1. Einführung

Prof. Dr. Markus Gross

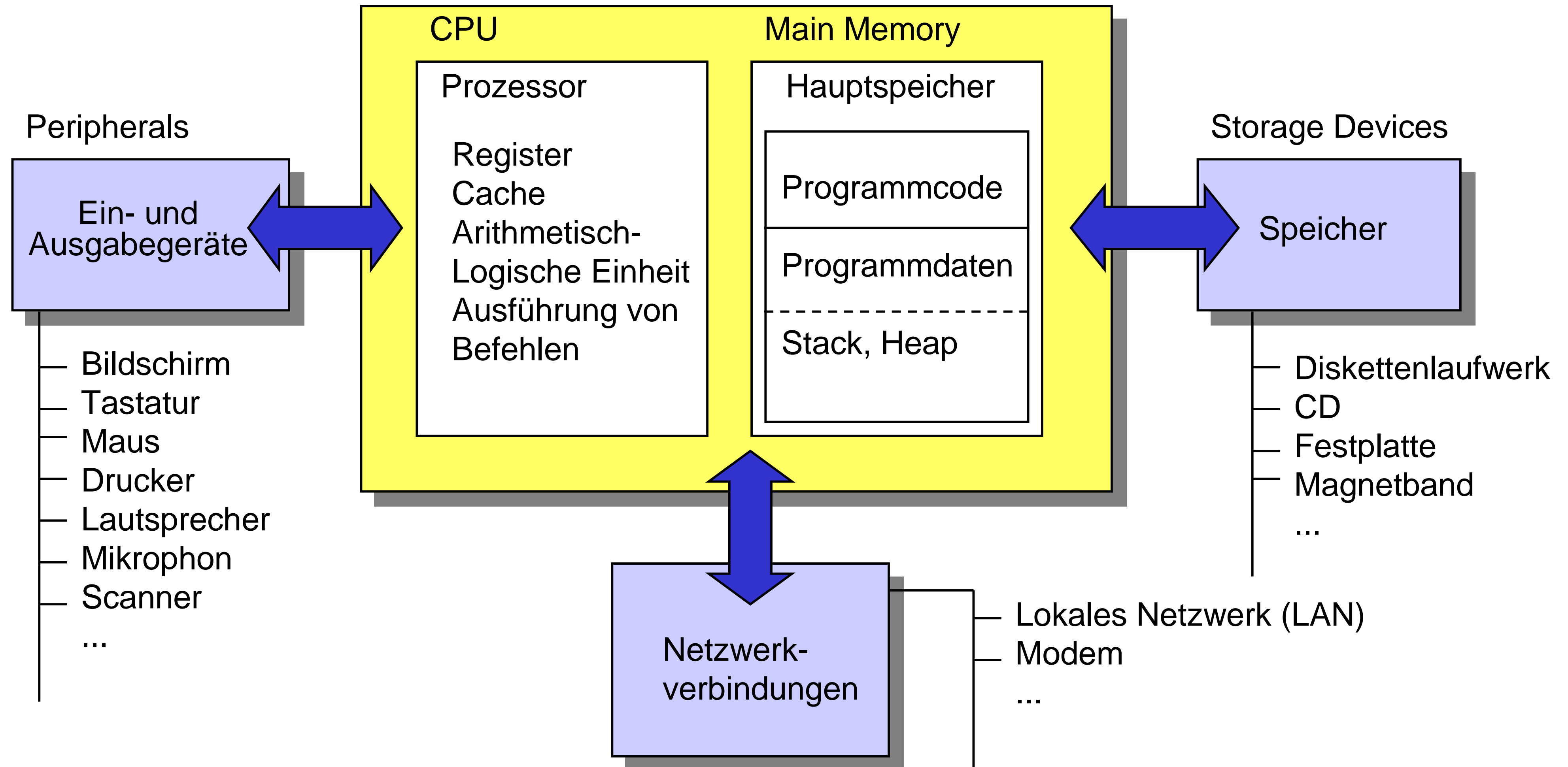
Informatik I für D-ITET (WS 03/04)

- Aufbau eines Computers
- Systemumgebung
- Vorgang des Programmierens
- Editor, Debugger, Linker
- Programmiersprachen - Geschichte

