

Stack am Beispiel erklärt



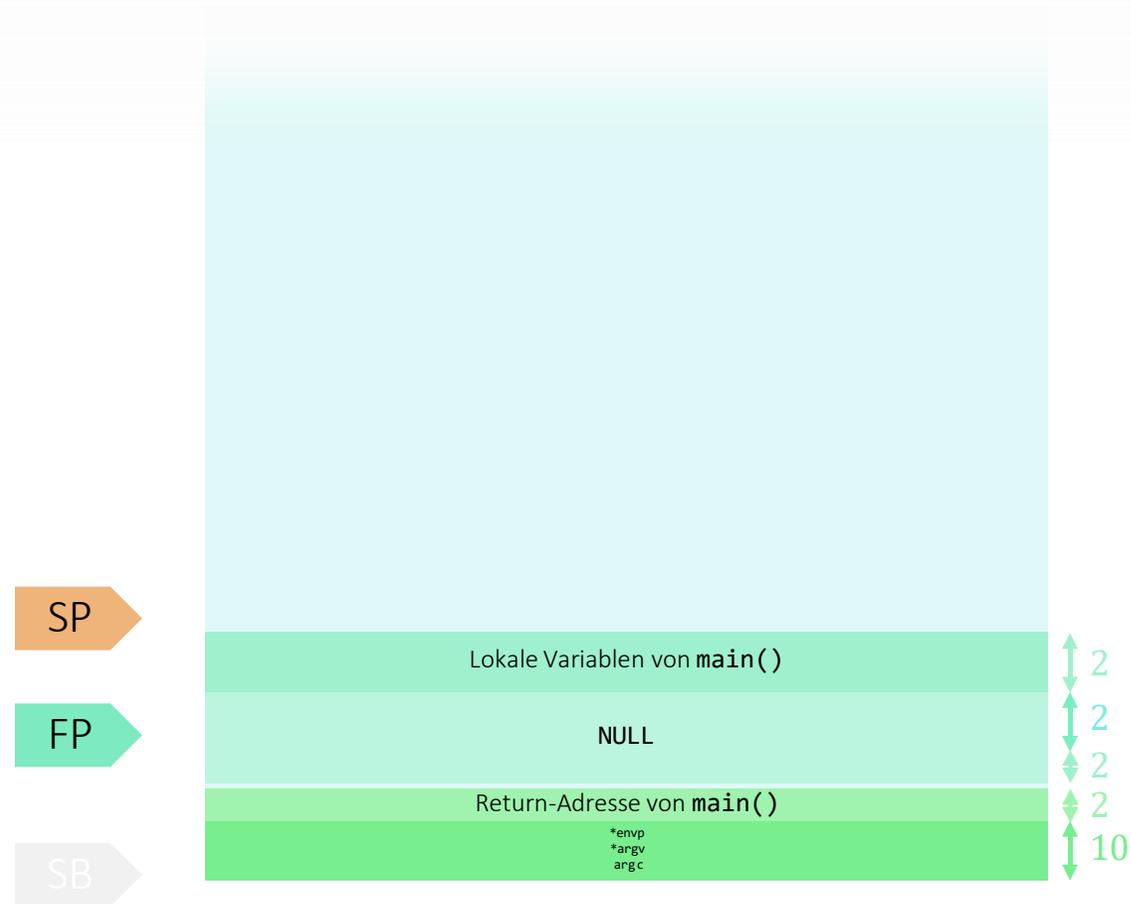
Stack am Beispiel erklärt

```
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;
}

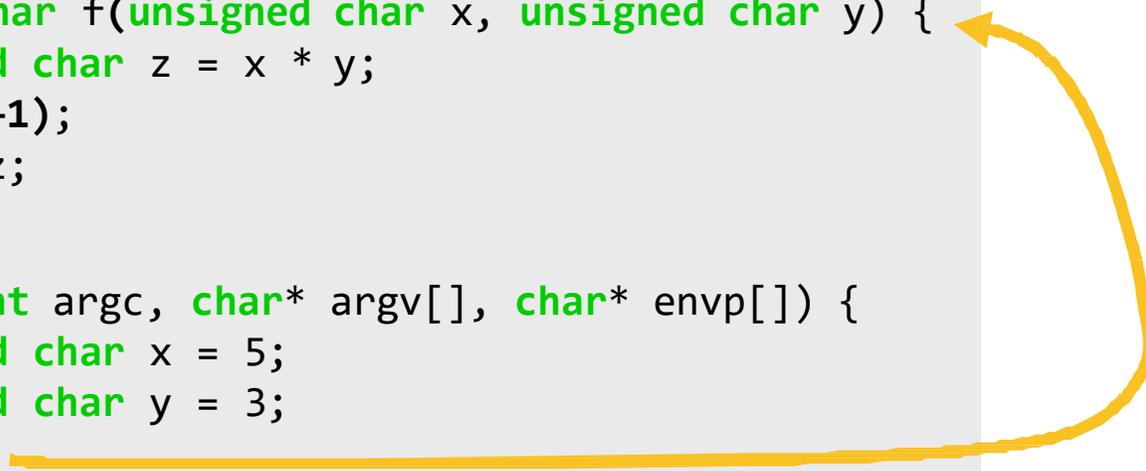
int main(int argc, char* argv[], char* envp[]) {
    unsigned char x = 5;
    unsigned char y = 3;
    f(x, y)
    return 0;
}
```

