

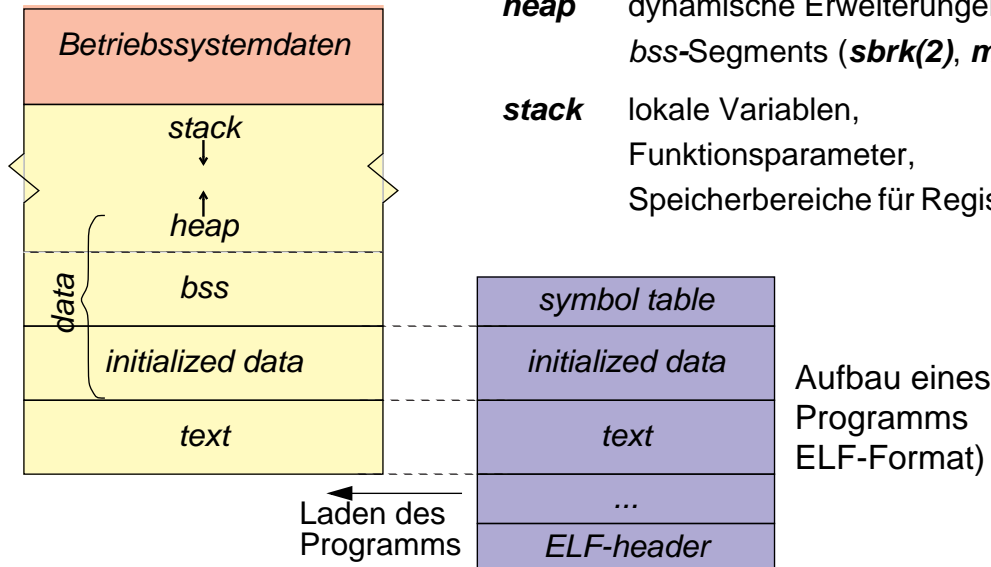
I Speicherorganisation eines Prozesses

text Programmcode
data globale und static Variablen

bss nicht initialisierte globale und *static* Variablen (wird vor der Vergabe an den Prozess mit 0 vorbelegt)

heap dynamische Erweiterungen des *bss*-Segments (**sbrk(2)**, **malloc(3)**)

stack lokale Variablen, Funktionsparameter, Speicherbereiche für Registerinhalte,



Aufbau eines Programms
ELF-Format)