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Der Computer – von Neumann



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Von Neumann‘sche Maschine

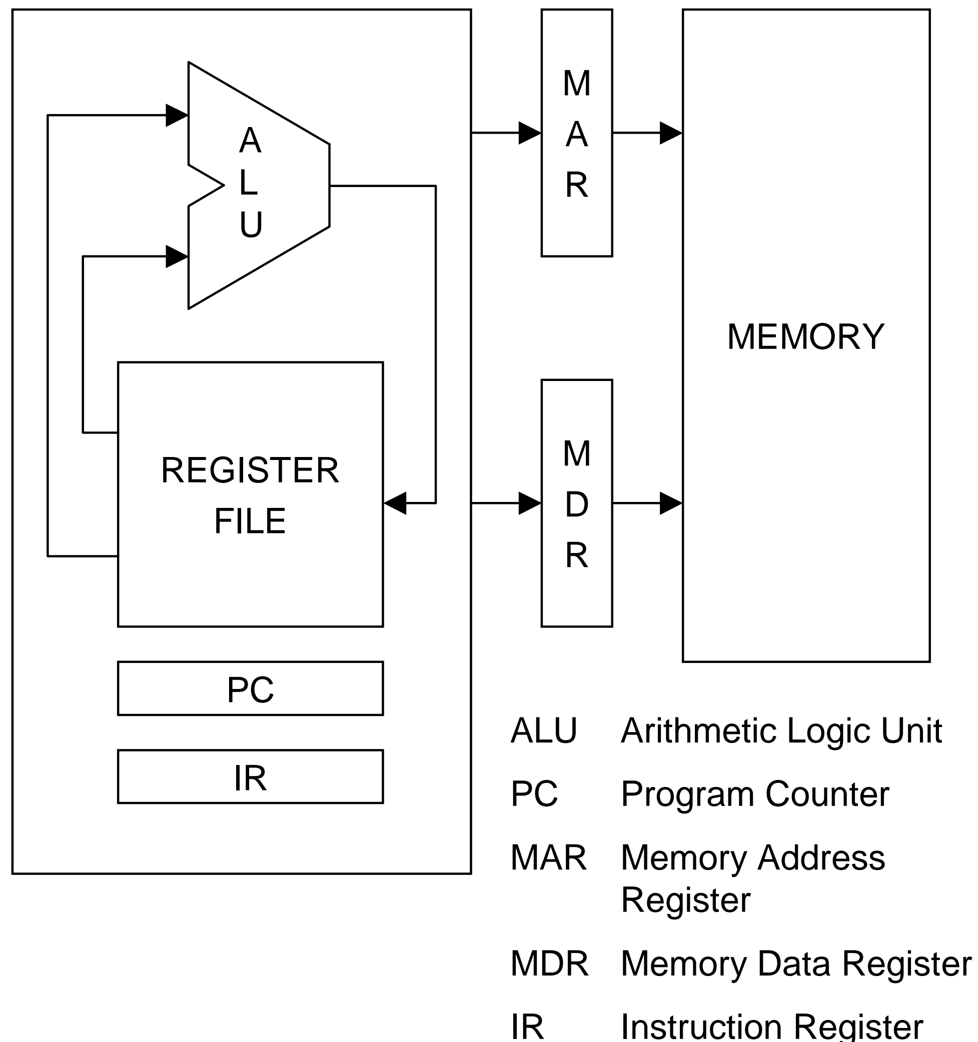
```
// von Neumann Zyklus


pc = 0;

do
{
      instruction=memory[pc++];
      decode(intruction);
      fetch (operands);
      execute;
      store(results);
}
while (intruction !=halt);
```

- Erster General Purpose Computer ENIAC
- 20 Register
- 18000 Röhren
- Integration von Daten- und Programmspeicher
- Adressierbarer Speicher

