

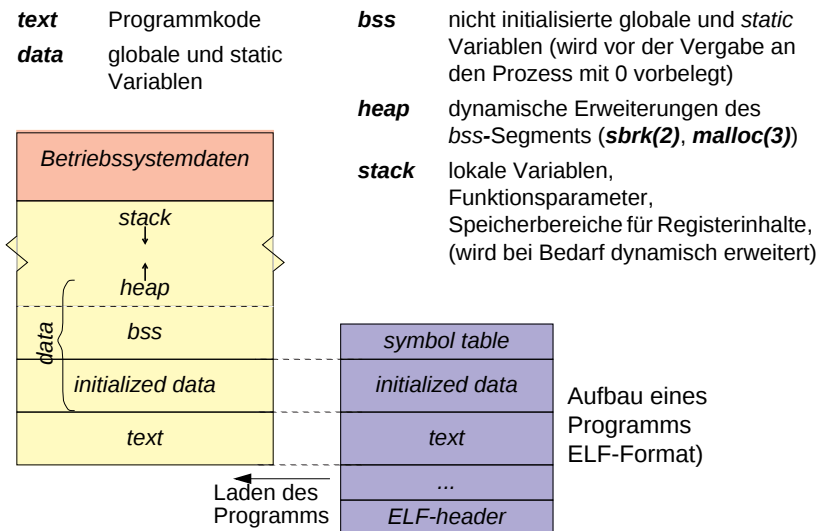
H Programme und Prozesse

- **Programm:** Folge von Anweisungen (hinterlegt beispielsweise als ausführbare Datei auf dem Hintergrundspeicher)
- **Prozess:** Betriebssystemkonzept
 - Programm, das sich in Ausführung befindet, und seine Daten (Beachte: ein Programm kann sich mehrfach in Ausführung befinden)
 - eine konkrete Ausführungsumgebung für ein Programm Speicher, Rechte, Verwaltungsinformation (verbrauchte Rechenzeit,...),...
- jeder Prozess bekommt einen eigenen virtuellen Adressraum zur Verfügung gestellt
 - eigener (virtueller) Speicherbereich von 0 bis 2 GB (oder mehr bis 4 GB)
 - Datenbereiche von verschiedenen Prozessen und Betriebssystem sind gegeneinander geschützt
 - Datentransfer zwischen Prozessen nur durch Vermittlung des Betriebssystems möglich

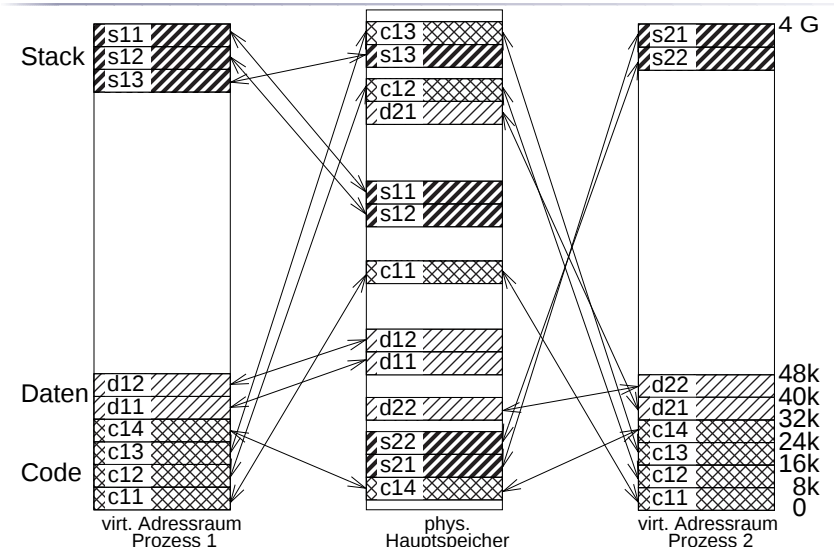
H.1 Speicherorganisation eines Prozesses (2)

