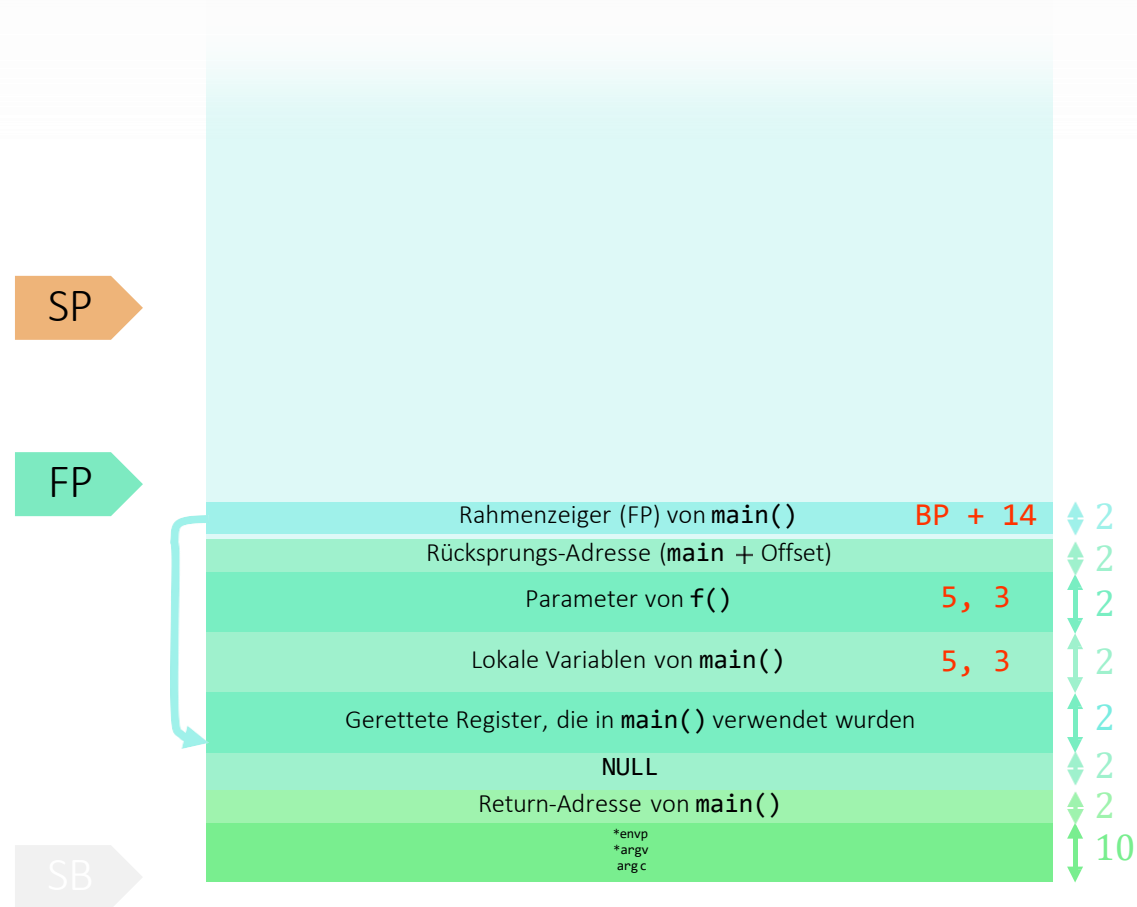
Rahmenzeiger verschieben



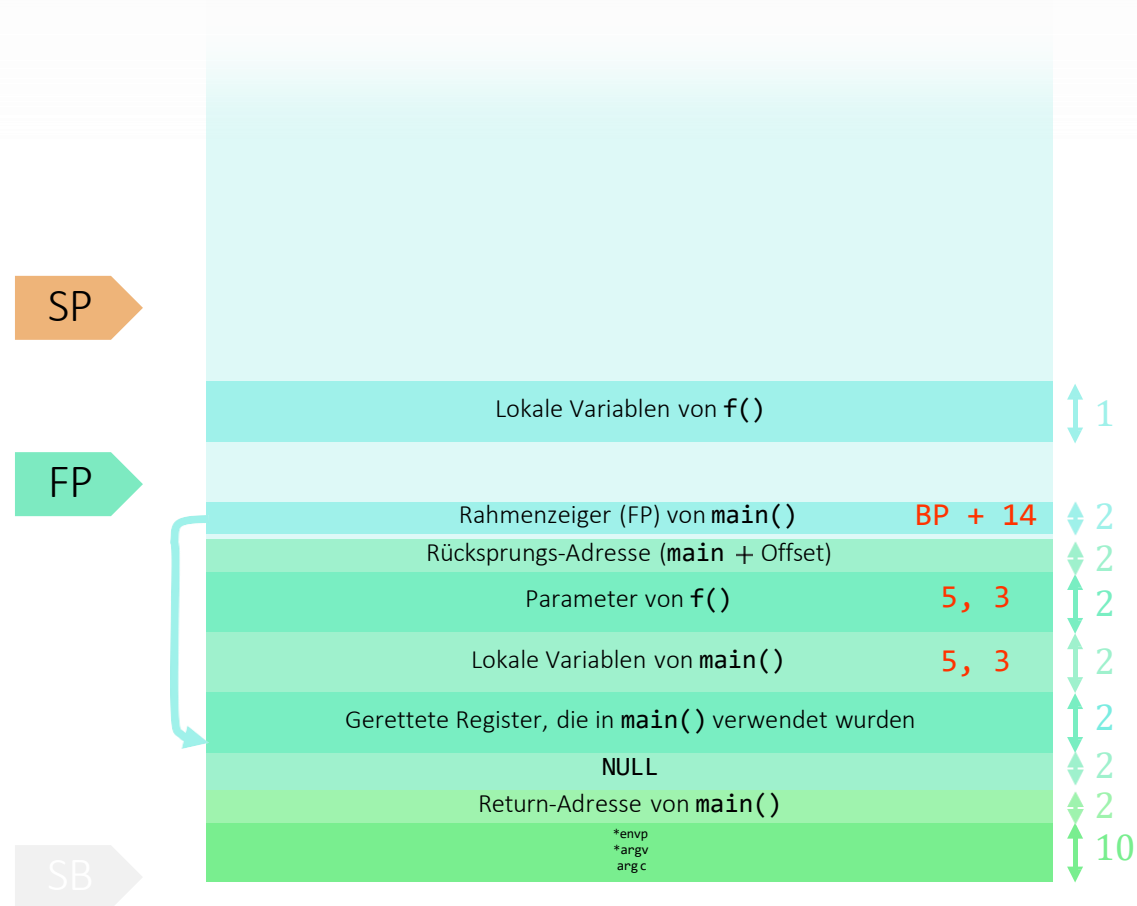
Stapelzeiger verschieben



Rücksprungsadresse speichern



Unterprogrammaufruf

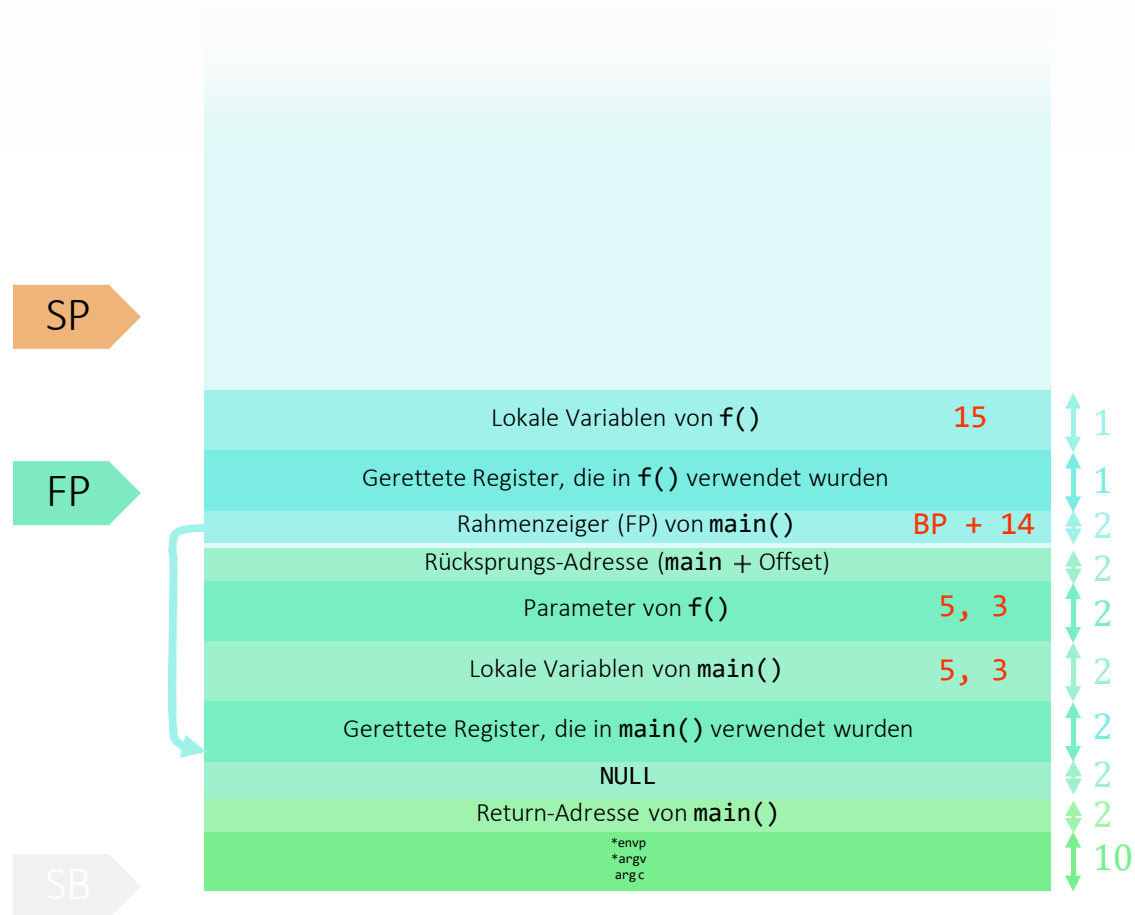


Aufruf einer weiteren Funktion