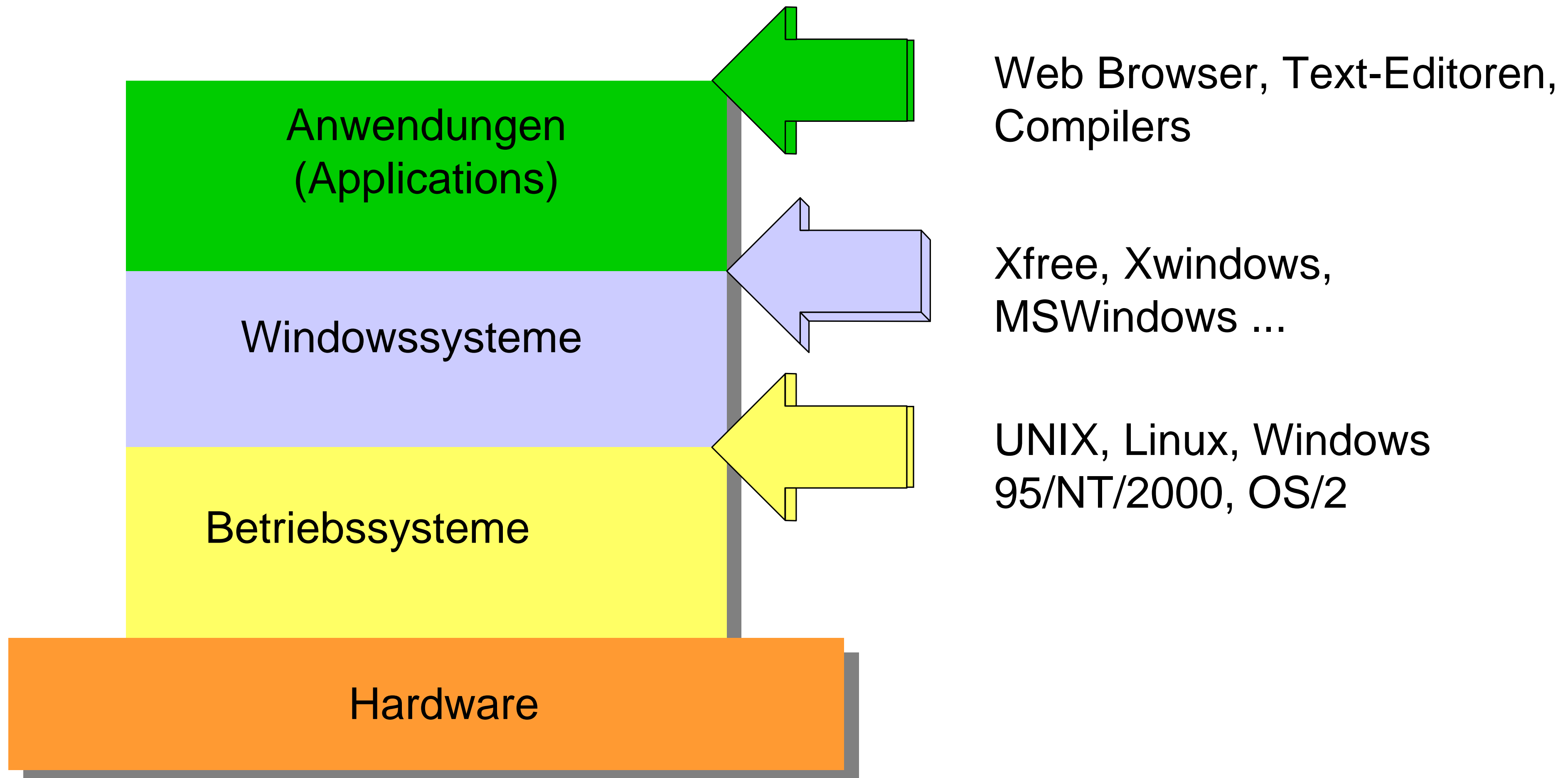
Programmierbeispiel mit Text



- Registersatz (file) zum schnellen Zugriff auf Daten
- Häufig gebrauchte Daten werden in Registern gehalten
- Versteht nur Maschinen-Code
- Häufig Instruktionen mit 3 Operanden
- **Instr <source1>, <source2>, <dest>**
- z. B.:**add %r2, %r3, %r2** (SPARC)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systemumgebung - Software



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programmieren

- Erzeugen einer Folge von Anweisungen (*Instruktionen*), die von der CPU eines Computers ausgeführt werden können
- Programm muss im Maschinencode vorliegen
- Es wird ebenfalls im Memory der Maschine gespeichert
- Es wird Instruktion für Instruktion aus dem Speicher genommen und von der CPU interpretiert
- CPU Register halten Programmadresse (pc) und Daten
- Wir können auf verschiedenen Abstraktionsebenen programmieren