Stack nach der Initialisierung

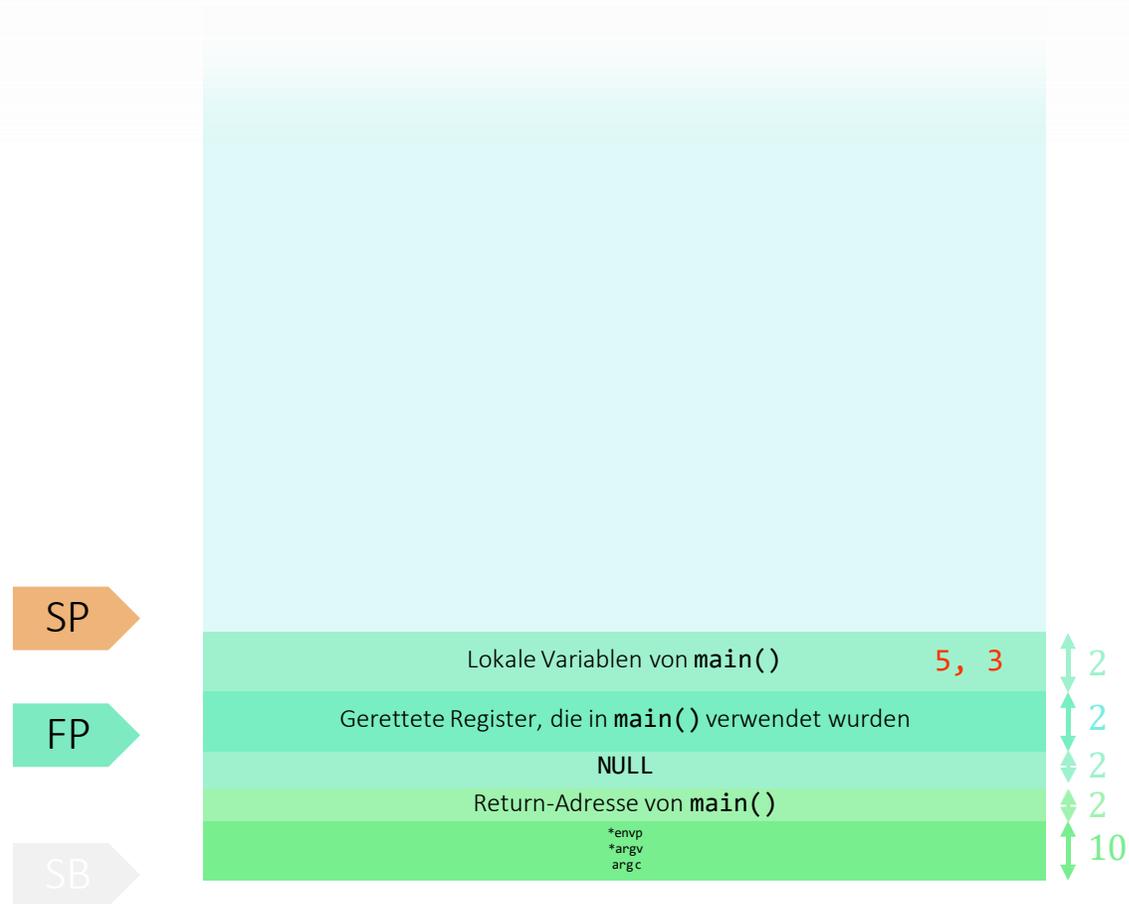


Aufruf einer Funktion

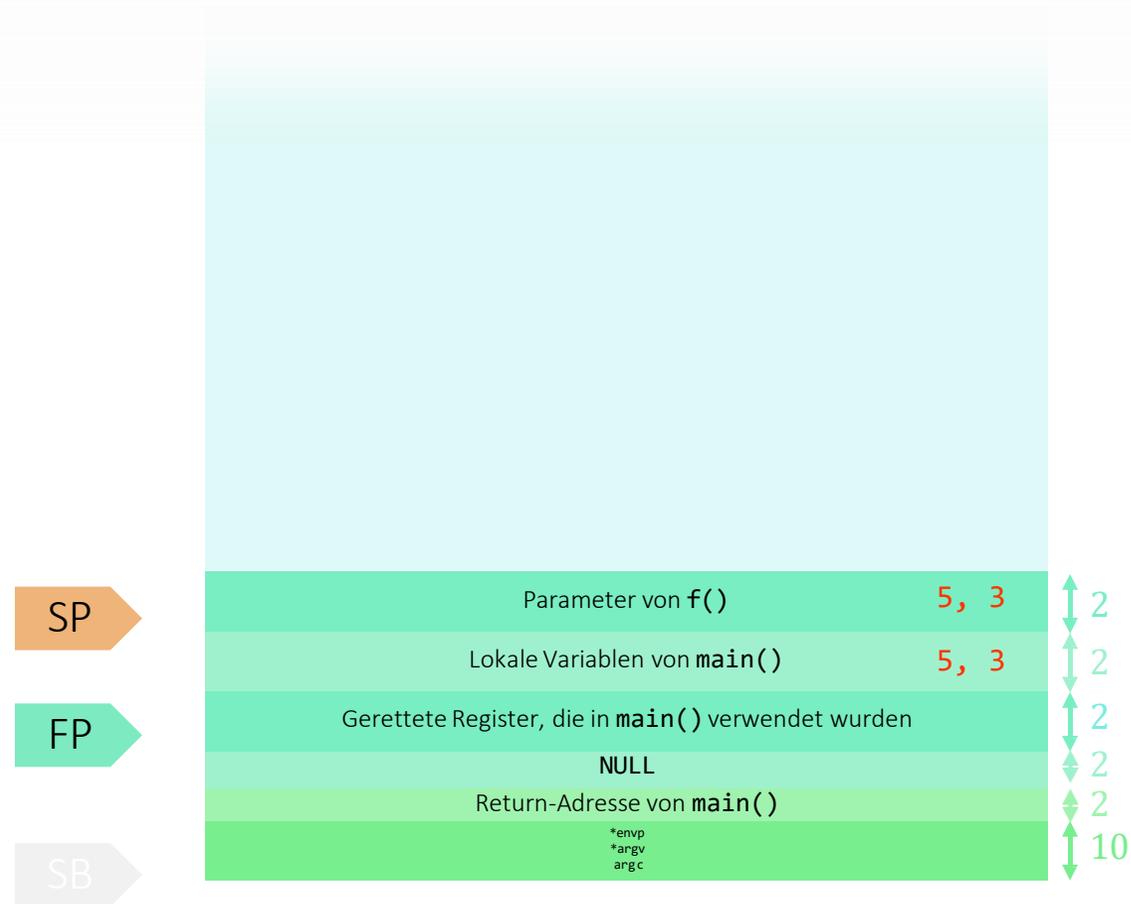
```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