Laden des Programms

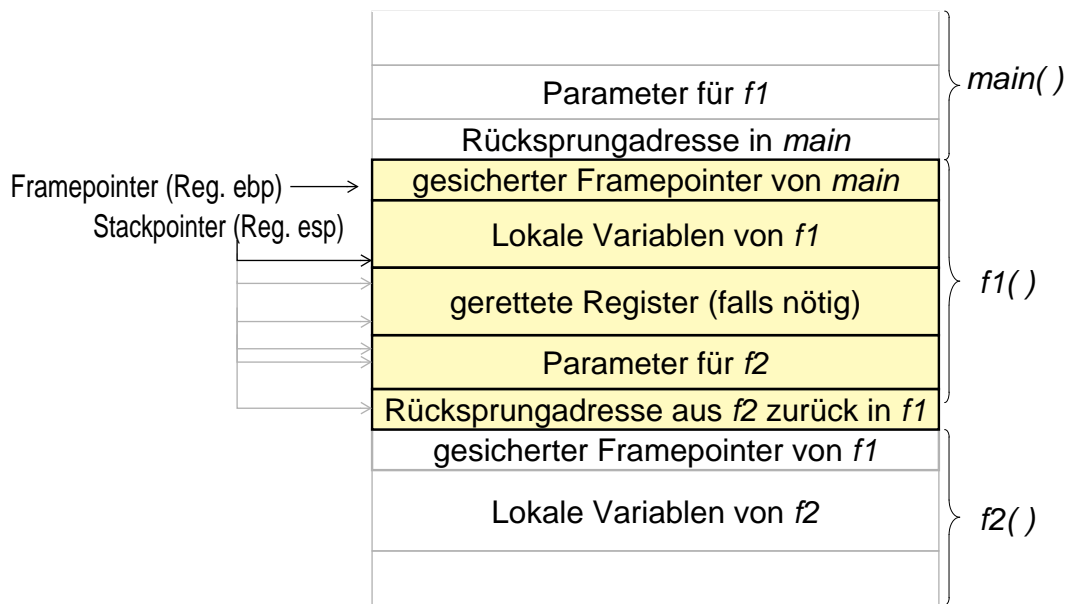
I.1 Stackaufbau eines Prozesses

1 Prinzip

- für jede Funktion wird ein **Stack-Frame** angelegt, in dem
 - lokale Variablen der Funktion
 - Aufrufparameter an weitere Funktionen
 - Registerbelegung der Funktion während des Aufrufs weiterer Funktionen
 gespeichert werden
- Stackorganisation ist abhängig von
 - Prozessor
 - Compiler und
 - Betriebssystem
- Beispiele aus einem UNIX auf Intel-Prozessor (typisch für CISC)
 - RISC-Prozessoren mit Registerfiles gehen teilweise anders vor!

2 Beispiel

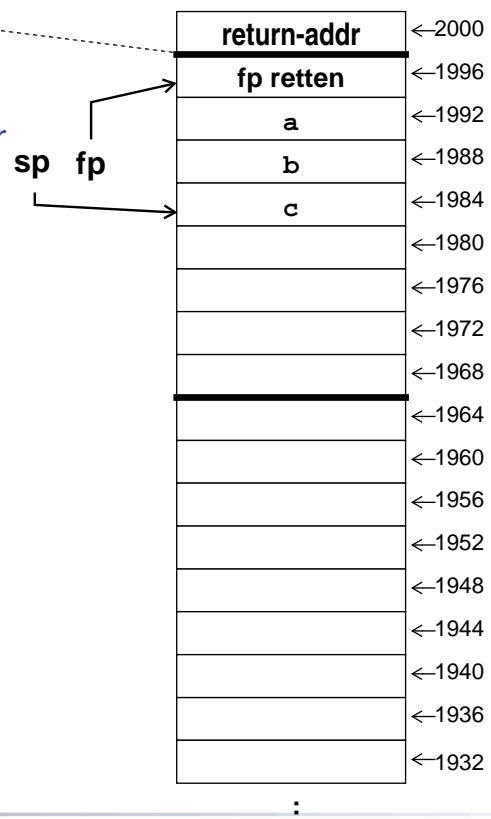
- Aufbau eines **Stack-Frames** (Funktionen *main()*, *f1()*, *f2()*)



2 ■ Stack mehrerer Funktionsaufrufe

```
main() {
    int a, b, c;
    a = 10;
    b = 20;
    f1(a, b);
    return(a);
}
```

Stack-Frame für
main erstellen
&a = fp-4
&b = fp-8
&c = fp-12



2 ■ Stack mehrerer Funktionsaufrufe

```

main() {
    int a, b, c;

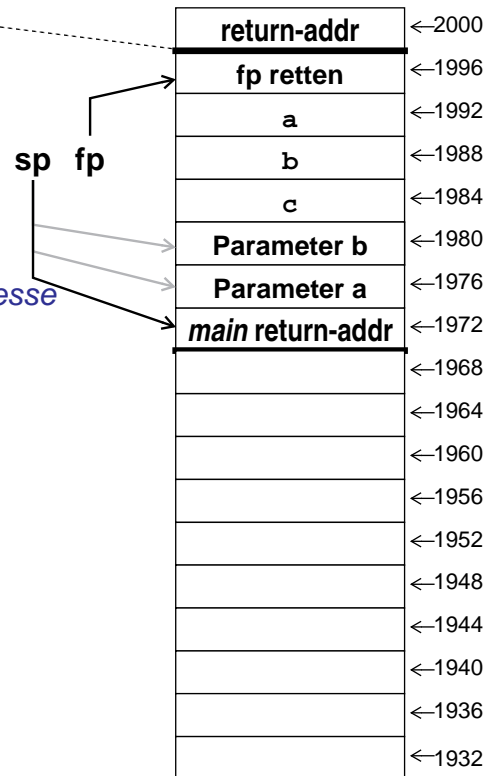
    a = 10;
    b = 20;

    f1(a, b);

    return(a);
}

```

Parameter
auf Stack legen
Bei Aufruf
Rücksprungadresse
auf Stack legen



2 ■ Stack mehrerer Funktionsaufrufe

```

main() {
    int a, b, c;

    a = 10;
    b = 20;

    f1(a, b);

    return(a);
}