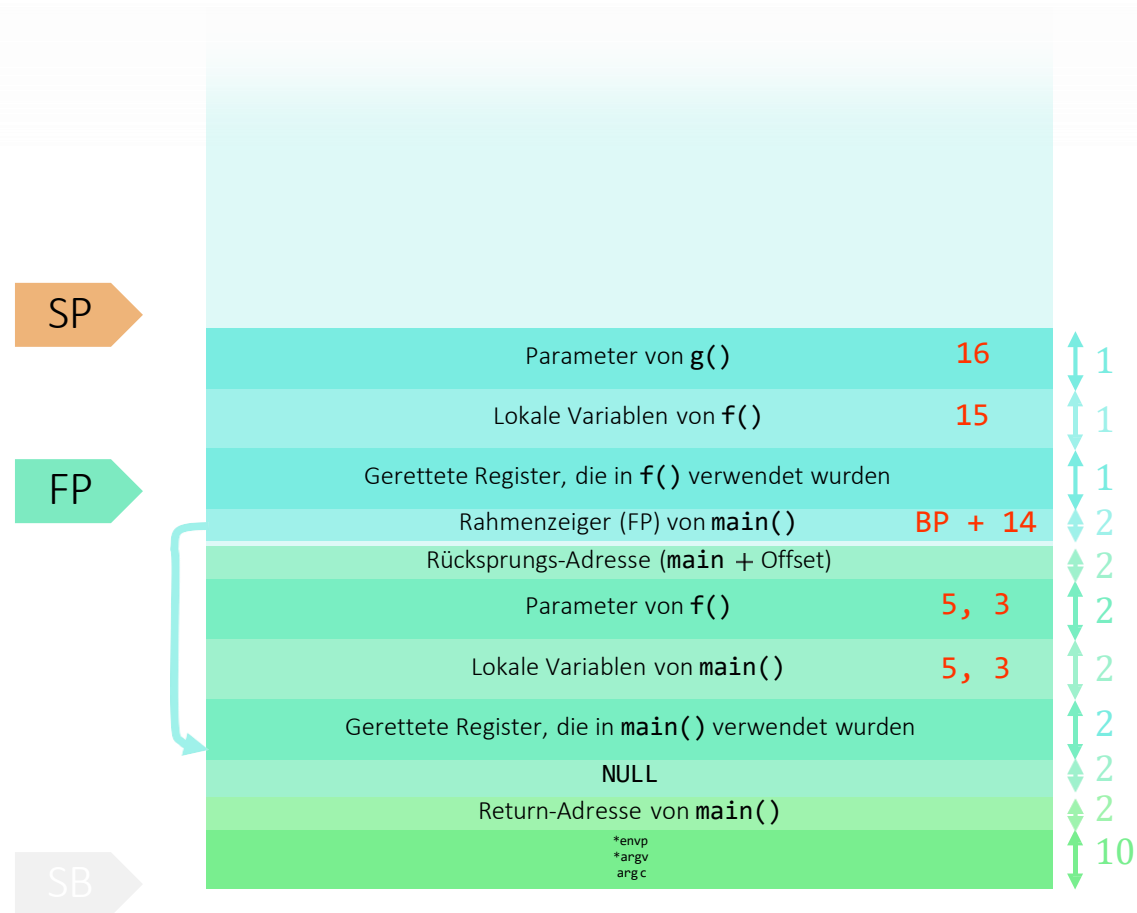
```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
    unsigned char y = 3;  
    f(x, y)  
    return 0;  
}
```

The diagram illustrates the sequence of function calls. A green arrow originates from the `f(x, y)` call in the `main` function and points to the start of the `f` function definition. A yellow arrow originates from the `g(z+1)` call within the `f` function and points to the start of the `g` function definition. This visualizes the call stack where `main` calls `f`, and `f` in turn calls `g`.

