Programmierkurs: „hello, world“

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 25/26

Rückblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine

VL 1 „Hello World“: „Lebenswichtiges“, Programablauf, Programmierablauf, Kompilierung und Ausführung von Programmen

VL 2 „Die ersten Schritte“: Erstes C-Programm, Elementare C-Strukturen, Datentypen, Operatoren, Schleifen

VL 3 „Kontrollstrukturen & Funktionen“: Syntax, Semantik, bedingte Anweisungen, Blöcke, Sichtbarkeit

VL 4 „Rekursive Funktionen & Bibliotheken“: rekursive Funktionsaufrufe, Modularisierung

VL 5 „Typen“: Einfache und strukturierte Datentypen, Wertebereiche, Typendefinition

VL 6 „Speicher und Adressen“: Speicher, Pointer, Funktionsaufrufe „call by value“ vs. „call by reference“

VL 7 „Speicher und Arrays“: Speicher, Arrays, mehrdimensionale Arrays, Arrays und Pointer

VL 8 „Dynamische Speicherverwaltung“: Speicherallokation, Fehlerbehandlung, Rückgabewerte, Arrays/Pointer/Adressen

VL 9 „Strings, Kanäle, Git“: Strings und Arrays, Zeichensätze, Stringlänge, Ein- und Ausgabe, Arbeiten mit git

VL 10 „Debugging und Stack“: Fehlverhalten/Bugs, Fehlersuche Strategien und Werkzeuge

Überblick

- Was tun Computer?
- Wie interagiere ich mit einem Computer?
- Was sind (Text-) Dateien?
- Das 1. Programm: „hello, world“

Computer …

… tun nicht viel

- Informationen speichern (Bilder, Videos, Texte, Chats, …)
- Anweisungen **exakt** befolgen