 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



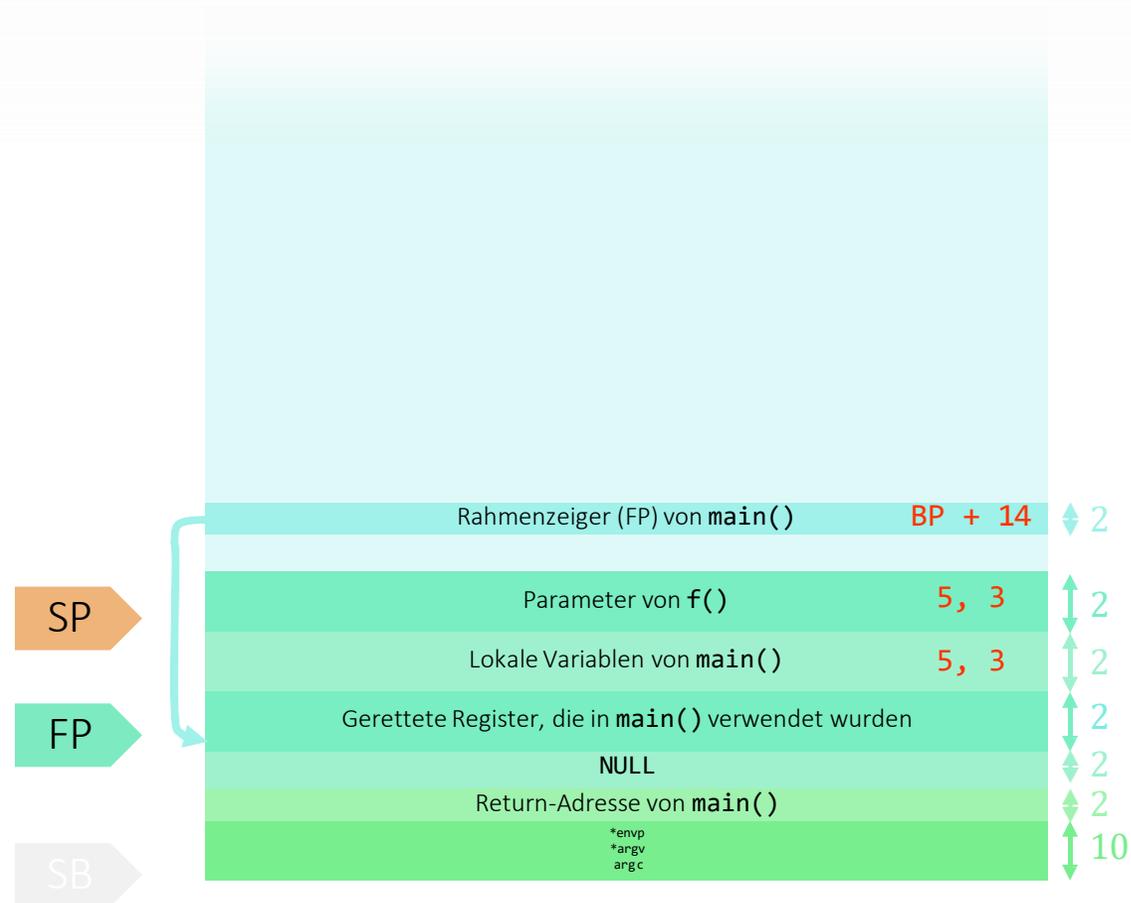
Register retten



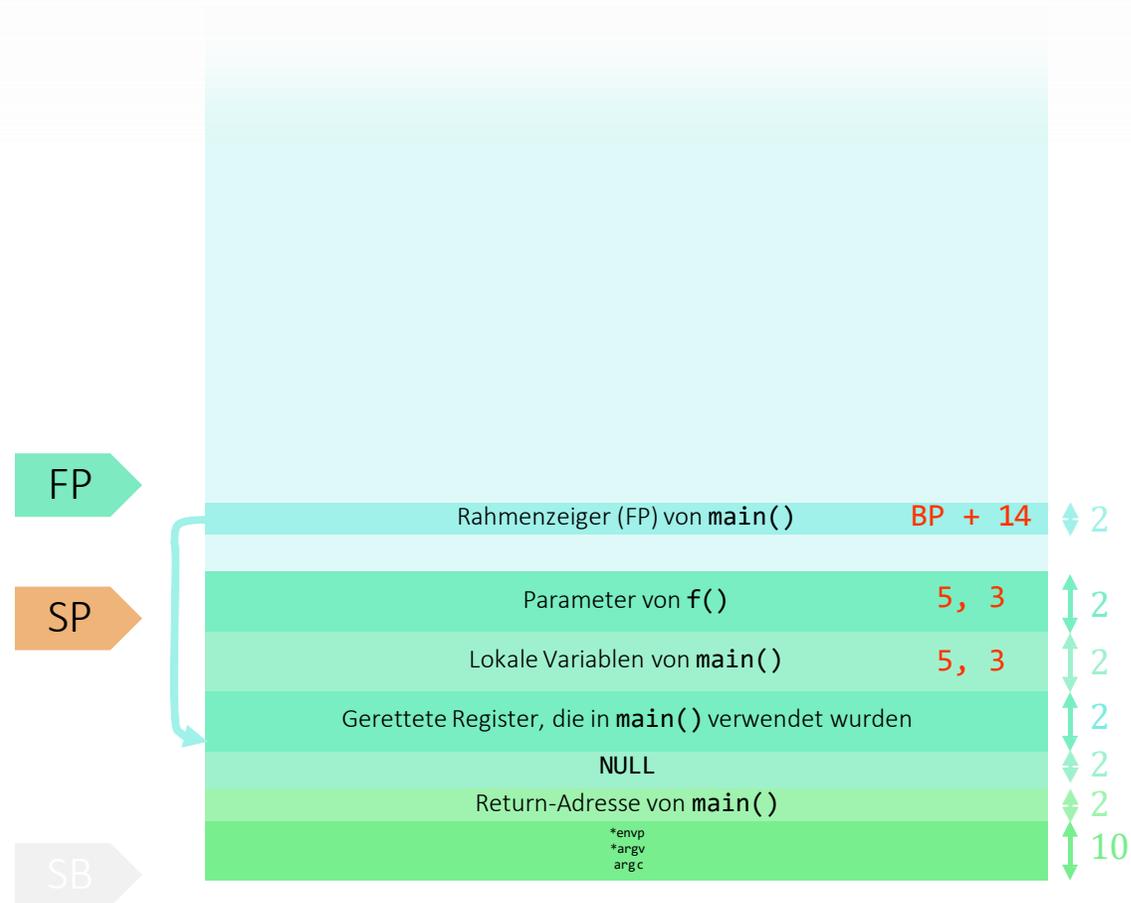
Parameter auf den Stapel legen



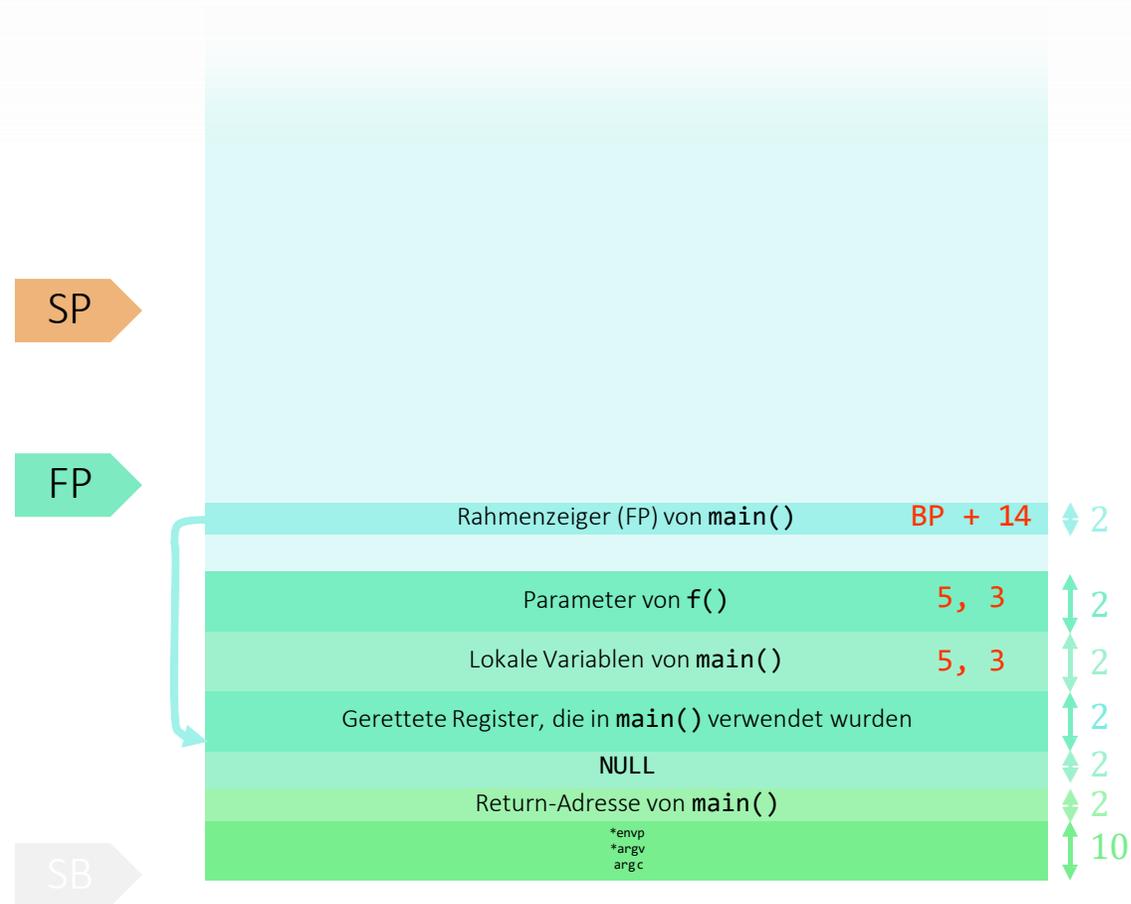