```

```

int f1(int x, int y) {
    int i[3];
    int n;

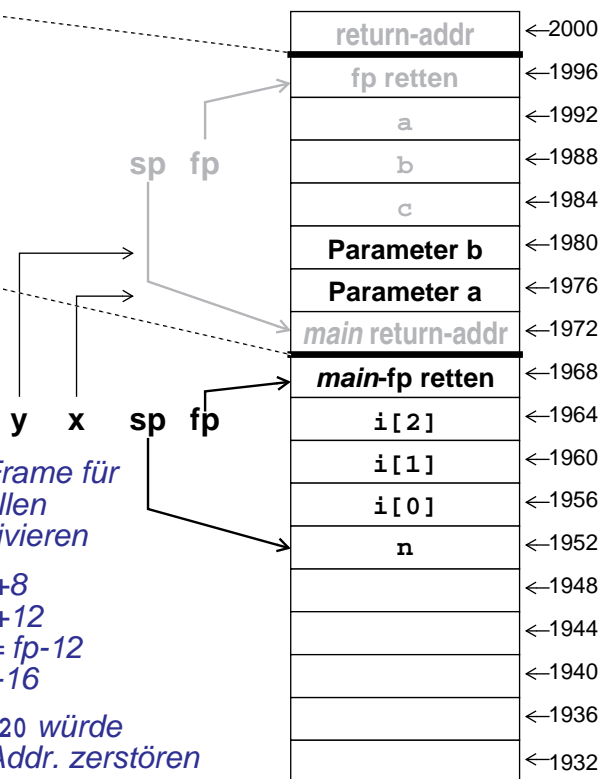
    x++;

    n = f2(x);

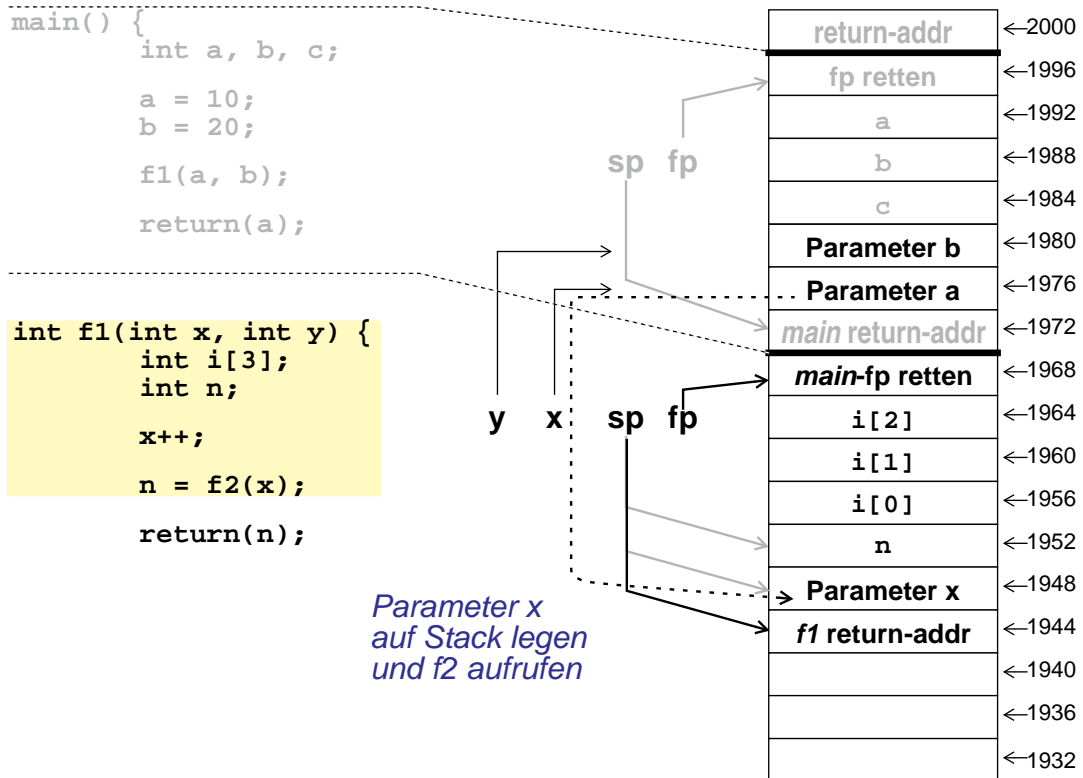
    return(n);
}