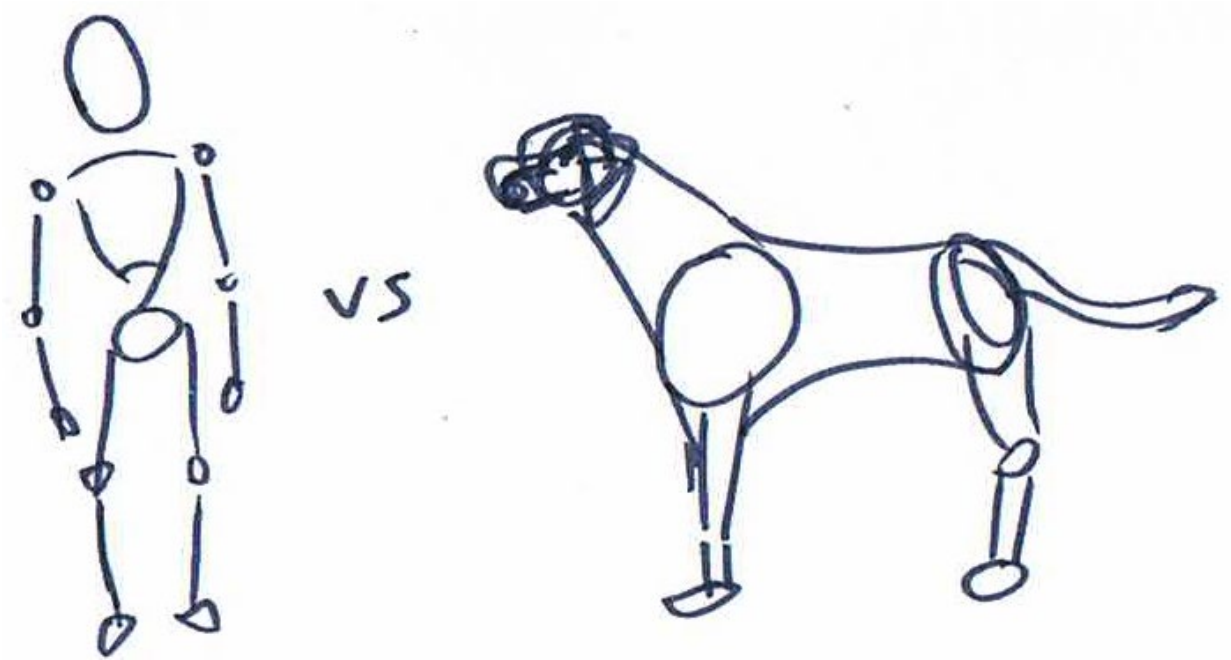
Informatiker:innen

… tun etwas mehr

- Informationen speichern (Bilder, Videos, Texte, Chats, …)
  - Definieren wie Daten aussehen und gespeichert werden
- Anweisungen **exakt** befolgen
  - Geben **exakte (einfache und präzise) Anweisungen**, was mit diesen Daten gemacht werden soll

Wie erkläre ich einem Computer, was er machen soll?

Kommunikation



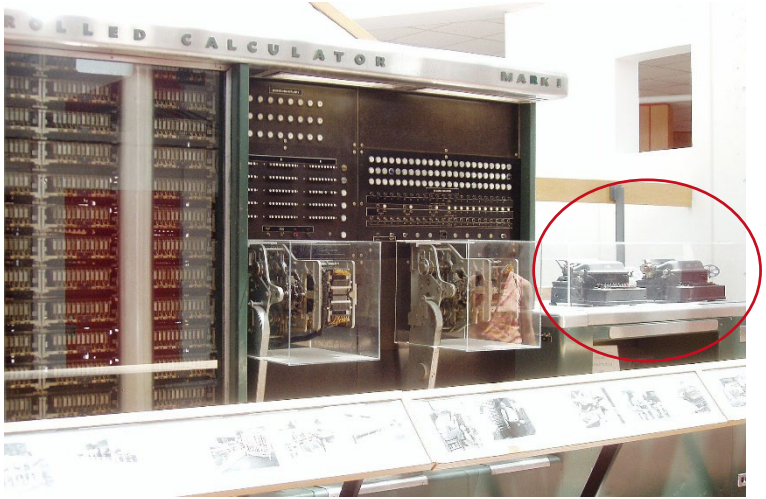
Wie kommuniziere ich nun?

- Gesprochene Sprache
  - Alexa, Siri, …
- Video
  - Gestenerkennung, Face ID, …

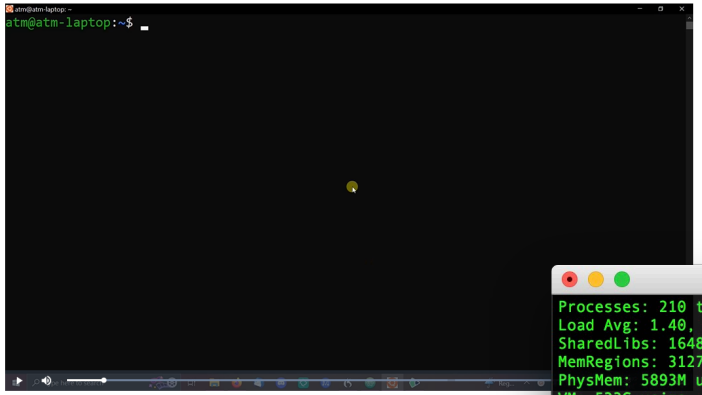
Erfordert **sehr komplexe Verarbeitung**: Das muss jemand programmiert haben, bevor es zur Kommunikation genutzt werden konnte. Also wie ging das?