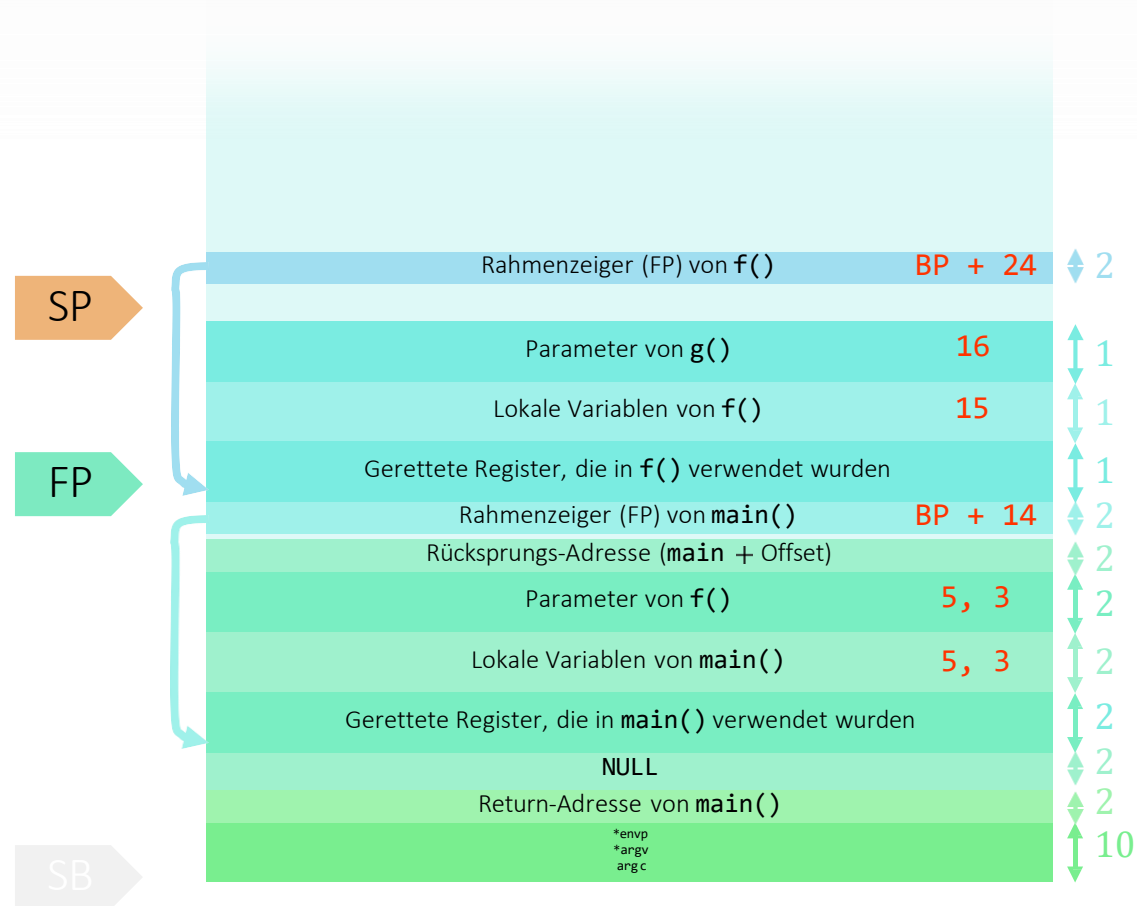
Register retten



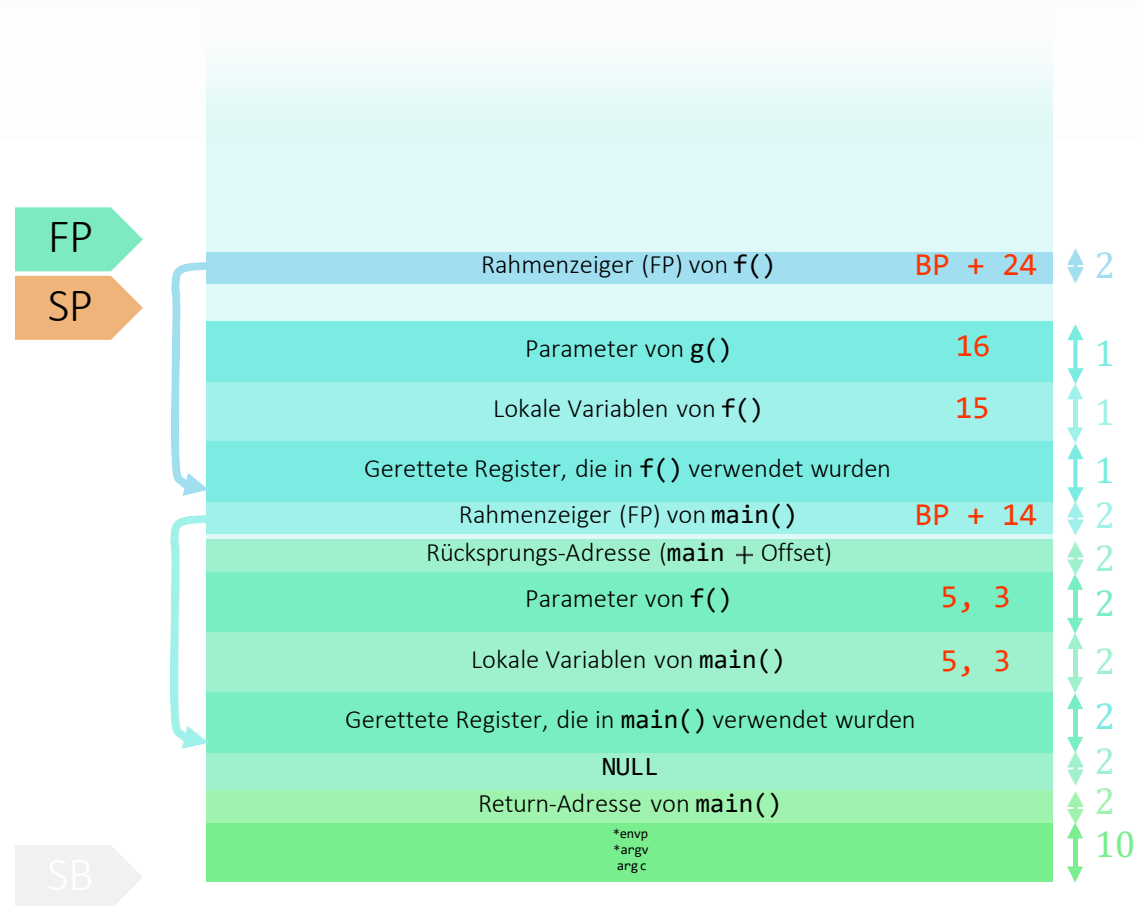
Parameter auf den Stapel legen



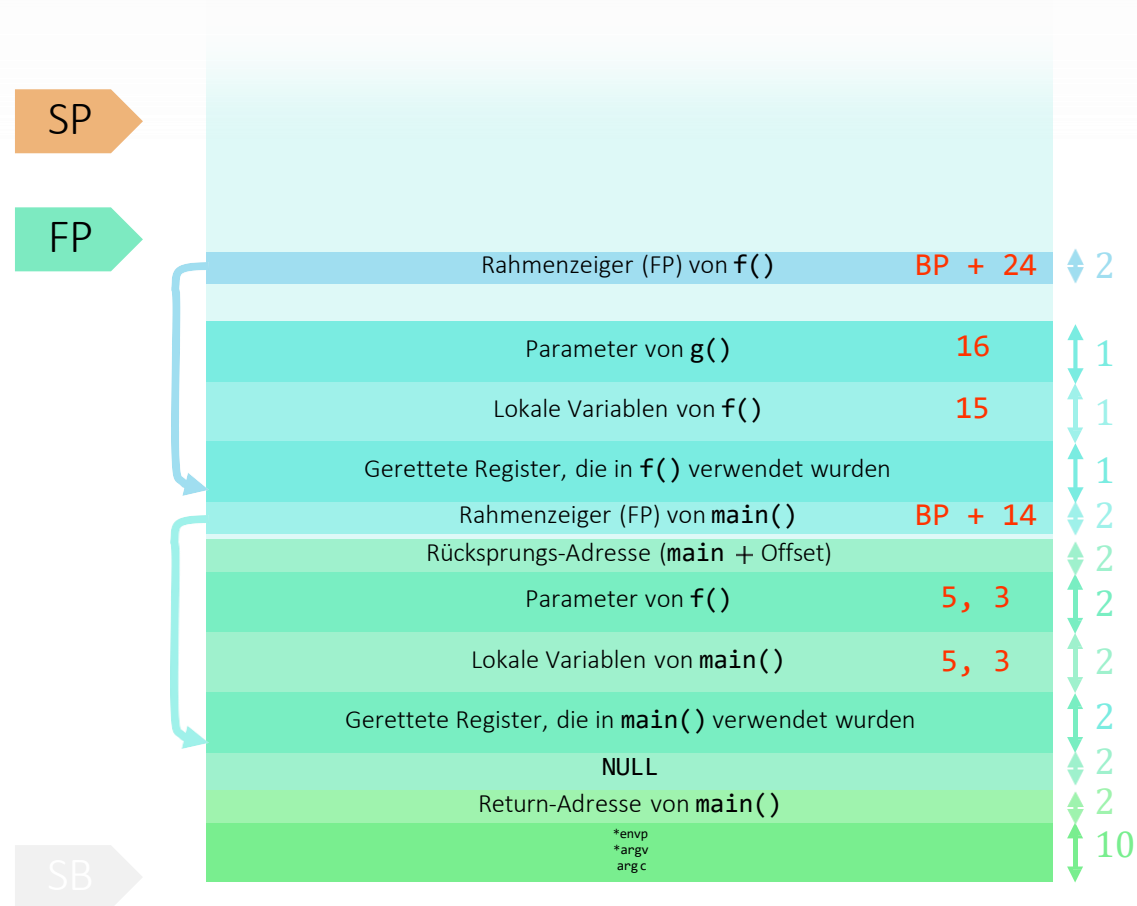