- Abbildung des virtuellen Adressraums in den realen Hauptspeicher durch Seitenadressierung (**Paging**)
 - Adreßraum ist in kleine (4 oder 8 kB) Stücke unterteilt (**Seiten**)
 - jede Seite wird über eine Tabelle in ein entsprechendes Stück des Hauptspeichers (**Kachel**) abgebildet
 - bei jedem Speicherzugriff wird die virtuelle Adresse in die entsprechende physikalische Adresse umgerechnet (spezielle Hardware: Memory Management Unit - MMU)
 - zu jeder Seite sind Zugriffsrechte vermerkt (nur lesen, lesen+schreiben, Maschinenbefehle ausführen)
 - eine Seite kann bei Speichermangel von Betriebssystem auf Festplatte ausgelagert werden und bei Bedarf automatisch wieder eingelagert werden

H.1 Speicherorganisation eines Prozesses



H.1 Speicherorganisation eines Prozesses(3)



H.2 Stackaufbau eines Prozesses

H.2 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 Funktionengespeichert 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!

Gdl2 - Systemnahe Programmierung in C

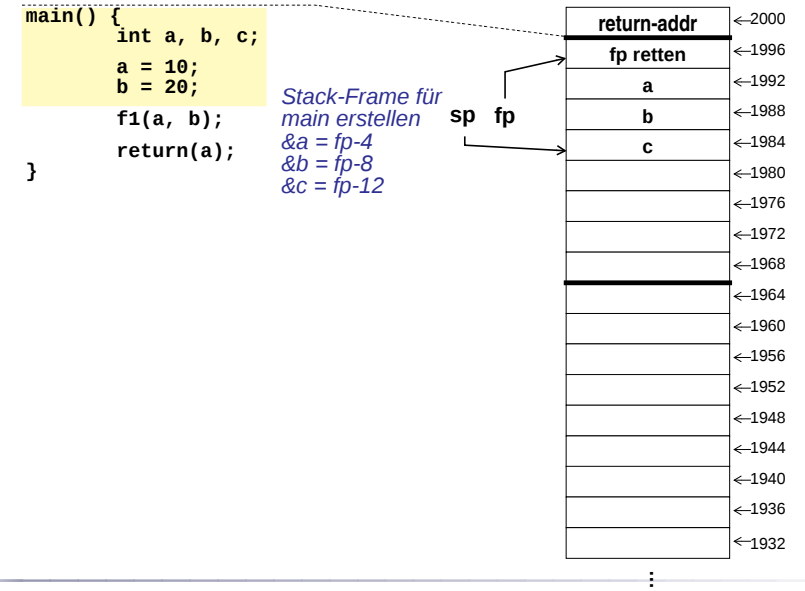
© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2009

H-Prozess.fm 2009-06-03 11.13

H.5

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 ■ Stack mehrerer Funktionsaufrufe