Rahmenzeiger retten



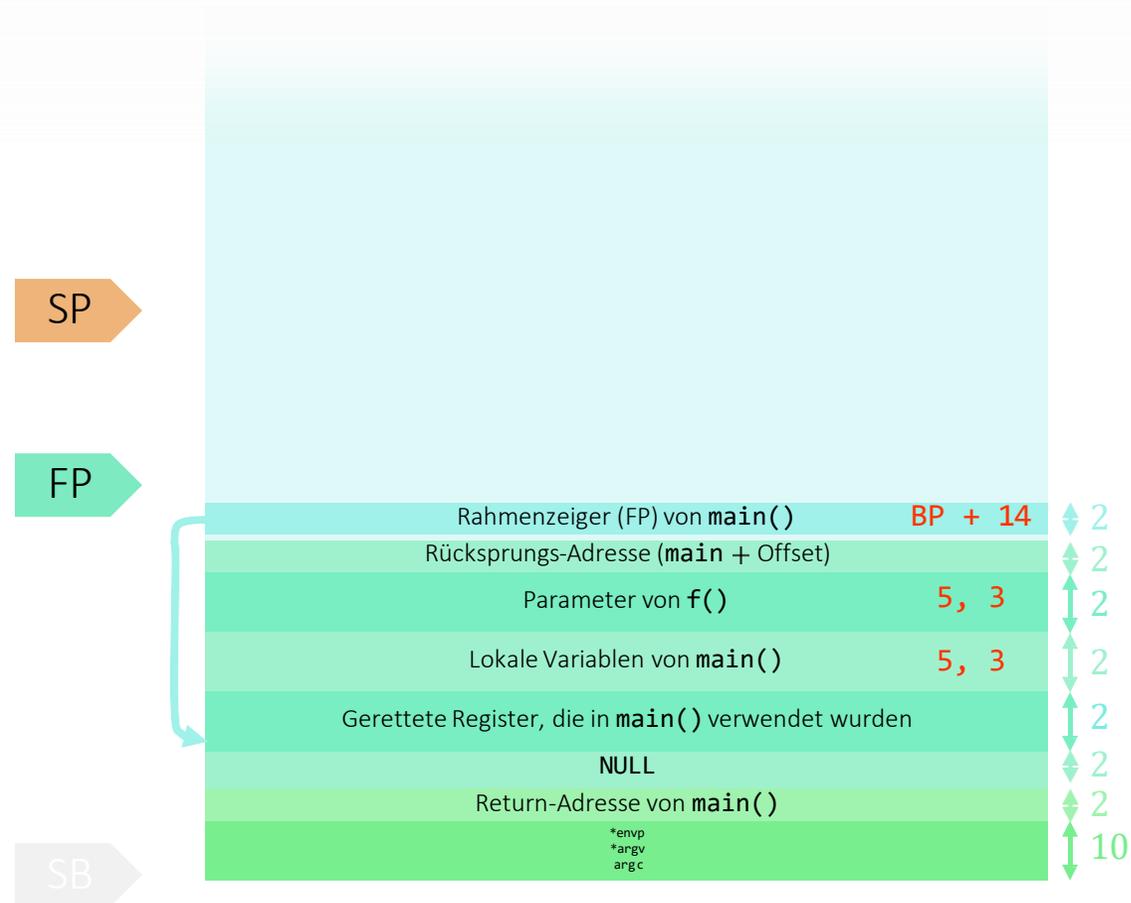
Rahmenzeiger verschieben



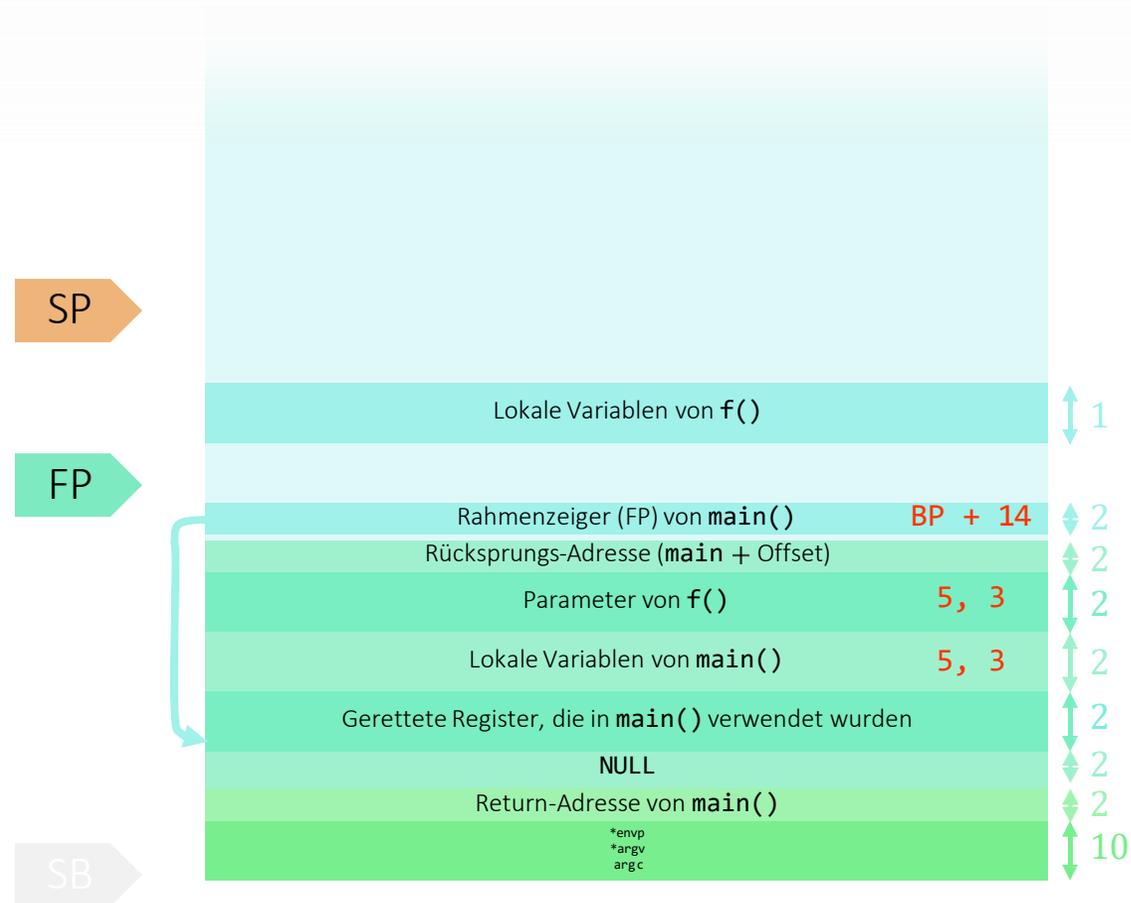
Stapelzeiger verschieben



Rücksprungsadresse speichern



Unterprogrammaufruf