Rahmenzeiger retten



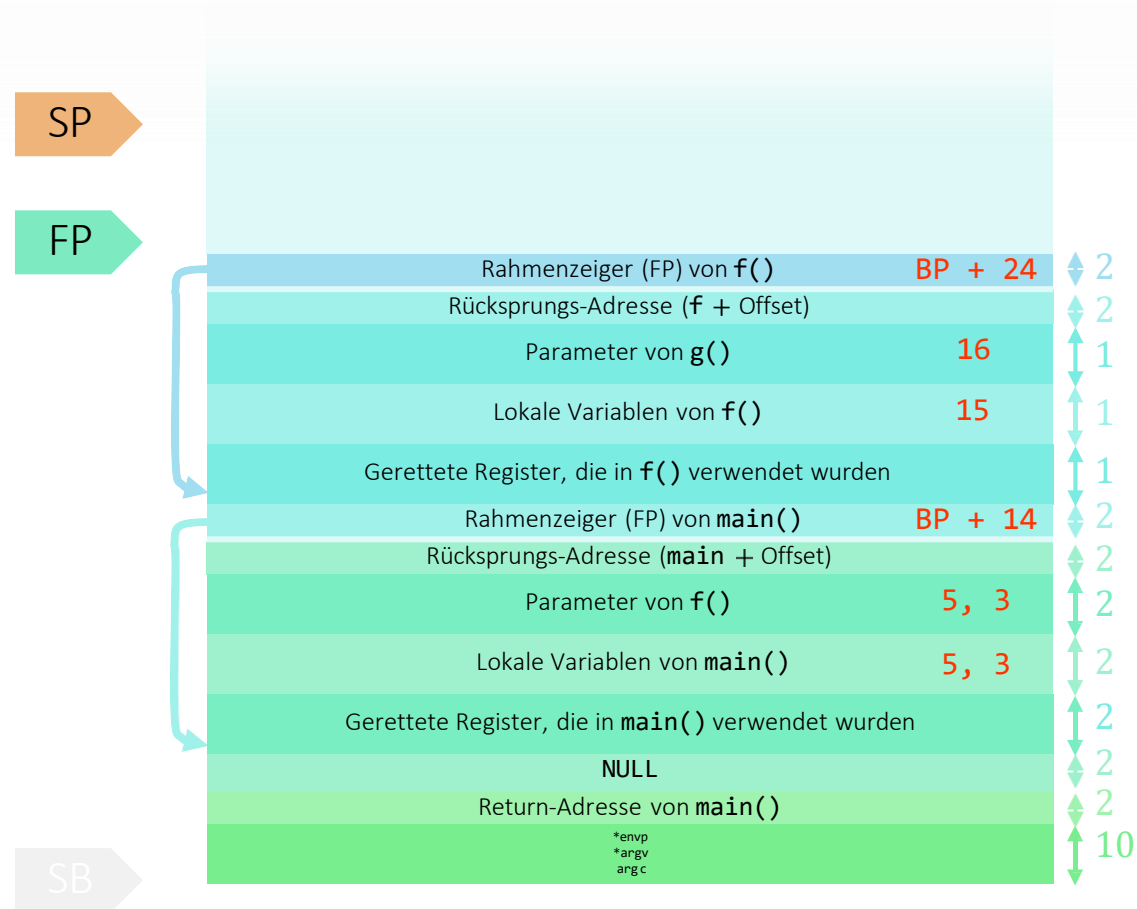
Rahmenzeiger verschieben



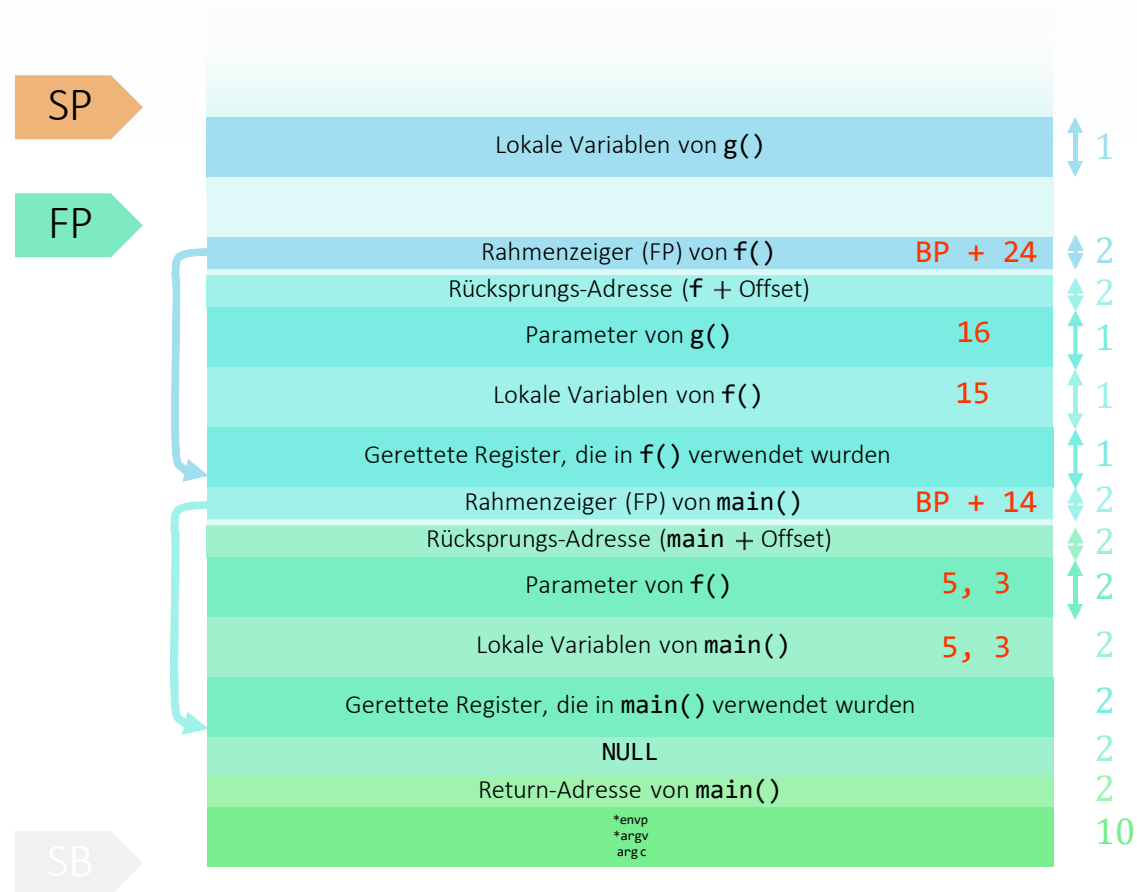