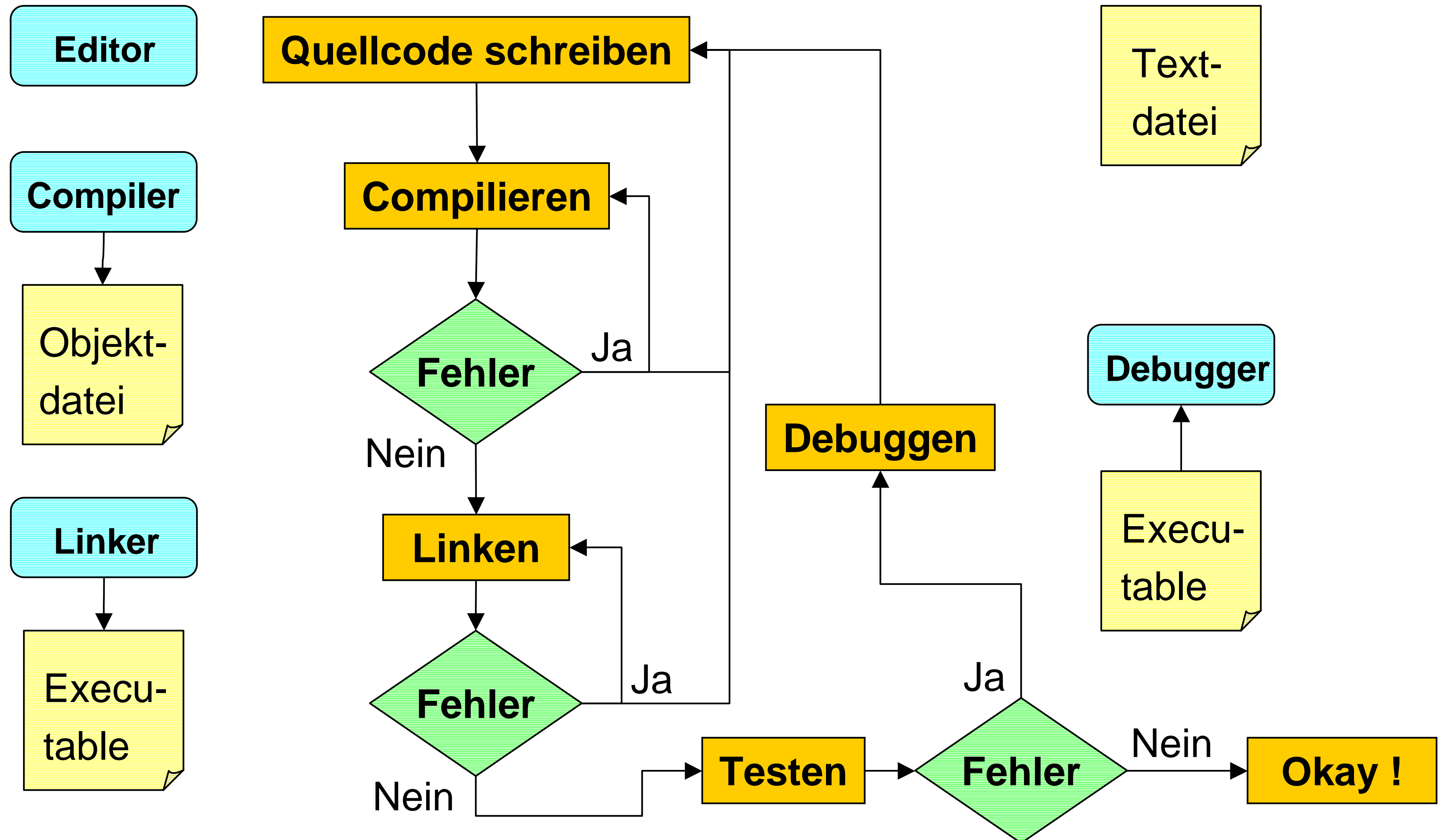
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programmieren

- Wir können auf verschiedenen Ebenen programmieren
  - Maschinen-Level („mit 0 und 1“)
  - Assembler-Level (`mov %r1, %r2`)
  - Hochsprachen-Level (`while (TRUE) fork();`)
- Programme werden heute hauptsächlich in höheren Programmiersprachen geschrieben
- Wir unterscheiden
  - Prozedurale: FORTRAN, COBOL, PASCAL, MODULA, C ...
  - Objektorientierte: C++, Java, Eiffel, SMALLTALK ...
  - Funktionsorientierte: LISP, …
  - Logische: PROLOG, ...