```

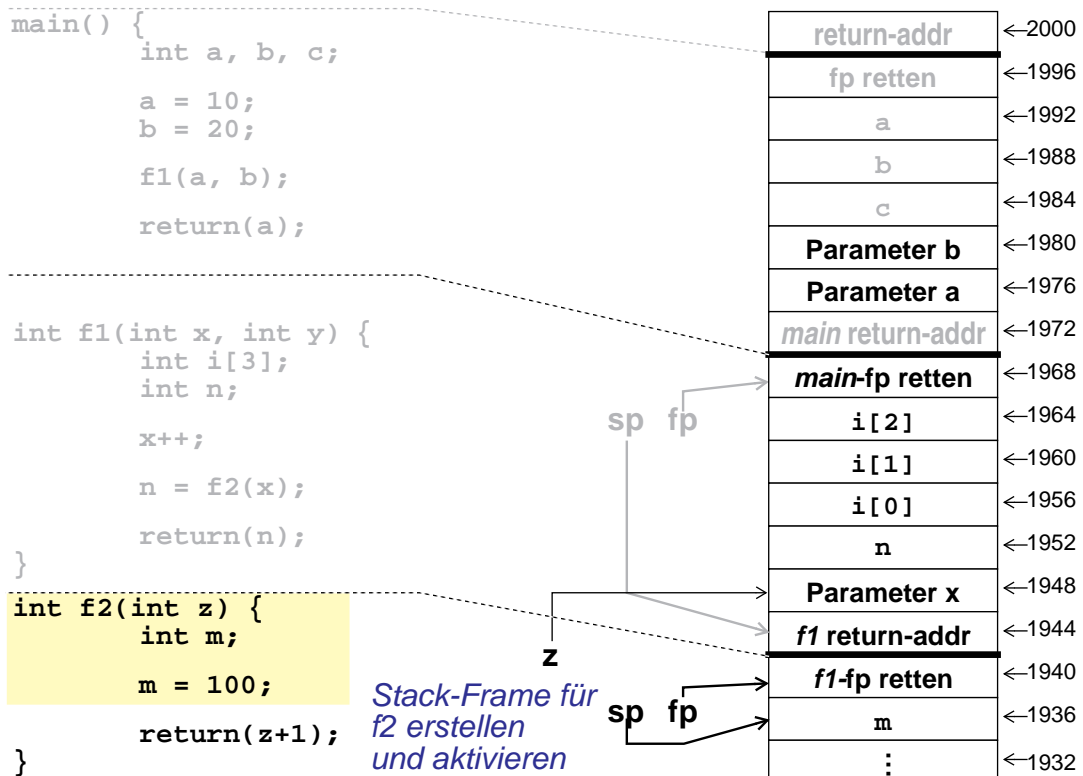
Stack-Frame für
f1 erstellen
und aktivieren
 $&x = fp+8$
 $&y = fp+12$
 $&(i[0]) = fp-12$
 $&n = fp-16$
 $i[4] = 20$ würde
return-Addr. zerstören