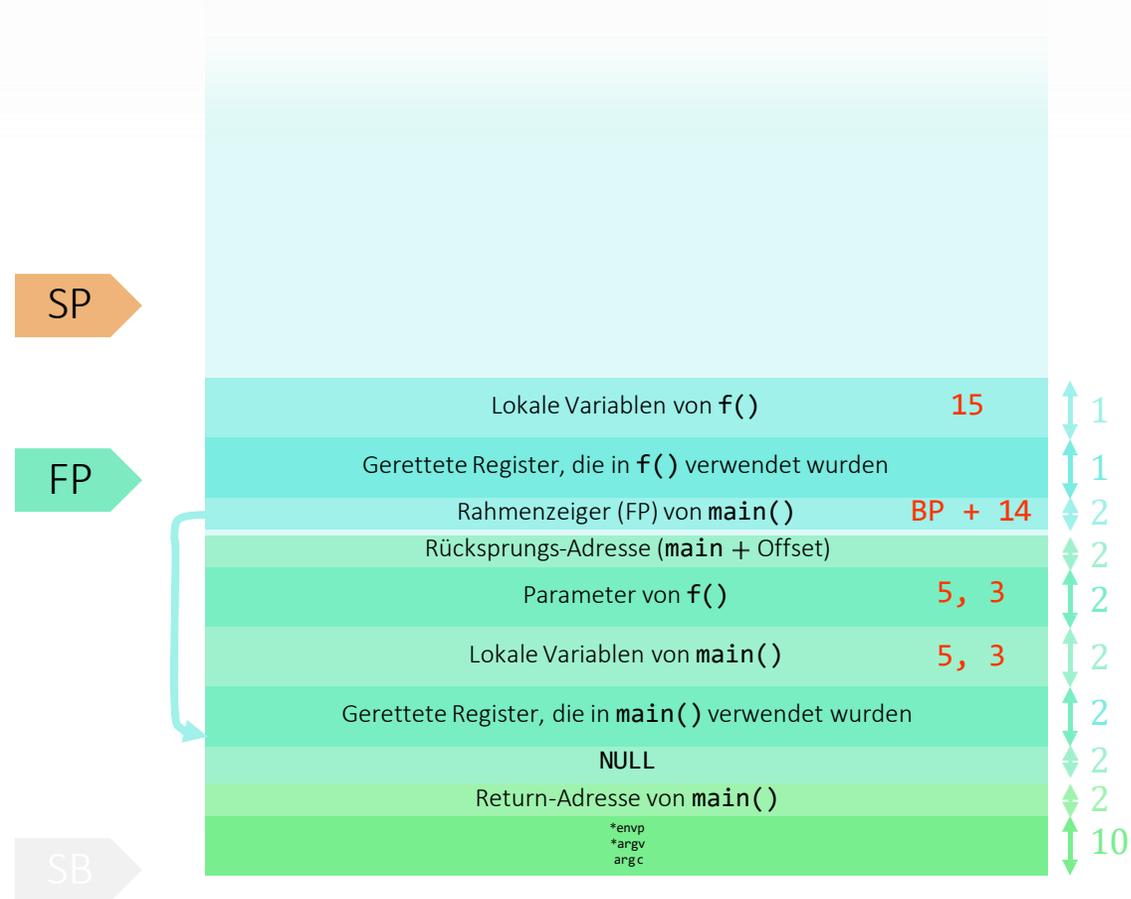


Aufruf einer weiteren Funktion

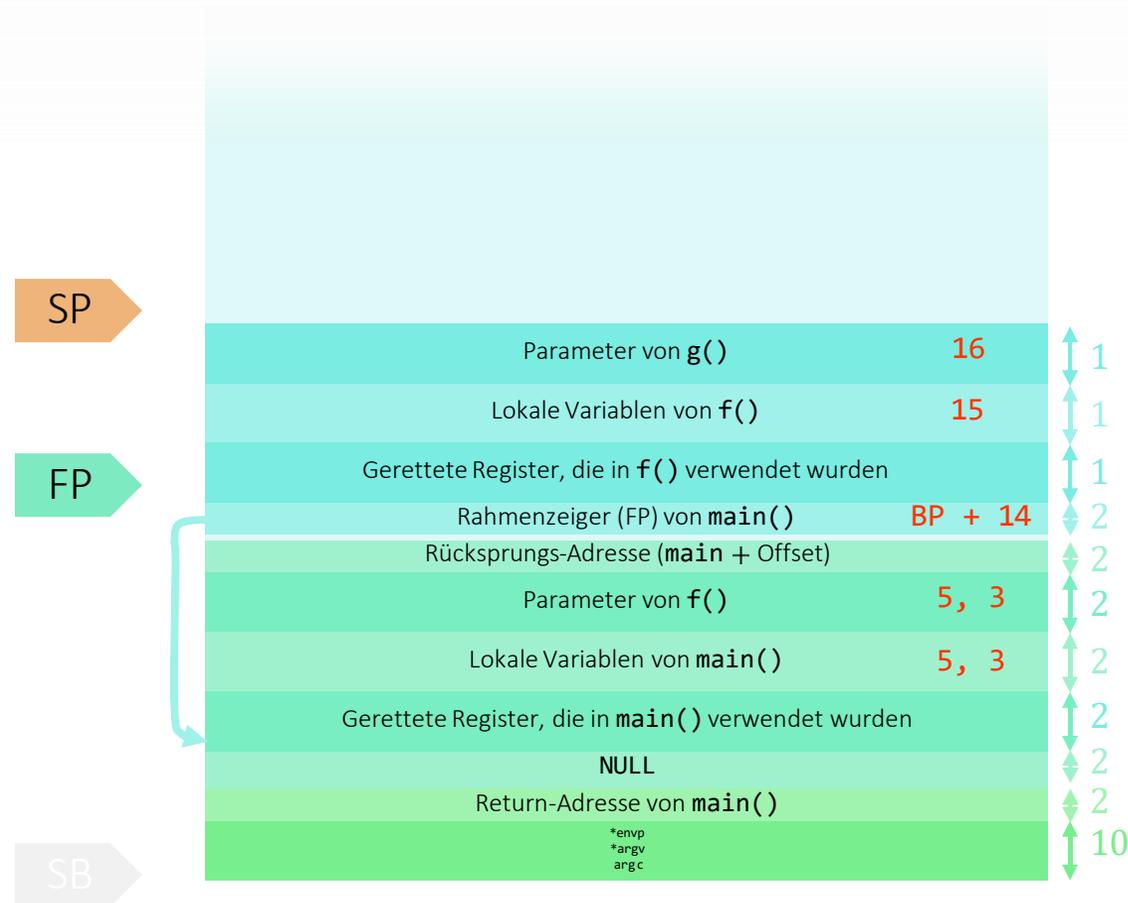
```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
 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