Gdl2 - Systemnahe Programmierung in C

© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2009

H-Prozess.fm 2009-06-03 11.13

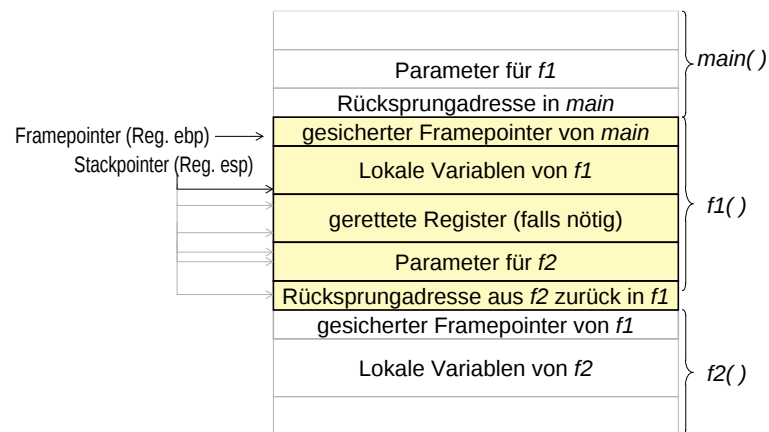
H.7

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Beispiel

H.2 Stackaufbau eines Prozesses

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



Gdl2 - Systemnahe Programmierung in C

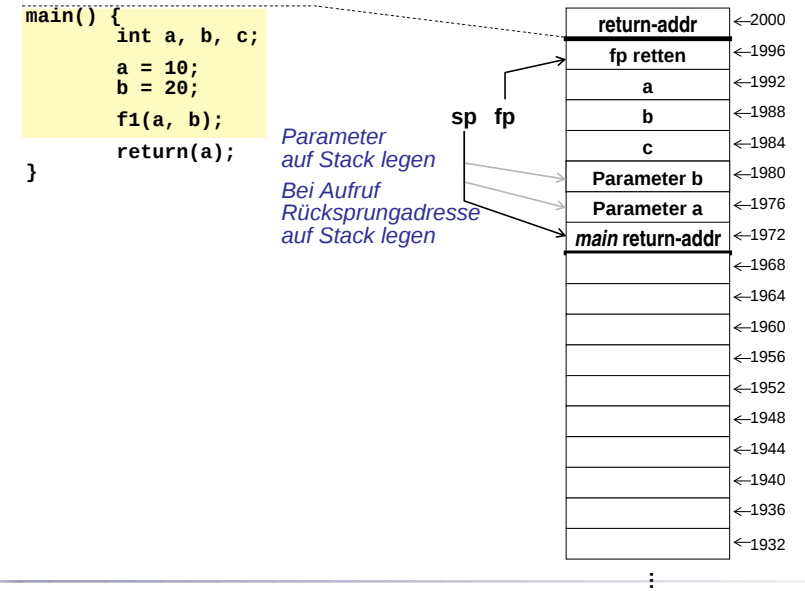
© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2009

H-Prozess.fm 2009-06-03 11.13

H.6

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 ■ Stack mehrerer Funktionsaufrufe



Gdl2 - Systemnahe Programmierung in C

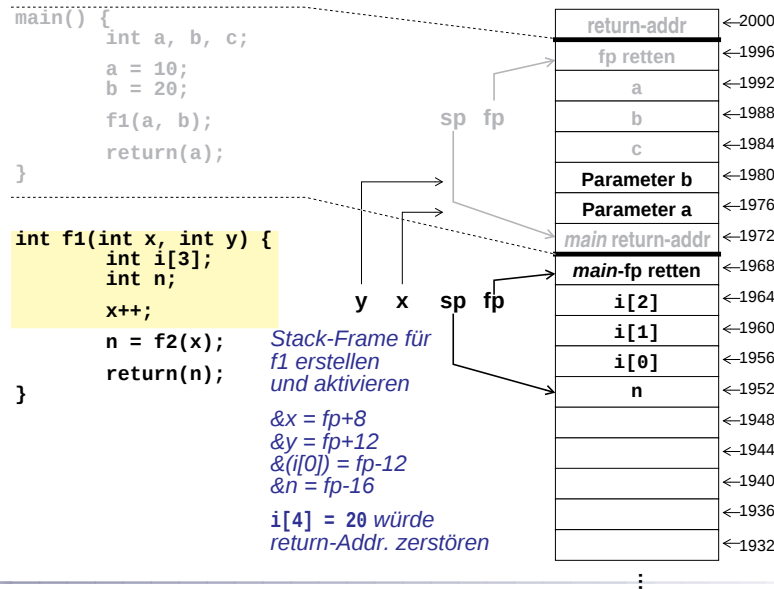
© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2009