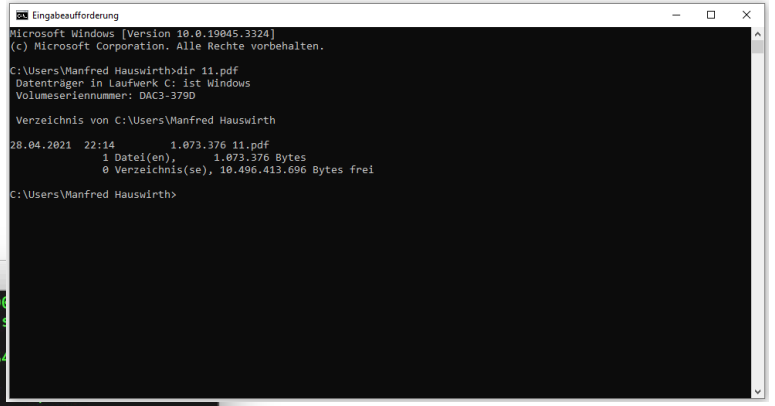
➔ Text: Eingabe per „Terminal“

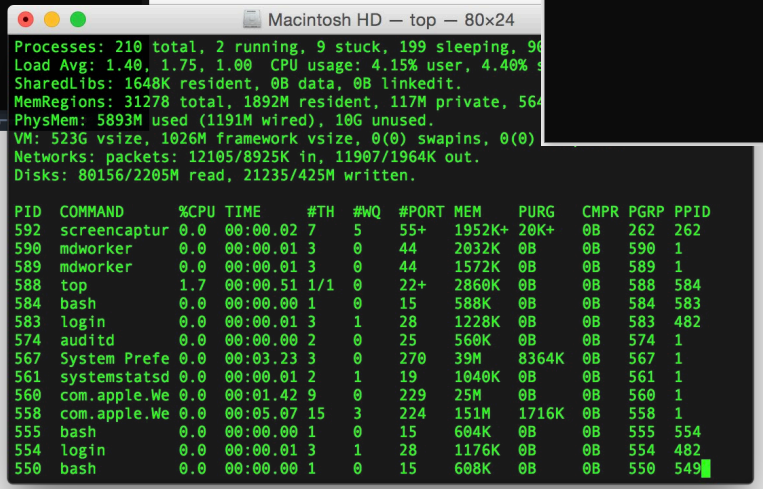
Terminals damals …