The diagram illustrates the flow of control between functions. A green arrow originates from the `f(x, y)` call in the `main` function and points to the opening curly brace of the `f` function definition. A yellow arrow originates from the `z = g(z+1);` line in the `f` function and points to the opening curly brace of the `g` function definition. A second green arrow originates from the `return z;` line in the `f` function and points back to the closing curly brace of the `f` function definition.

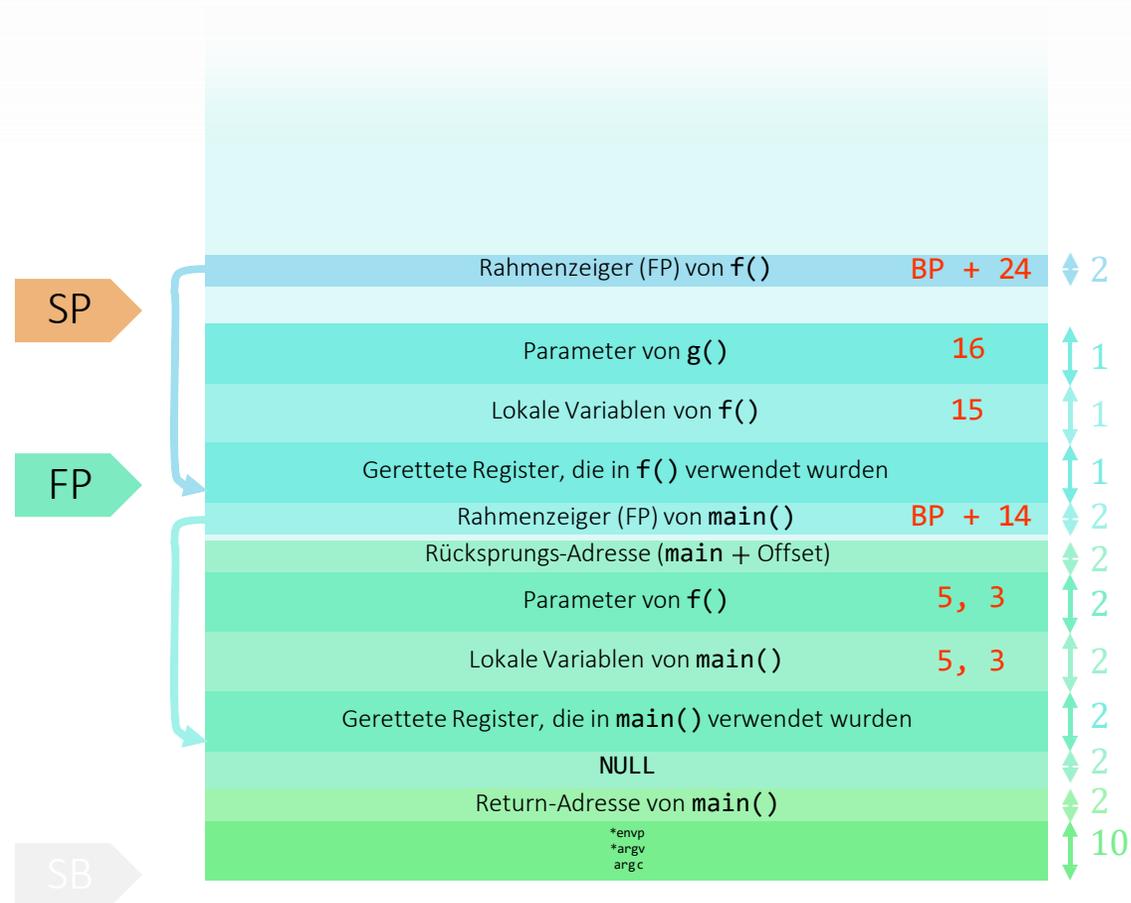
Register retten



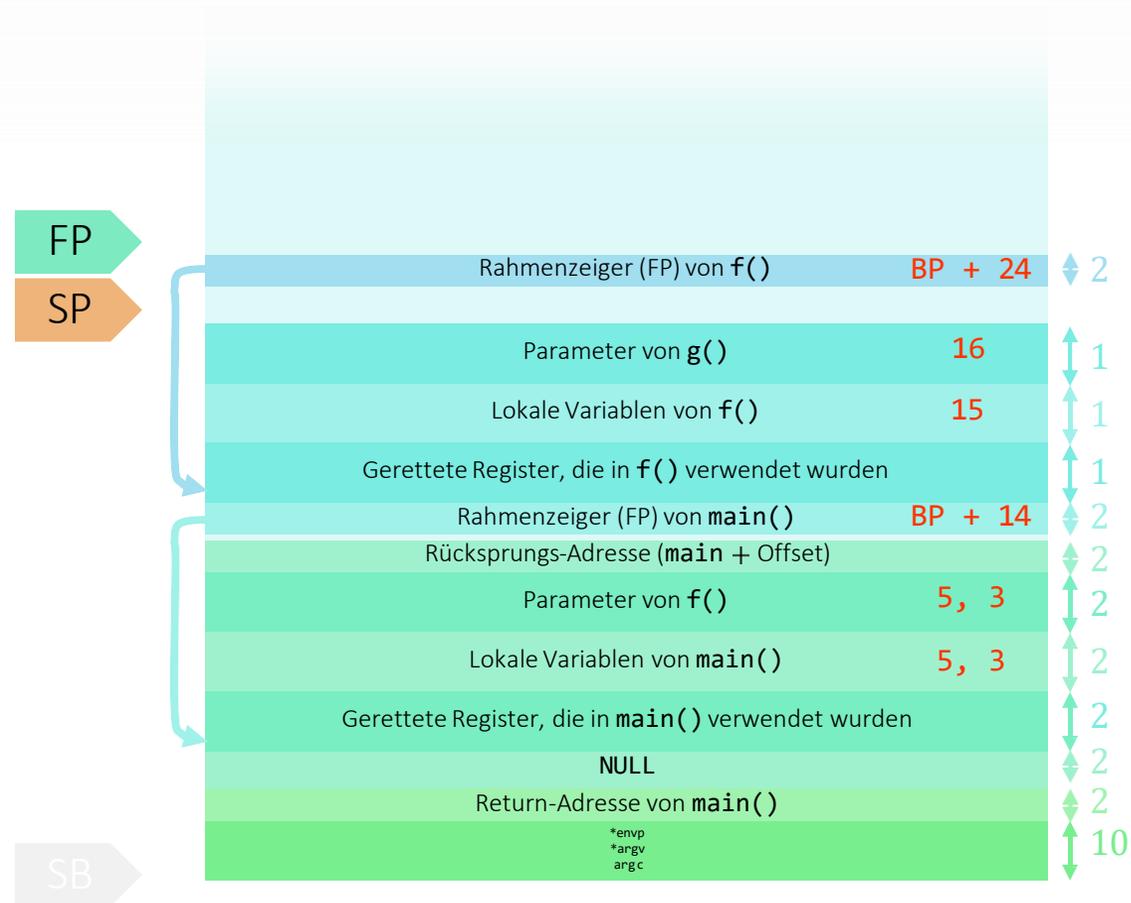
Parameter auf den Stapel legen



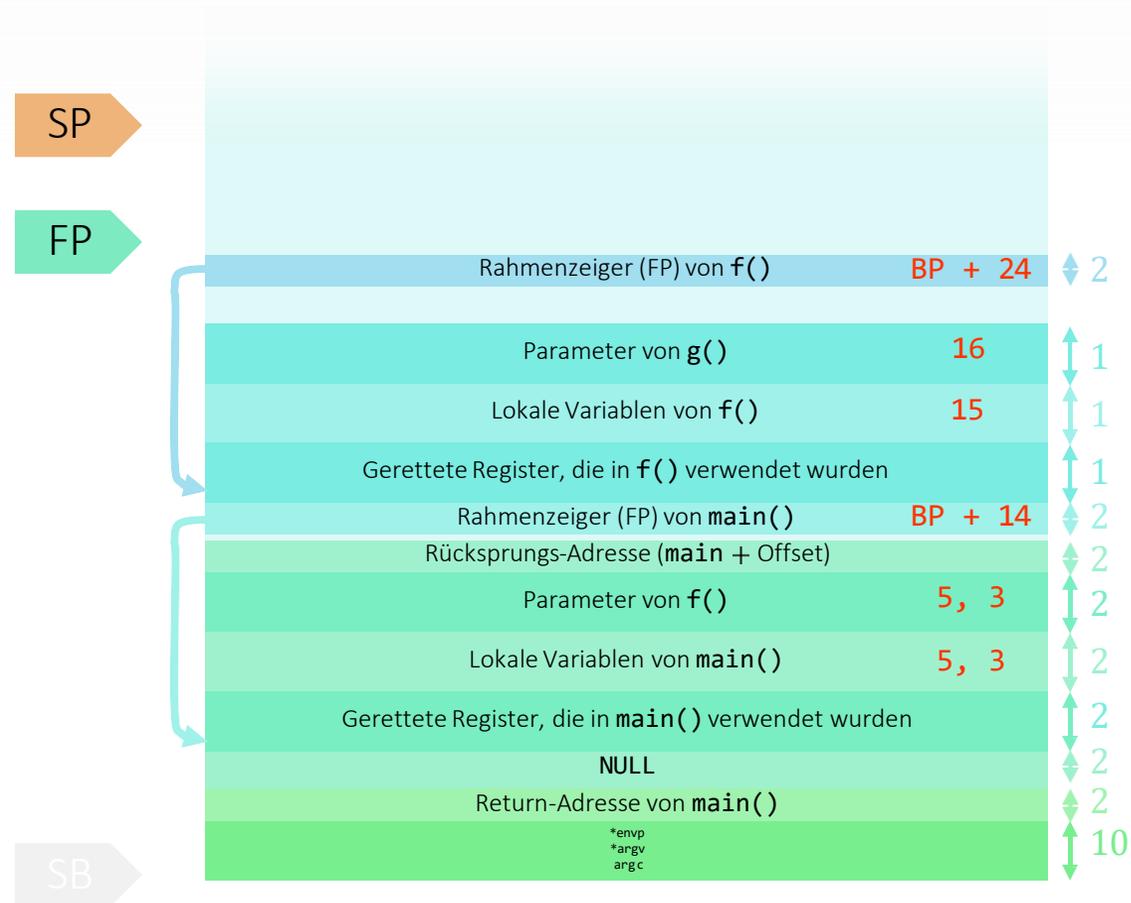
Rahmenzeiger retten



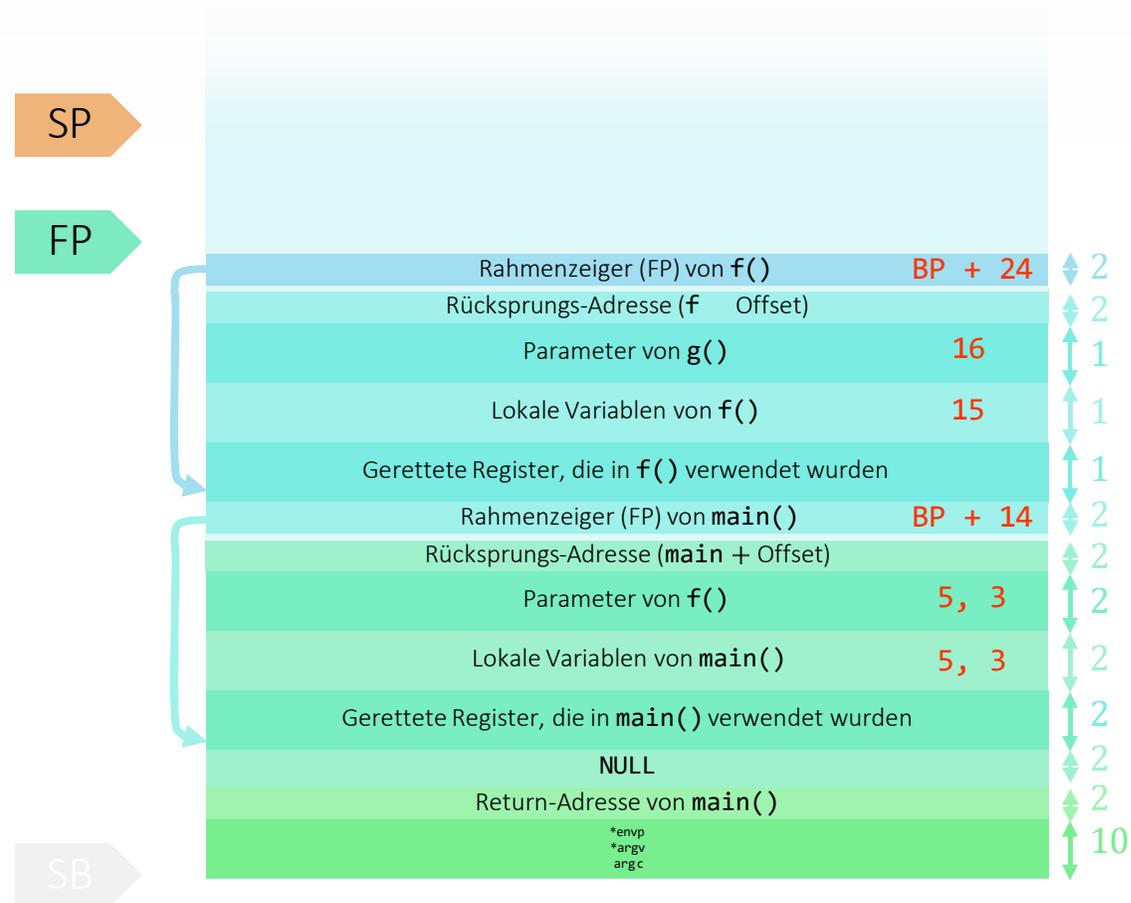
Rahmenzeiger verschieben