H-Prozess.fm 2009-06-03 11.13

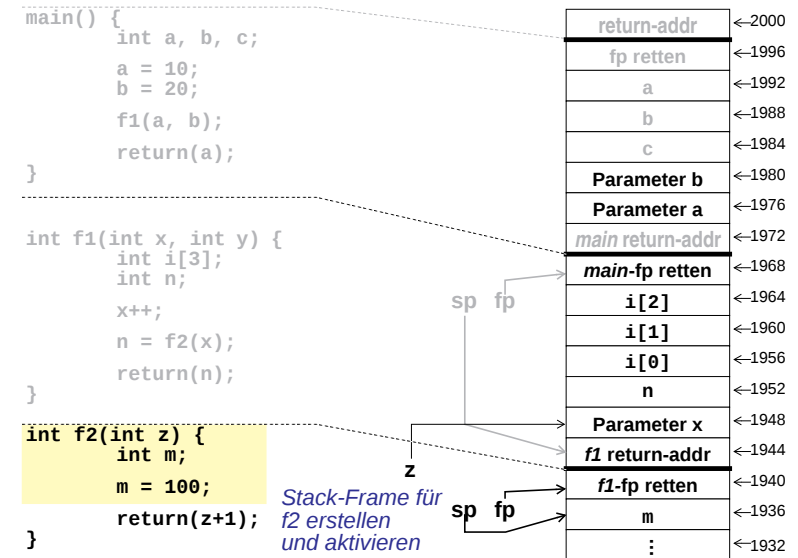
H.8

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

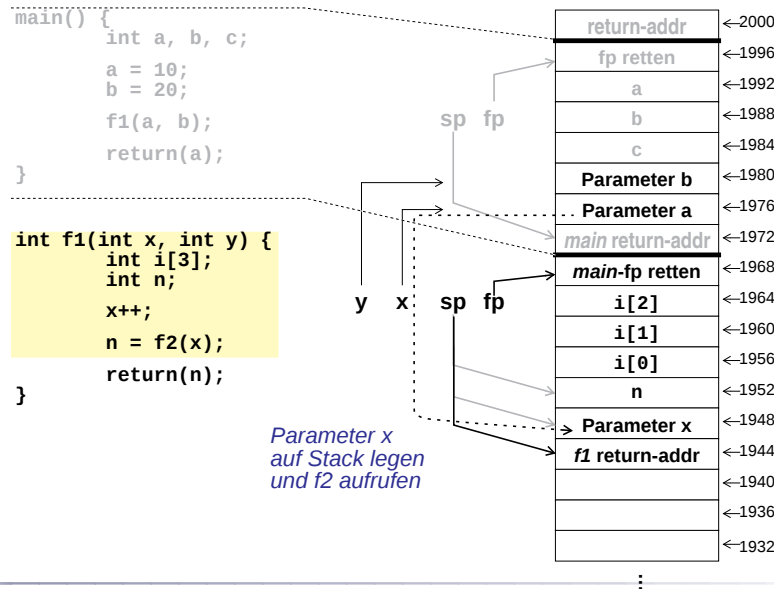
2 ■ Stack mehrerer Funktionsaufrufe



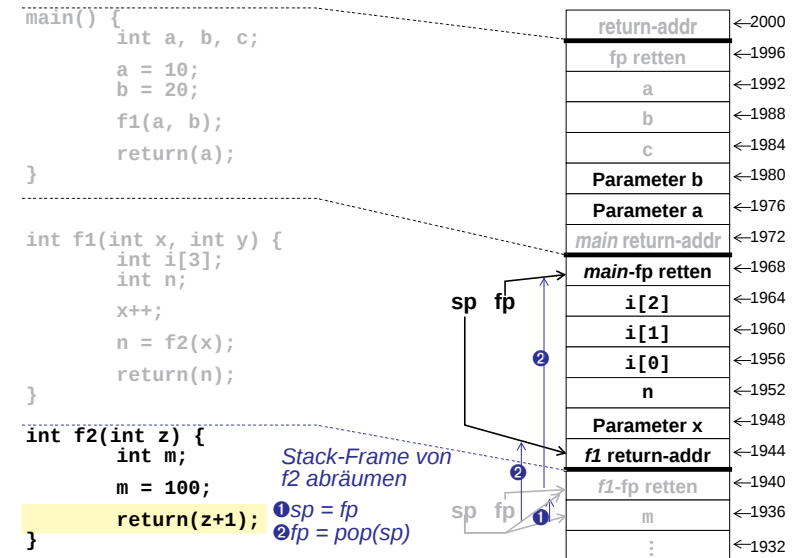
2 ■ Stack mehrerer Funktionsaufrufe



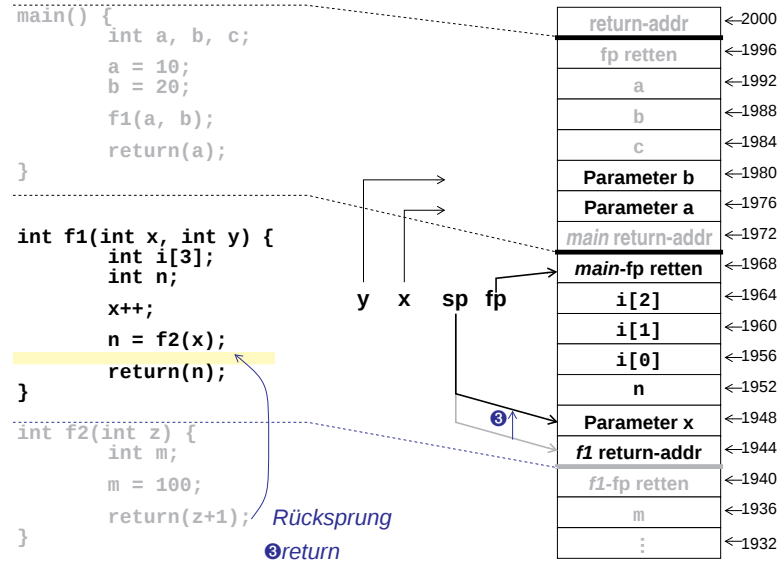
2 ■ Stack mehrerer Funktionsaufrufe