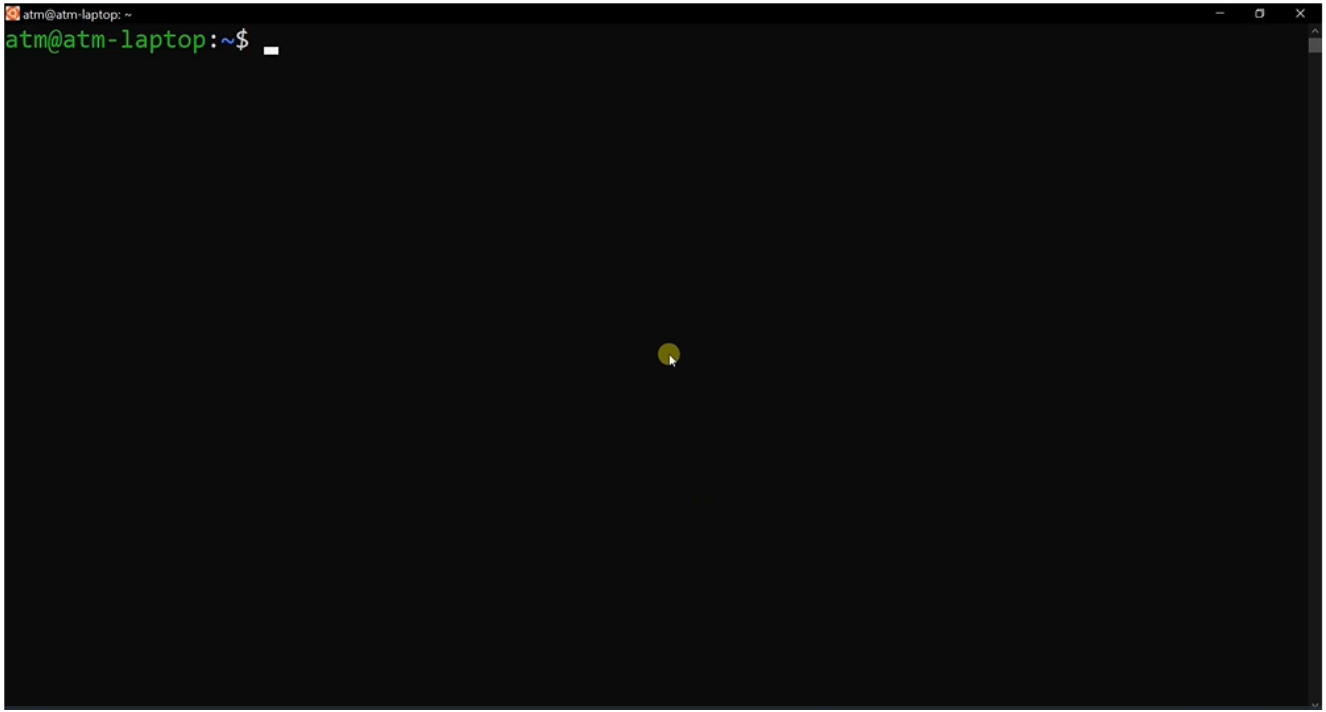
Terminals heute …







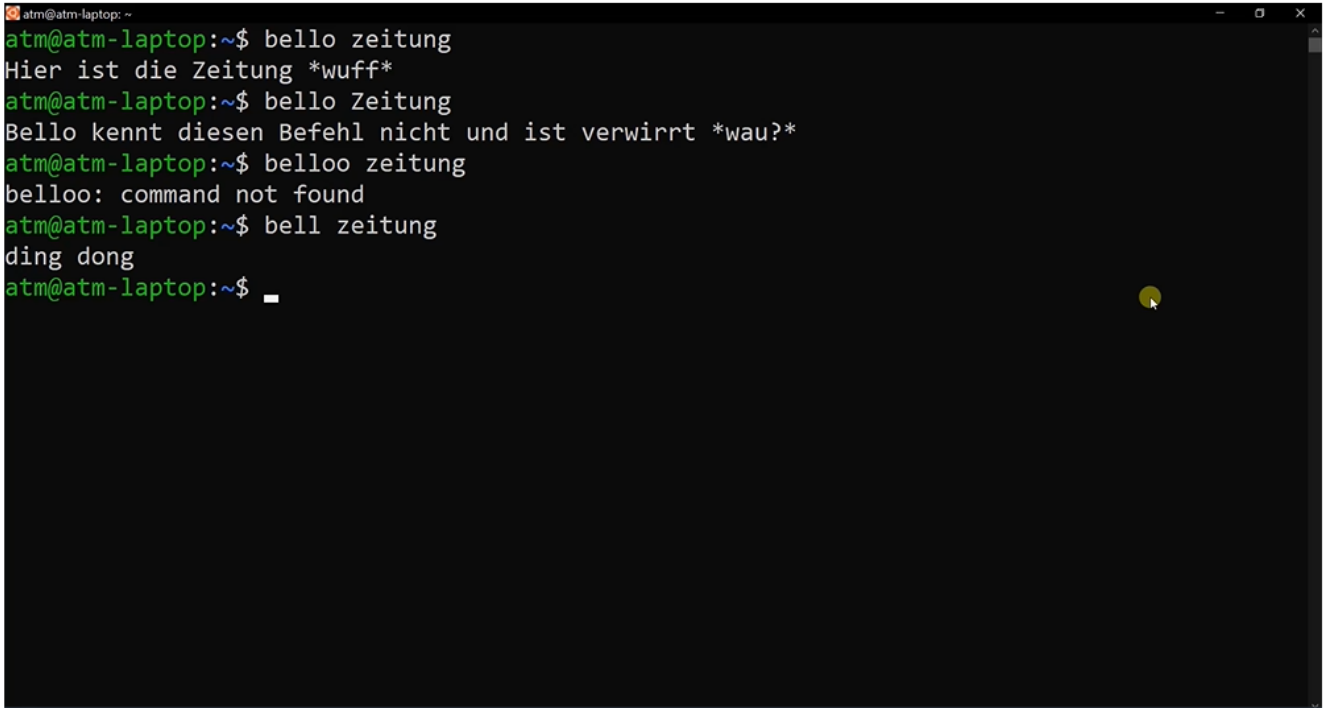
Beispiel: „bello“



Beispiel: „bello“

Exakte(!) Anweisungen!