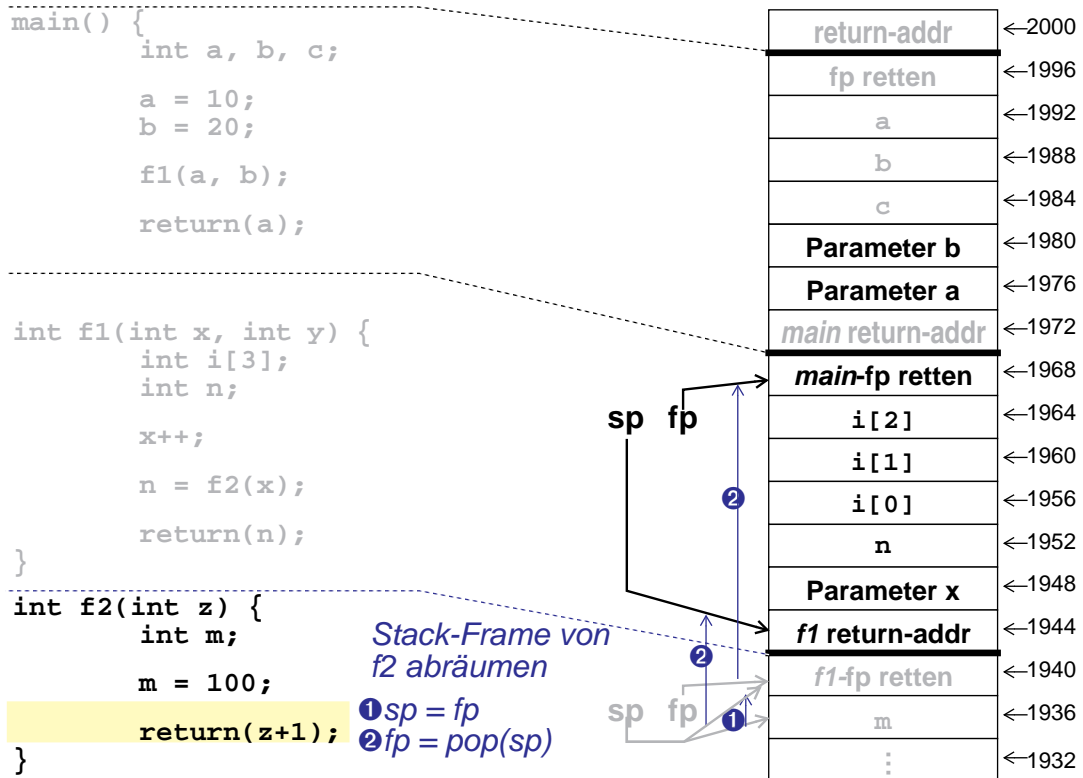
2 ■ Stack mehrerer Funktionsaufrufe



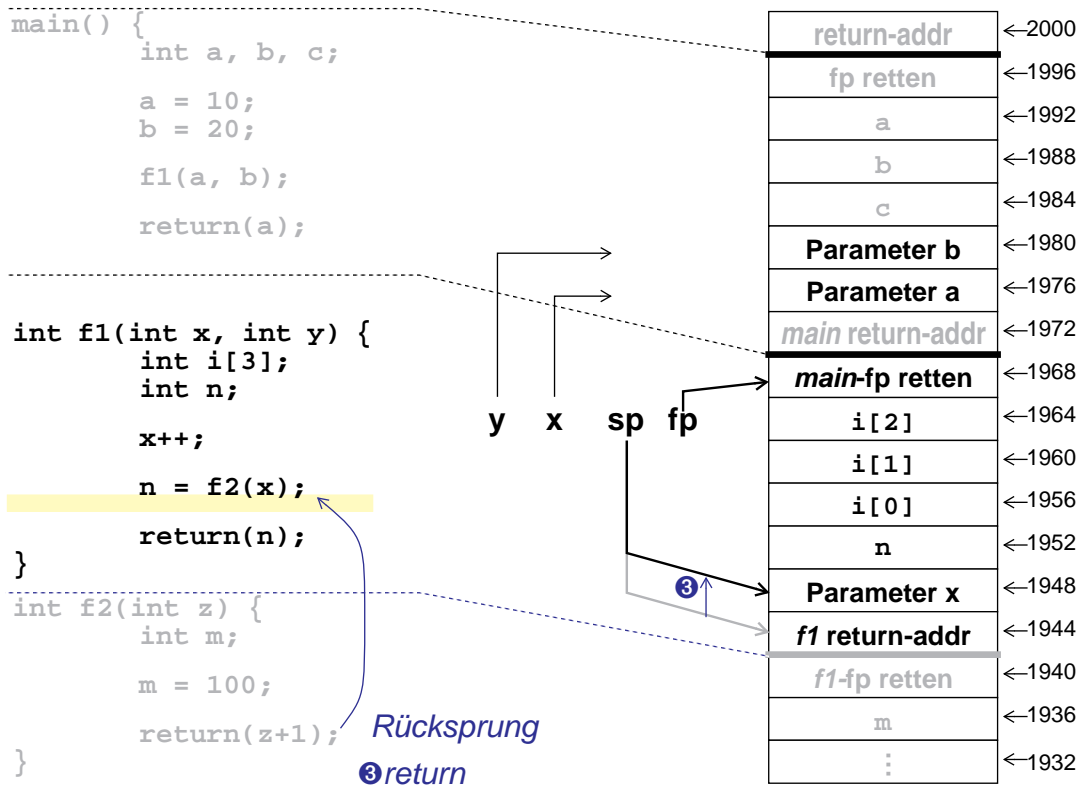
2 ■ Stack mehrerer Funktionsaufrufe



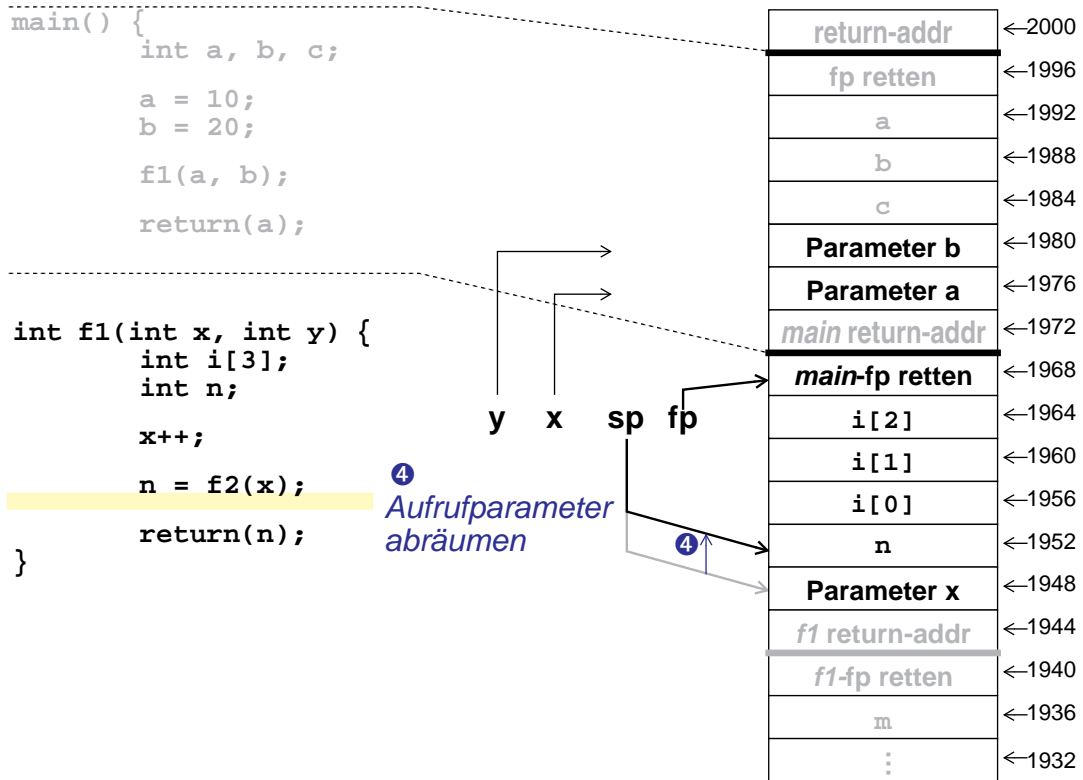
2 ■ Stack mehrerer Funktionsaufrufe