Stapelzeiger verschieben



Rücksprungadresse speichern



Unterprogrammaufruf

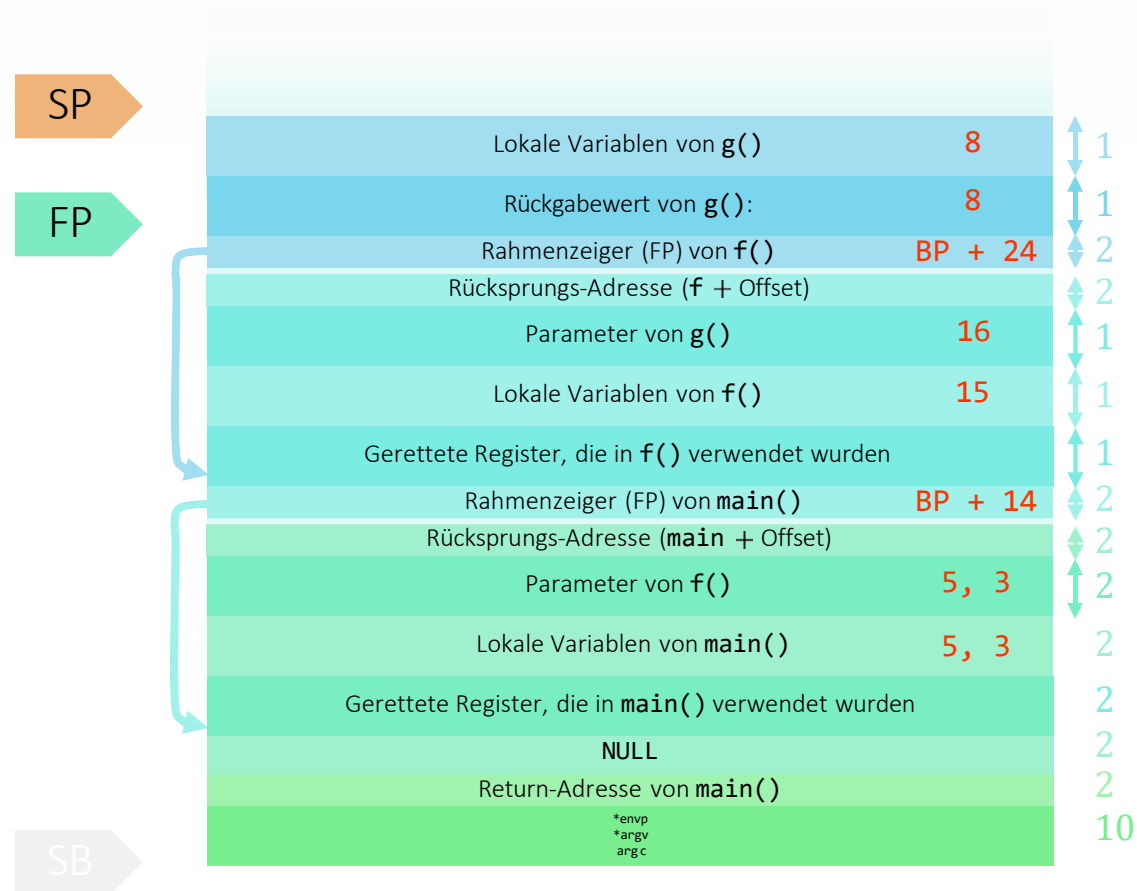


Aufruf der Return-Anweisung