Exakte(!) Anweisungen!



Hilfe bei Computerproblemen

- Was will ich erreichen?
  - vom **`bello`** Programm die Zeitung holen lassen
- – Was habe ich getan?
  - im Terminal **`bell zeitung`** ausgeführt
- – Was habe ich erwartet?
  - eine Ausgabe, die das Holen der Zeitung bestätigt
- – Was ist tatsächlich passiert?
  - Ausgabe **`ding dong`**

Das Terminal

- terminal
- command-line (Kommandozeile)
- command prompt (Eingabeaufforderung)
- shell
- bash
- tty
- terminal emulator

Wie öffne ich ein Terminal? Siehe ISIS-Kurs.

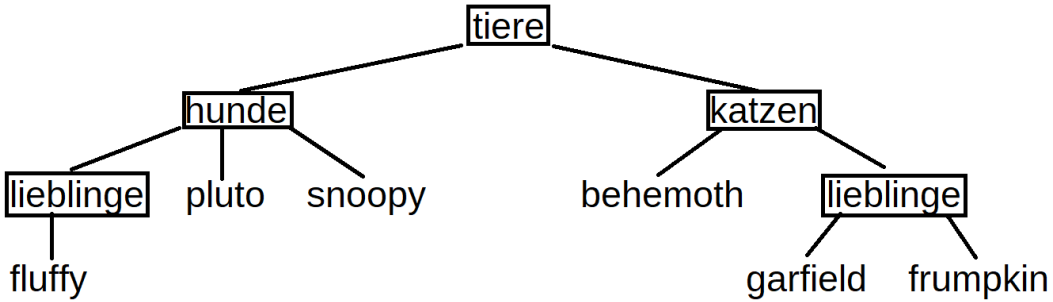
⇨ Video

Daten

- Daten werden in Dateien gespeichert
- Dateien
  - beinhalten eine Sequenz aus Nullen und Einsen
  - haben keine inhärente Bedeutung
  - Interpretation erfolgt durch Programme
- Die wichtigsten zwei Dateitypen für Intoprog:
  - Maschinencode: Die „Sprache“ des Prozessors (CPU: Central Processing Unit)
  - Plaintext: Das vom Menschen geschriebene Programm („Source Code“, „Quellcode“)

Dateien

- Haben einen Namen
- Können in Verzeichnissen (Folder) organisiert werden



Textdateien

Text **ohne Formatierung**, reine Aneinanderreihung von Symbolen („plain text“ ➔ Plaintext)

- Texteditoren:
  - traditionell:
    - vim (neovim)
    - Emacs
  - modern:
    - sublime text
    - visual studio code
  - dutzende Alternativen
- Microsoft Word und Open Office sind **keine** Texteditoren

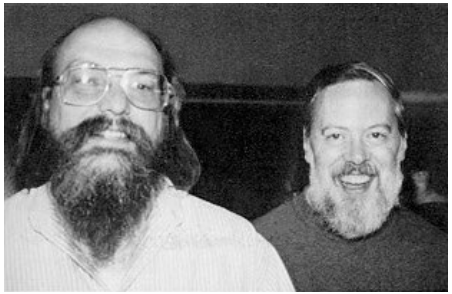
Source-Code (Quellcode)

- Plaintext
- Definiert in einer Programmiersprache, was der Computer machen soll
- Programmiersprache
  - Eine (einfache) Kunst-Sprache mit
    - Syntax (welche Worte gibt es)
    - Grammatik (Regeln, um aus Worten Sätze zu formen)
    - Semantik (was bedeuten die Worte und Sätze, d.h. was „tun“ sie)

Programmiersprachen

- … gibt es wie Sand am Meer …
  - Es gibt nicht „die“ Programmiersprache oder die „beste“ Programmiersprache
    - Hängt ab von Umfeld, Anwendung, Problemstellung, Vorlieben, Toolkits, etc.
  - C, C++, Pascal, BASIC, Lisp, Prolog, Haskel, Java, Scheme, Python, JavaScript, COBOL, FORTRAN, etc., etc.