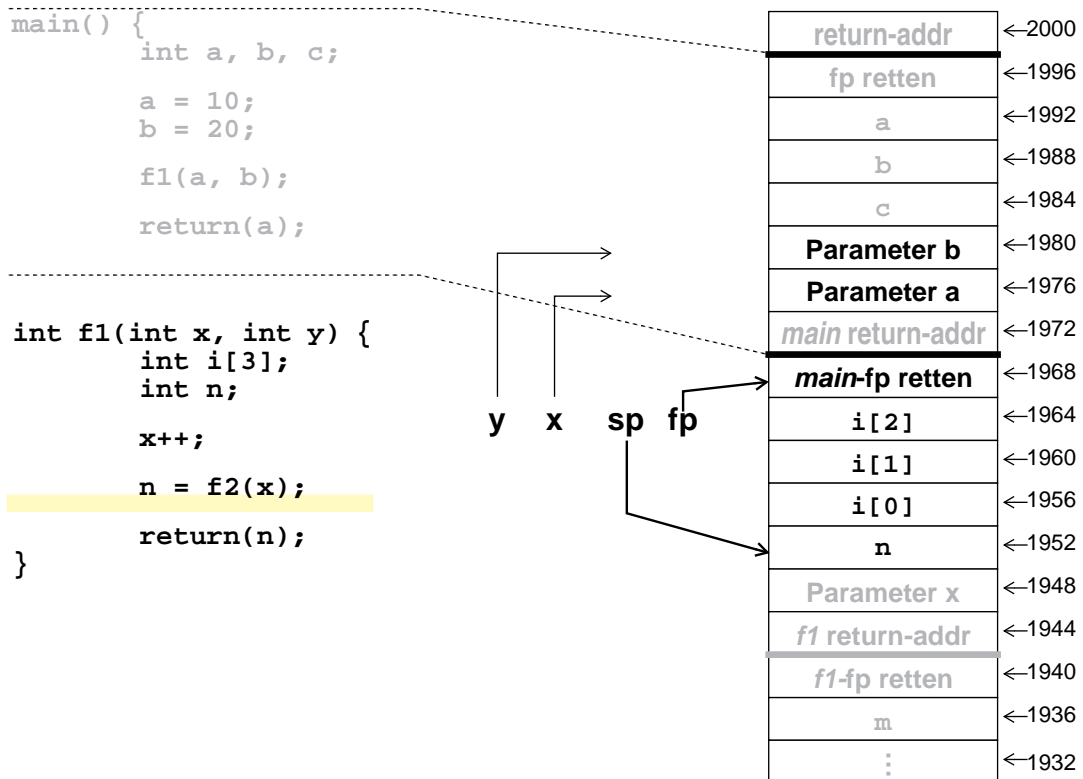
2 ■ Stack mehrerer Funktionsaufrufe



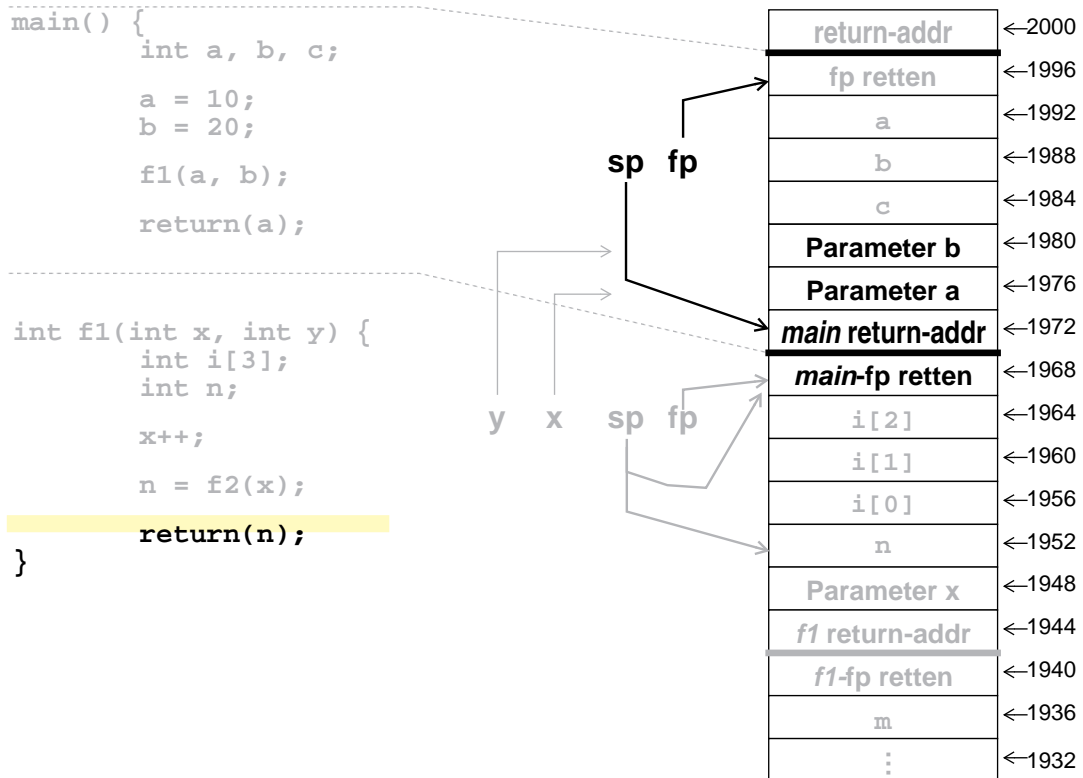
2 ■ Stack mehrerer Funktionsaufrufe



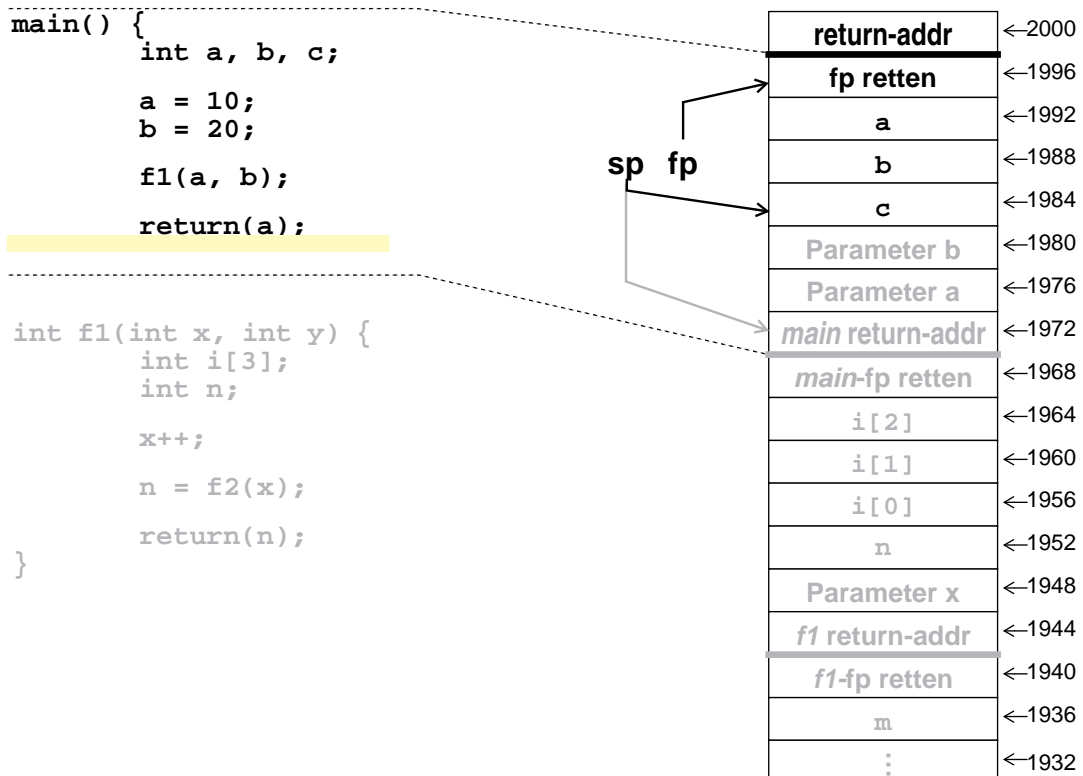
2 ■ Stack mehrerer Funktionsaufrufe