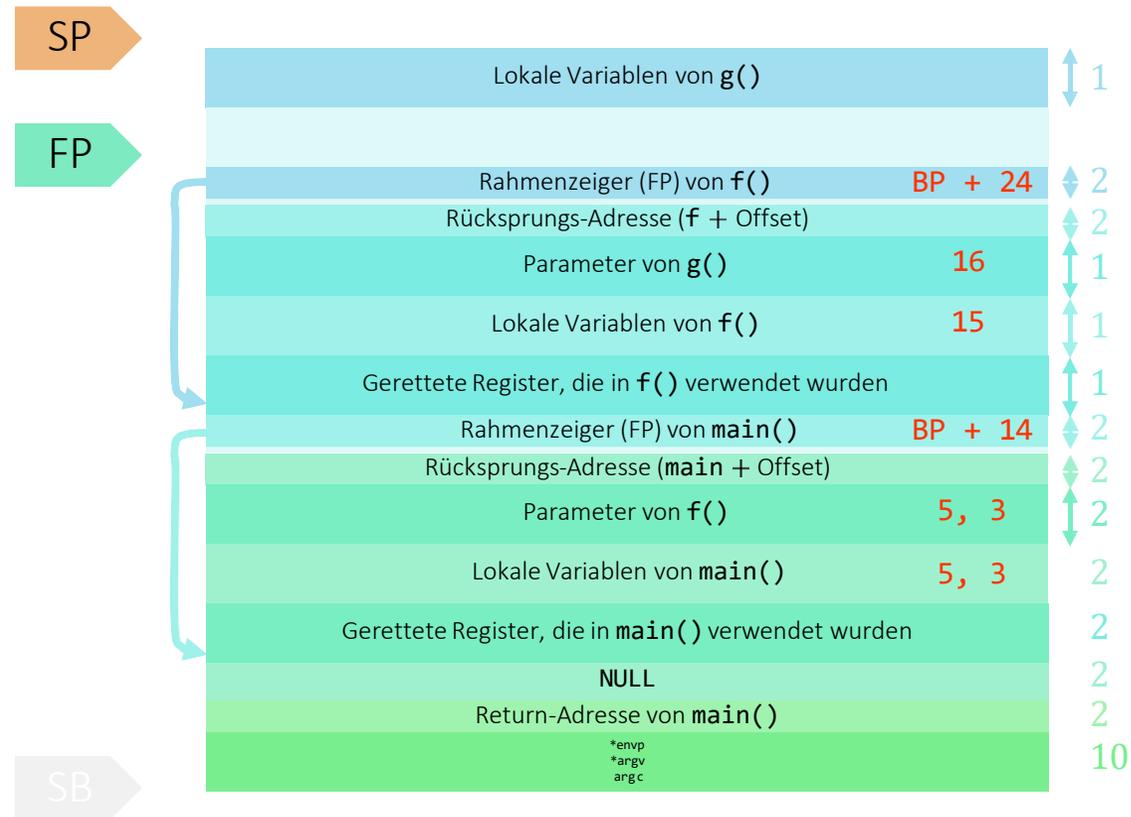
Stapelzeiger verschieben



Rücksprungadresse speichern



Unterprogrammaufruf

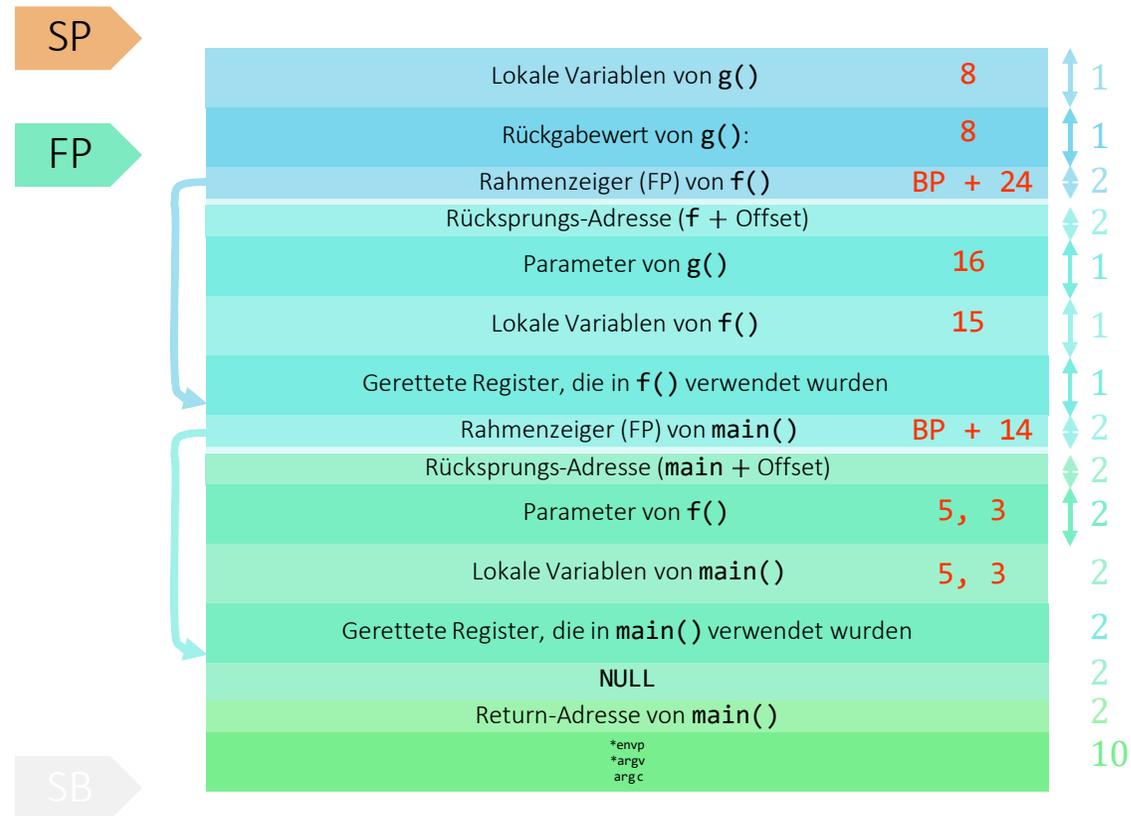


Aufruf der Return-Anweisung

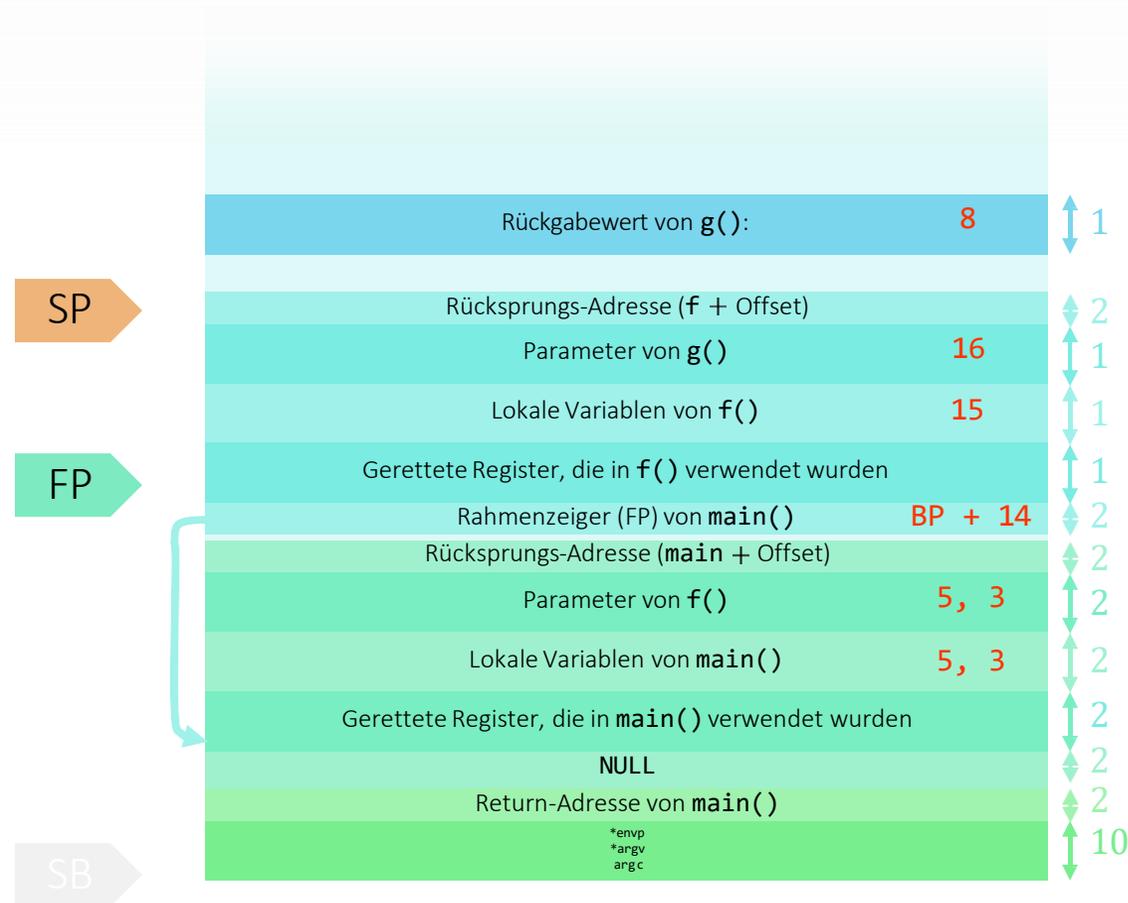
```
unsigned char g(unsigned char z) {  
    unsigned char t = z / 2;  
    return t;  
}  
  
unsigned char f(unsigned char x, unsigned char y) {  
    unsigned char z = x * y;  
    z = g(z+1);  
    return z;  
}  
  
int main(int argc, char* argv[], char* envp[]) {  
    unsigned char x = 5;  
 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

The diagram illustrates the flow of return values. A yellow arrow originates from the `return t;` statement in the `g` function and points to the `z = g(z+1);` statement in the `f` function. Another yellow arrow