Wir verwenden „C“



- Entwickelt 1969-1973 von Dennis Ritchie bei Bell Labs
- ANSI C Standard 1989 ratifiziert durch American National Standards Institute
- C99 und C11: kleinere Erweiterungen zu ANSI C
- „K&R“ (Brian Kernighan, Dennis Ritchie) über C:
  C ist „quirky, flawed, and an enormous success“
  - C und Unix sind eng miteinander verbunden
  - C ist eine kleine und einfache Sprache
  - C ist für praktische Zwecke entworfen worden
  - C ist die Sprache für systemnahe Programmierung

Warum gerade „C“ ?

- Extrem weit verbreitet, etabliert – z.B. sind in C programmiert

  Windows, Linux, MacOS, Android, iOS, Oracle, MySQL, MS SQL Server, Web Server, Embedded Systems, Internet of Things, etc., etc., etc.
- auf allen Plattformen verfügbar
- Grundlage für viele weitere Vorlesungen, u.a. Rechnerorganisation
- C-Syntax Grundlage für Java, JavaScript, Python, etc.

Programmbeispiel

„Hello World“ Programm aus
„The C programming language“ Kernighan, Ritchie

```
1 #include <stdio.h>
2
3 int main() {
4   printf (“hello, world\n“);
5 }
```

Quelltext zu Programm

„Hello World“ Programm aus
„The C programming language“ Kernighan, Ritchie

```
1 #include <stdio.h>
2
3 int main() {
4   printf (“hello, world\n“);
5 }
```

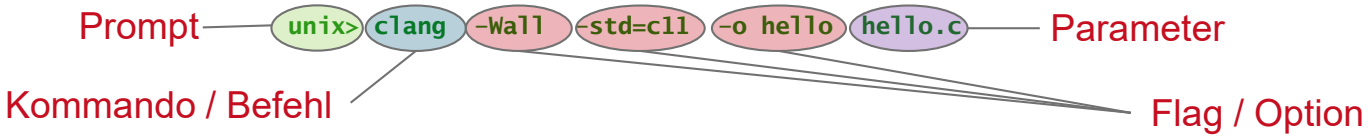
Frage: Wie kommt man vom Quelltext (Source Code) zum ausführbaren Programm?

Compiler / Übersetzer

- Übersetzt den Quell-Code
  - hello.c
  - Menschen-lesbar
- In Maschinen-Code
  - hello.o
  - Maschinen-lesbar (die CPU versteht, was sie machen soll)
- Ein Compiler ist selbst wieder ein Programm

C-Compiler

- Beispiel: clang



- Beispiel: gcc

```
unix> gcc -Wall  -std=c11 -o hello hello.c
```

  - Benennen der Ausgabedatei: Compilerflag –o <name>
    (Ansonsten wird die Datei a.out generiert)
  - Wir benutzen den Programmierstandard C11
    (Compilerflag: -std=c11)
  - Wir empfehlen mit Warnungen zu kompilieren
    (Compilerflag: -Wall)

Was passiert hinter den Kulissen?