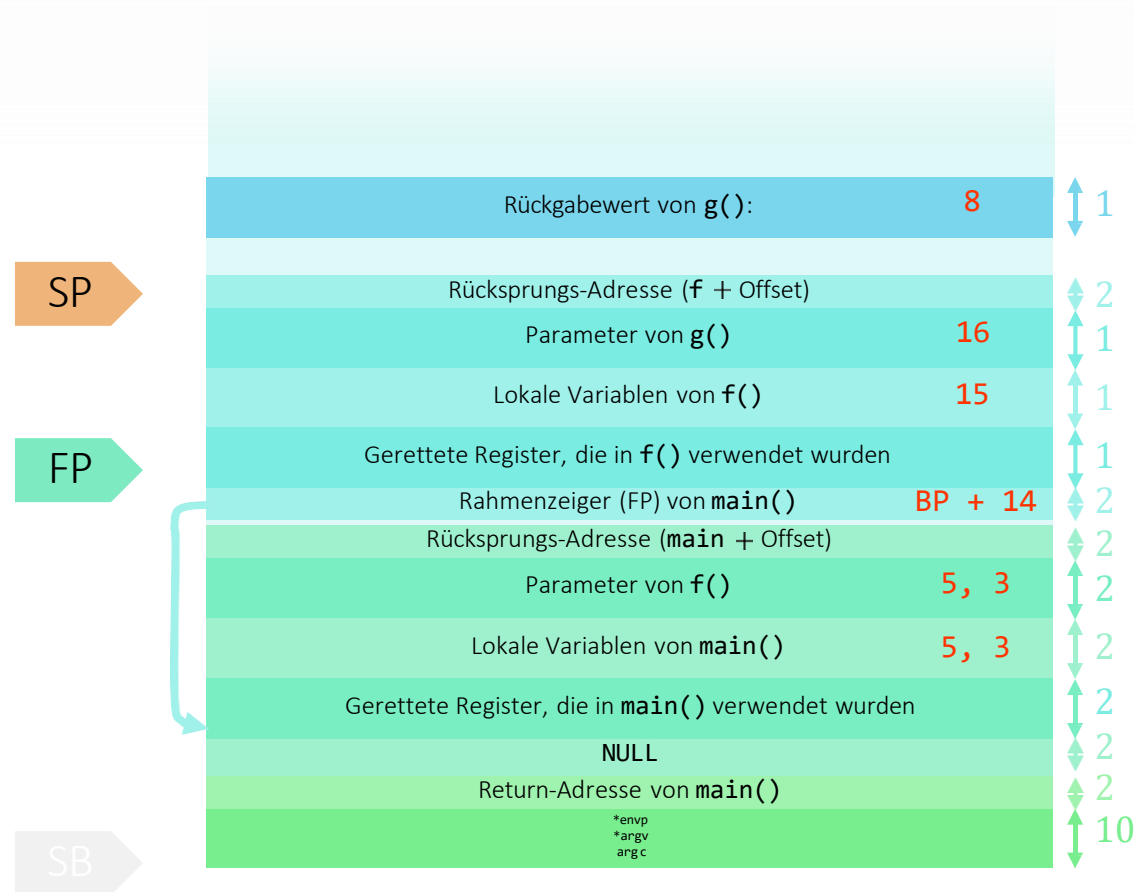
```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
    unsigned char y = 3;  
    f(x, y)  
    return 0;  
}
```