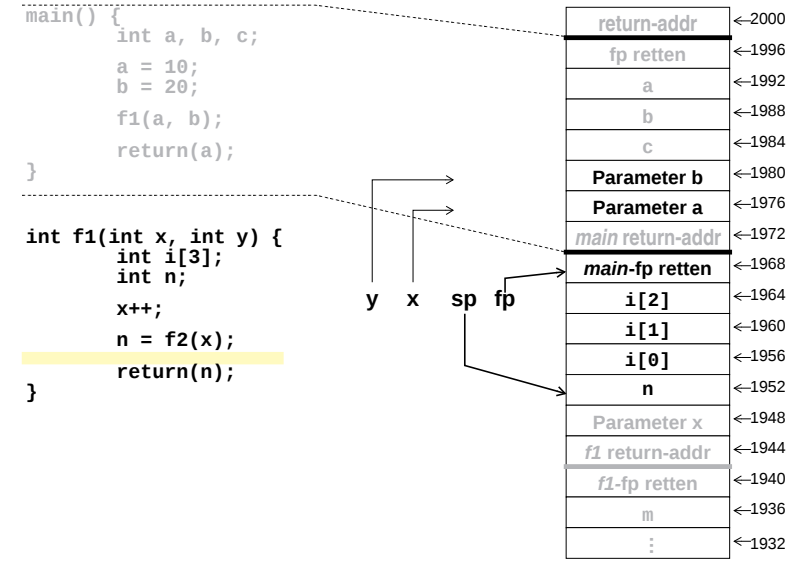
2 ■ Stack mehrerer Funktionsaufrufe



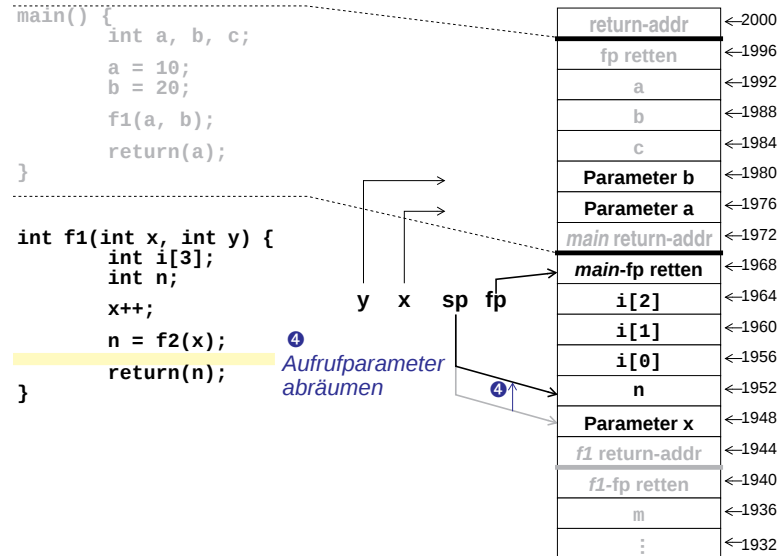
2 ■ Stack mehrerer Funktionsaufrufe



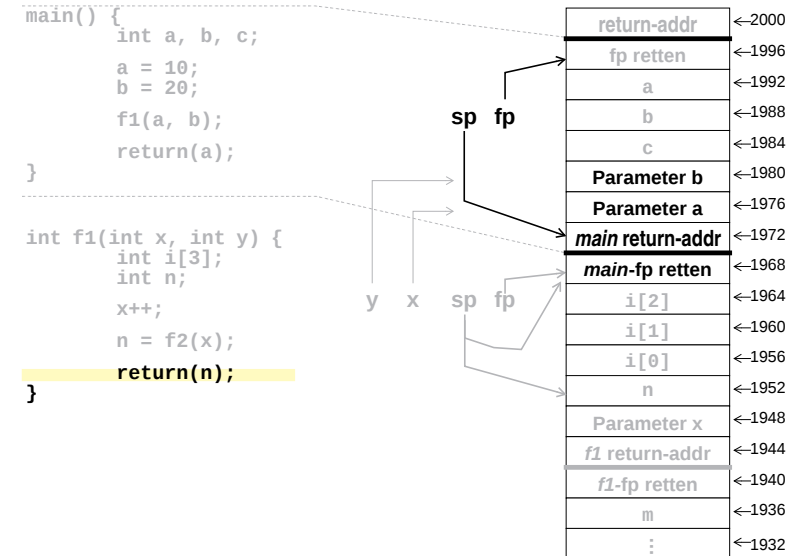
2 ■ Stack mehrerer Funktionsaufrufe



2 ■ Stack mehrerer Funktionsaufrufe



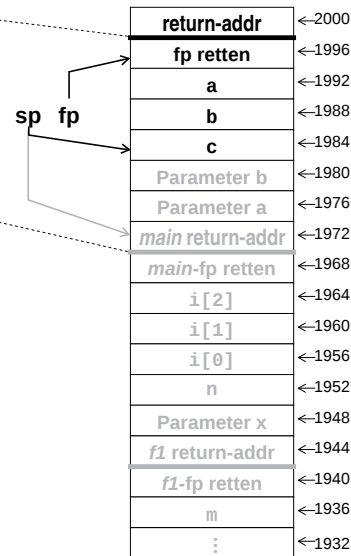
2 ■ Stack mehrerer Funktionsaufrufe



2 ■ Stack mehrerer Funktionsaufrufe

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

```
int f1(int x, int y) {
    int i[3];
    int n;
    x++;
    n = f2(x);
    return(n);
}
```

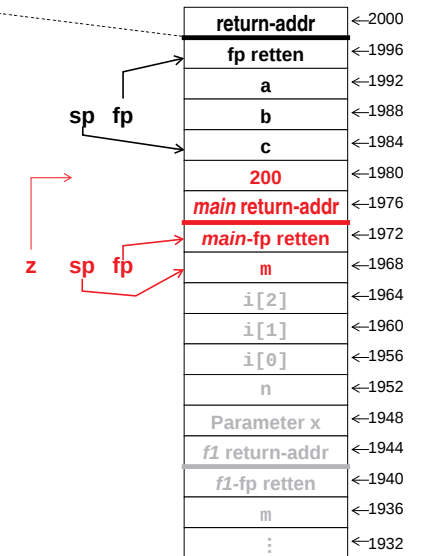


2 ■ Stack mehrerer Funktionsaufrufe

```
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);
}
```



2 ■ Stack mehrerer Funktionsaufrufe

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