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programmieren



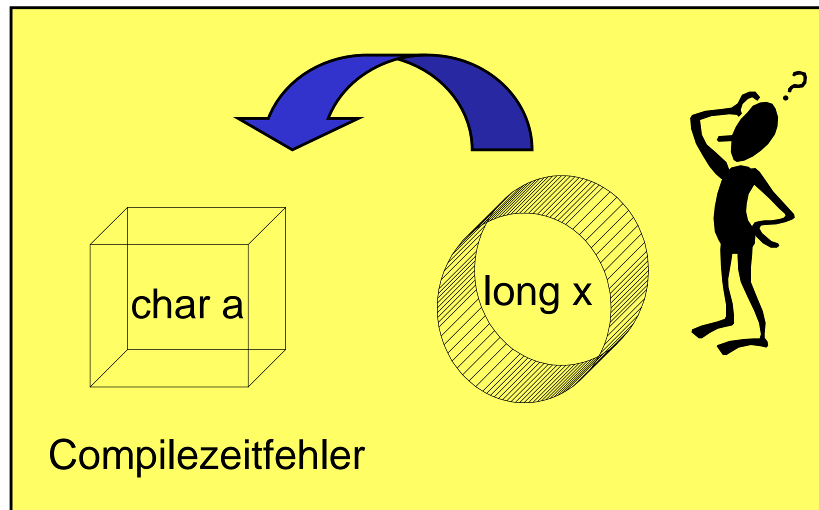
Programmierbeispiel mit Text

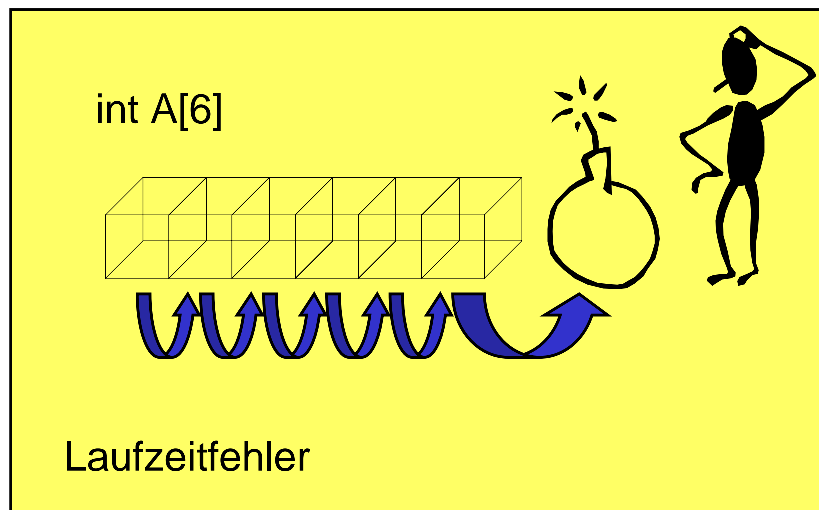
```
/* Testprogramm
  * Berechnung der Fläche eines
    Dreiecks
  */
#include <iostream.h>
int main() {
   int hoehe = 3;
   int grundseite = 5;
   double flaeche = hoehe * grundseite
     * 0.5;
   cout<<"Fläche:  "<<flaeche<<"\n";
   return 0;
}
```

- *Programmtext* (*source code*) besitzt einen formalen Aufbau, die Syntaxstruktur
- Syntax ist von Sprache zu Sprache verschieden
- Beispiel:
- `If (<Bedingung>) <Anweisung_1>; else <Anweisung_2>;`

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programmierbeispiel mit Text





- *Compilezeitfehler* sind Fehler, die der Compiler finden kann, wie z.B. Typumwandlungen:

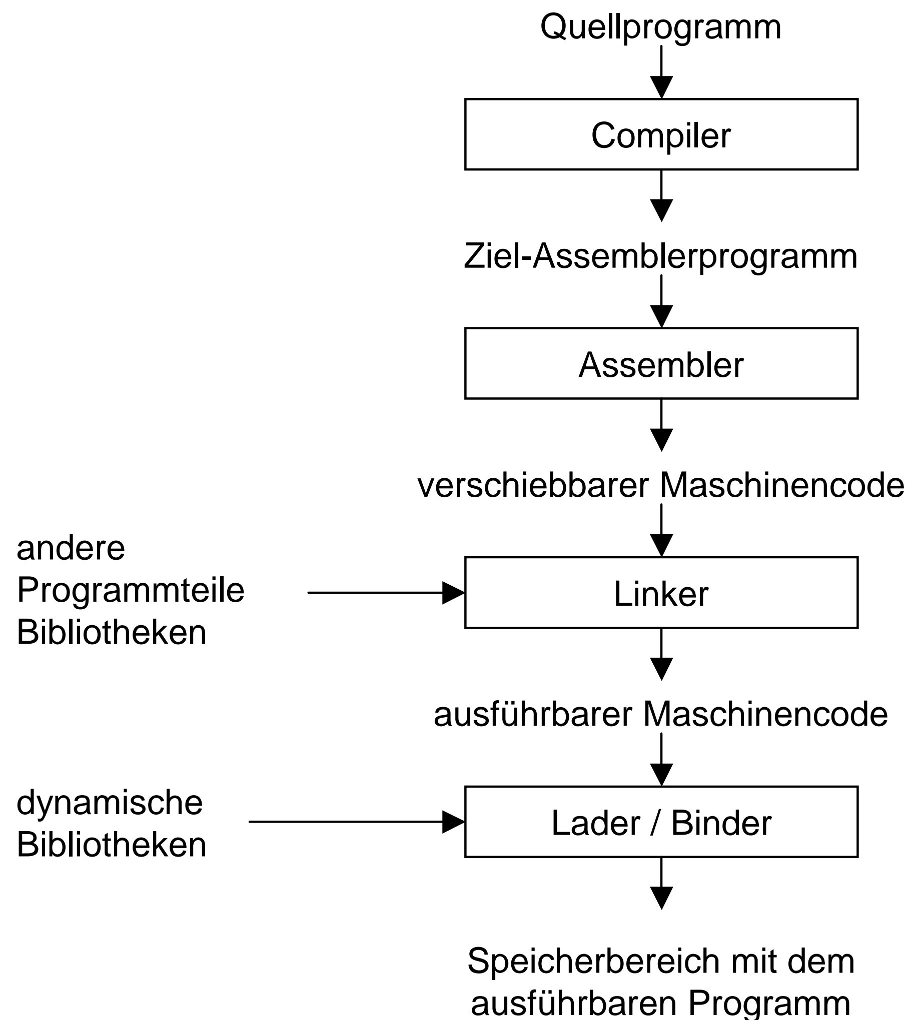
```
long x; /*eine ganze Zahl*/
char a; /* ein Zeichen */
a = x;
```