originates from the `return z;` statement in the `f` function and points to the `f(x, y)` call in the `main` function. A green arrow originates from the `return 0;` statement in the `main` function and points to the right, indicating the final return value of the program.

Erzeugen des Rückgabewertes



Stapel- und Rahmenzeiger wiederherstellen

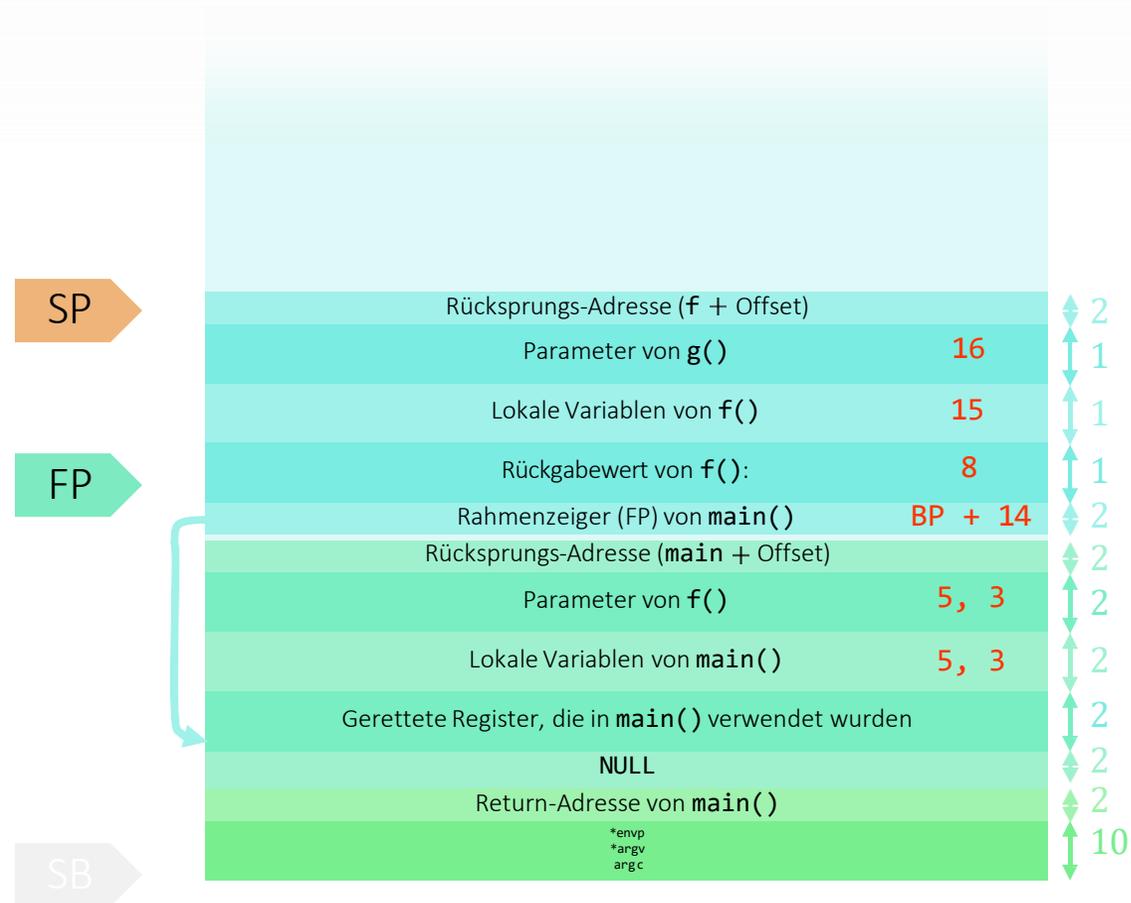


Rückkehr in die main

```
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;  
}
```

The diagram illustrates the return flow between the functions. Green arrows show the return path from the function `g` to the function `f`, and from `f` to `main`. A yellow arrow shows the return path from `main` to the caller.

Erzeugen des Rückgabewertes



Stapel- und Rahmenzeiger wiederherstellen