The diagram illustrates the flow of return values between the functions. Green arrows show the return path from the function body back to the caller: from `return t;` in `g` to the `z = g(z+1);` line in `f`, and from `f(x, y)` in `main` to the `return 0;` line. Orange arrows show the return path from the function body back to the caller: from `return z;` in `f` to the `return 0;` line in `main`, and from `return t;` in `g` to the `return z;` line in `f`.

Erzeugen des Rückgabewertes



Stapel- und Rahmenzeiger wiederherstellen

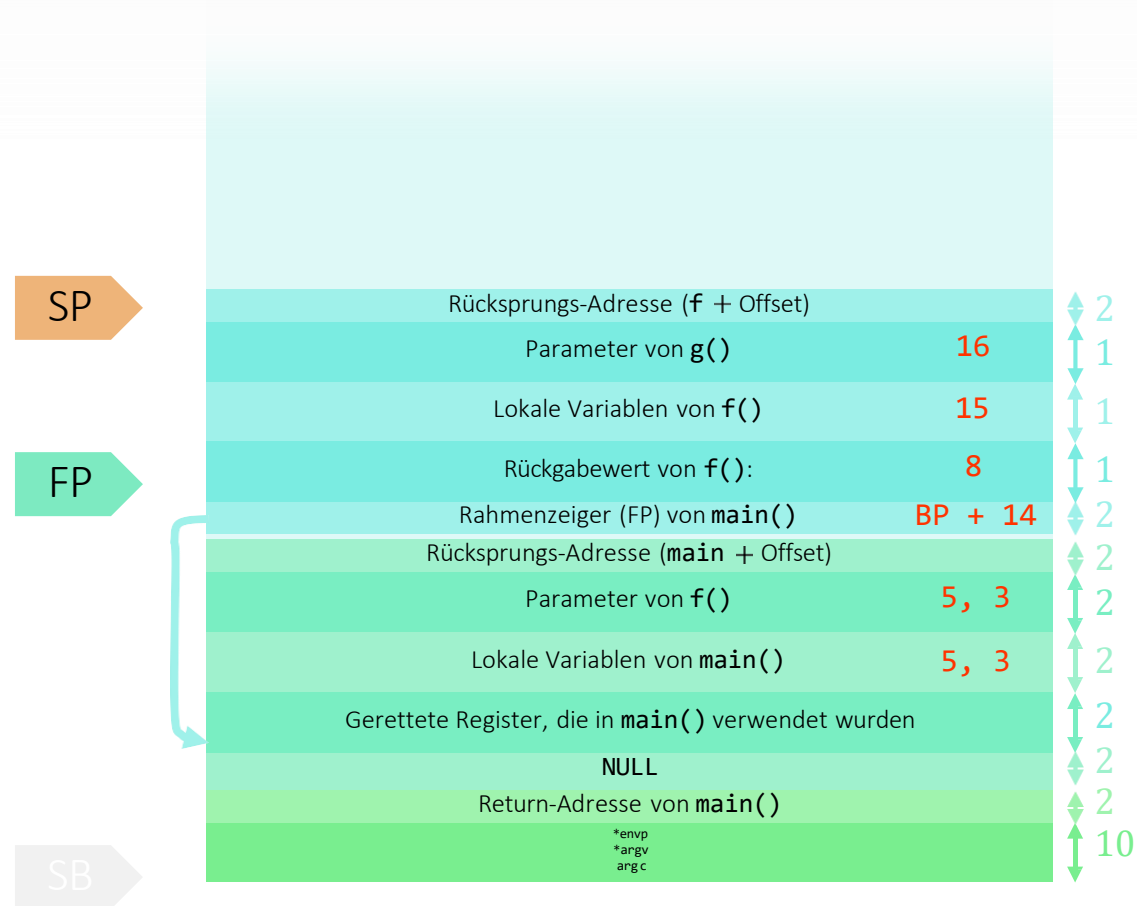


Rückkehr in die main

```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
    unsigned char y = 3;  
    f(x, y)  
    return 0;  
}
```

```
graph TD  
    g_return[return t] -- green --> f_call[f(z+1)]  
    f_return[return z] -- green --> main_call[f(x, y)]  
    main_return[return 0] -- orange --> exit(( ))
```

Erzeugen des Rückgabewertes



Stapel- und Rahmenzeiger wiederherstellen