- *Laufzeitfehler* sind Fehler die erst erscheinen, wenn das Programm läuft, wie z.B. Division durch Null
- Laufzeitfehler sind nicht immer einfach zu finden !
- Testen kann keine Fehlerfreiheit beweisen

Programmierfehler

- Syntaxfehler
  - werden vom Compiler gemeldet
- Linkfehler
  - auf fehlende Objektdateien oder Bibliotheken zurückzuführen
- Laufzeitfehler: semantische/logische Fehler
  - werden nur durch Testen und Analysieren des lauffähigen Programms entdeckt
  - Laufzeitfehler sind „teuer“:
    - Wenig Unterstützung seitens des Rechners
    - Kosten für Benutzer des fehlerhaften Programms
  - Qualität und Umfang der Tests sind entscheidend!

Der Compiler



- Quellprogramm in Hochsprache muss in Maschinencode übersetzt werden
- Hierzu dienen *Interpreter* oder *Compiler*
- Interpreter übersetzen während der Programmausführung (z.B. UNIX shell-scripts)
- Compiler übersetzen das Programm vorab (MS Visual C++)

Der Compiler

- Compiler übersetzt den *Quellcode* (*source*) zunächst in Assembler-Code
- Im Speicher noch verschiebbarer Maschinencode
- Hat eine symbolische Repräsentation
- Assembler übersetzt *Assembler-Code* in *Maschinen-Code*
- *Linker* bindet noch notwendige Module mit an
  - Können vorab übersetzte Bibliotheken sein
  - Liegen bereits im Maschinen-Code vor
- *Loader* lädt den Maschinen-Code zur Ausführung in den Hauptspeicher
  - Kann zur Laufzeit weitere Module anbinden (dynamic link)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel - Assembler

- Assembler-Programm (Sparc):

```
// q.c
int max(i,j)
  int i,j;
{
  if (i<j)
     return j;
  else
     return i;
}
```

- C-Codefragment zur Berechnung des Maximums zweier Zahlen
- `> gcc -S q.c`

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel - Assembler

- Symbolischer Assembler-Code mit CPU-Instruktionen

```
      .file       "q.c"
      .global     max
      .type       max,#function
   max:
      mov %o0,%g2
      mov %o1,%o0
      cmp %g2,%o0
      bge,a .LL2
      mov %g2,%o0
   .LL2:
      retl
      nop
      .ident      "GCC: (GNU) 2.7.2"
```

Beispiel - Assembler

- `> gcc -o q q.s main.c`
- `> gdb q`

```
0x106b8 <max>:     0x84100008 mov   %o0, %g2
0x106bc <max+4>:   0x90100009 mov   %o1, %o0
0x106c0 <max+8>:   0x80a08008 cmp   %g2, %o0
0x106c4 <max+12>:  0x36800002 bge,a 0x106cc
0x106c8 <max+16>:  0x90100002 mov   %g2, %o0
0x106cc <max+20>:  0x81c3e008 retl
0x106d0 <max+24>:  0x01000000 nop
```

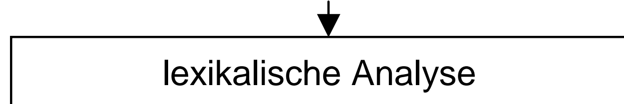
- Wir erhalten einen Ausdruck des Programmspeichers mit Maschinencode und *disassemblierten* Instruktionen