2 ■ Stack mehrerer Funktionsaufrufe



2 ■ Stack mehrerer Funktionsaufrufe



2 ■ Stack mehrerer Funktionsaufrufe

```

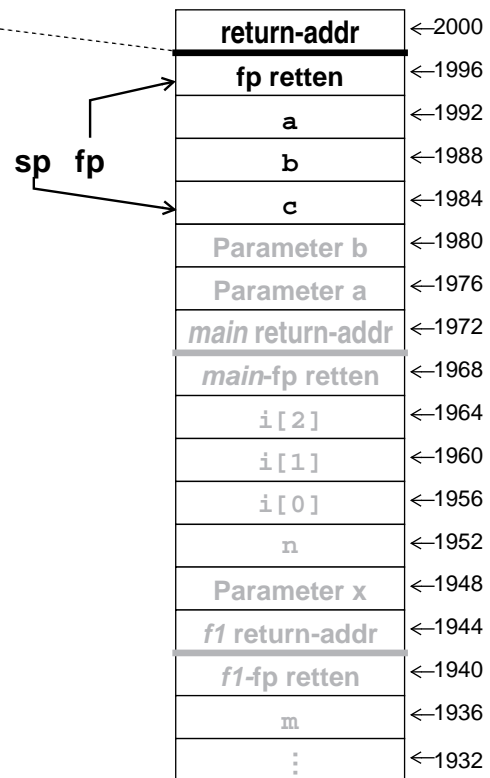
main() {
    int a, b, c;

    a = 10;
    b = 20;

    f1(a, b);

    return(a);
}

```



2 ■ Stack mehrerer Funktionsaufrufe

```

main() {
    int a, b, c;

    a = 10;
    b = 20;

    f1(a, b);

    f2(200);
}

```

was wäre, wenn man nach
f1 jetzt nochmal f2
aufrufen würde?

```

int f2(int z) {
    int m;

    m = 100;

    return(z+1);
}

```