Copyright: M. Gross, ETHZ, 2003

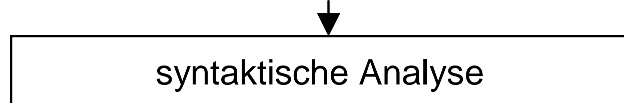
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

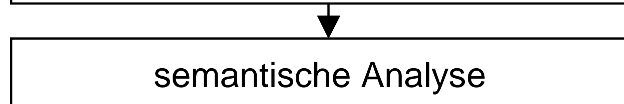
Der Compiler – nochmals im Detail

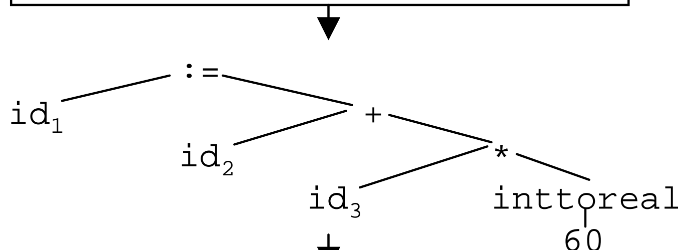
```
position := initial + rate * 60;
```

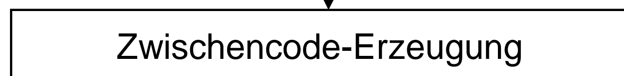


id_1 := id_2 + id_3 * 60







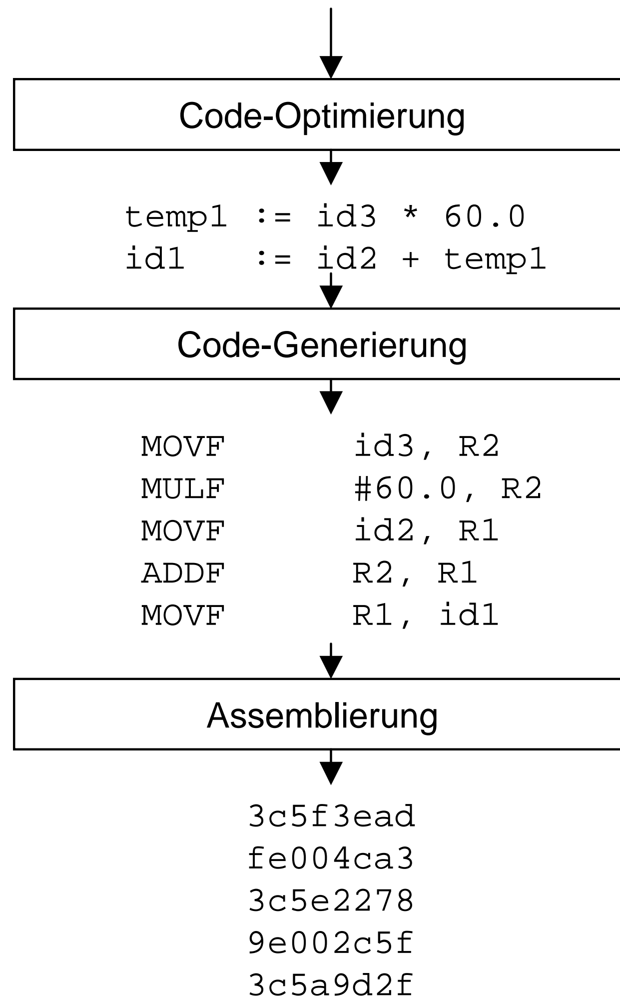



```
temp1  := inttoreal(60)
temp2  := id3 * temp1
temp3  := id2 + temp2
id1    := temp3
```

- *Lexikalische Analyse:* Namen von Zeichen werden ersetzt, Eindeutigkeit geprüft
- *Syntaktische Analyse:* Sprachsyntax wird geprüft
- *Semantische Analyse:* Positionen von Zeichen bestimmen den Sinn des Ausdruckes (rechts-links)
- *Zwischencode:* Oft wird ein maschinenunabhängiger Zwischencode erzeugt

Copyright: M. Gross, ETHZ, 2003

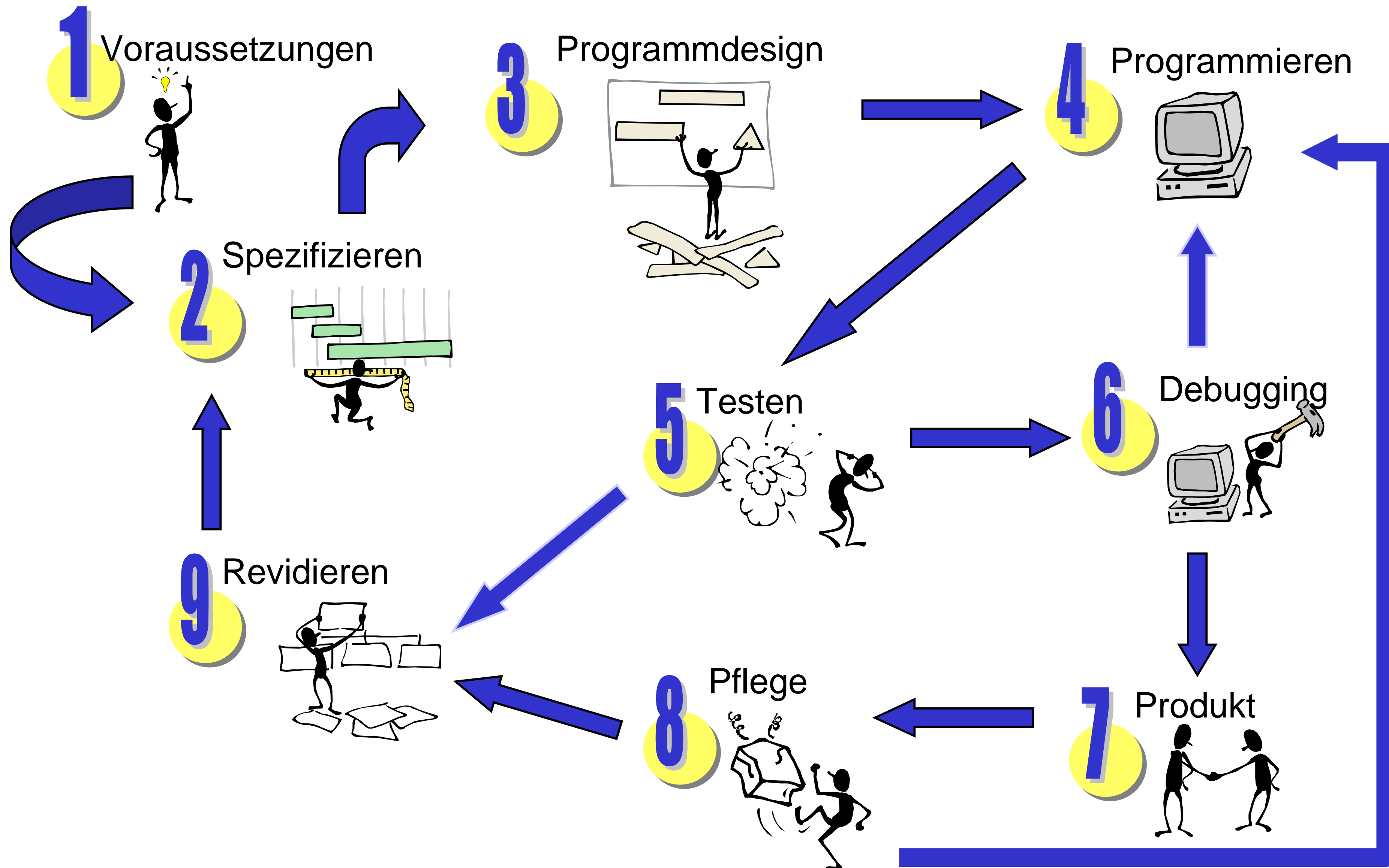
Der Compiler – nochmals im Detail



- *Code-Optimierungen:* Unnötiges Speichern Eliminieren, Umsortieren von Instruktionen, Register-Allokation
- *Code-Generierung:* Es wird Assembler-Code erzeugt (s- flag bei gcc)
- Maschinen-Code Erzeugung

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lebenszyklus eines Programmes



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Warum C, C++ ?

- C ist historisch eng mit der Entwicklung von UNIX verknüpft
- Ersetzt Assembler auf Systemprogrammierebene (I/O, Betriebssystem, HW-Treiber, Micorcontroller)
- Mehrzahl kommerzieller Anwendungspakete sowie Systemsoftware ist in C oder C++ geschrieben
- Als plattformunabhängige Sprache etabliert
- C++ ist mittlerweile die praktisch bedeutendste Programmiersprache
- C++ erlaubt sowohl prozedurale, als auch objektorientierte Programmierung
- C++ ist sehr mächtig – Obermenge von C

Historische Entwicklung

- BCPL (Richards 1967, typenlos)
- B (Thompson 1970, typenlos)
- C (Ritchie 1971, typisiert)
- ANSI C (1983-1987, Standardisierung)
- C++ (Stroustrup 1986, Klassen und OO)
- 1990 - 1998 Final Standard ISO/ANSI
- C++ entwickelt sich noch immer

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Der Prozessor