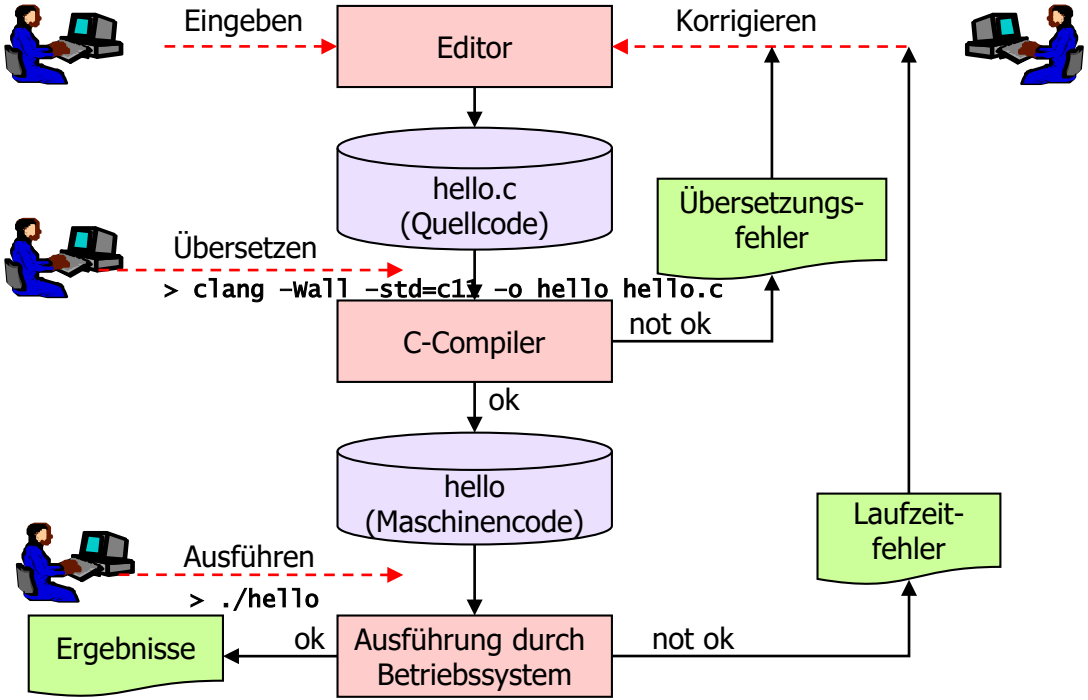
- Beispiel: clang

```
unix> clang -Wall -std=c11 -o hello hello.c
```

• Compiler	Übersetzt C (Hochsprache) in Assemblercode (simple Sprache, immer noch „Text“)
• Assembler	Übersetzt Assemblercode in Maschinensprache (Nullen und Einsen)

- Für Interessierte: Backup-Slides am Ende (wird nicht geprüft)

Ablauf des Programmierens



Ausführung eines Programms

- Programme werden von anderen Programmen gestartet
- „hello, world“ Programm:
  - Anfang: Ausführbares Programm
  - Ziel: Ergebnis des Programms
  - Mittel: Spezielle Anwendung: shell (terminal)
    - Kommandozeileninterpreter
      - Druckt Eingabeaufforderung („Prompt“)
      - Wartet auf Eingabe einer Kommandozeile
      - Führt Kommando aus
        (Annahme: 1. Wort: Shell-Kommando oder ausführbares Programm)

```
unix> ./hello
hello, world
unix>
```

Ausblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Hello World“: „Lebenswichtiges“, Programablauf, Programmierablauf, Kompilierung und Ausführung von Programmen
VL 2 „Die ersten Schritte“: Erstes C-Programm, Elementare C-Strukturen, Datentypen, Operatoren, Schleifen
VL 3 „Kontrollstrukturen & Funktionen“: Syntax, Semantik, bedingte Anweisungen, Blöcke, Sichtbarkeit
VL 4 „Rekursive Funktionen & Bibliotheken“: rekursive Funktionsaufrufe, Modularisierung
VL 5 „Typen“: Einfache und strukturierte Datentypen, Wertebereiche, Typendefinition
VL 6 „Speicher und Adressen“: Speicher, Pointer, Funktionsaufrufe „call by value“ vs. „call by reference“
VL 7 „Speicher und Arrays“: Speicher, Arrays, mehrdimensionale Arrays, Arrays und Pointer
VL 8 „Dynamische Speicherverwaltung“: Speicherallokation, Fehlerbehandlung, Rückgabewerte, Arrays/Pointer/Adressen
VL 9 „Strings, Kanäle, Git“: Strings und Arrays, Zeichensätze, Stringlänge, Ein- und Ausgabe, Arbeiten mit git
VL 10 „Debugging und Stack“: Fehlverhalten/Bugs, Fehlersuche Strategien und Werkzeuge

Slides für Interessierte

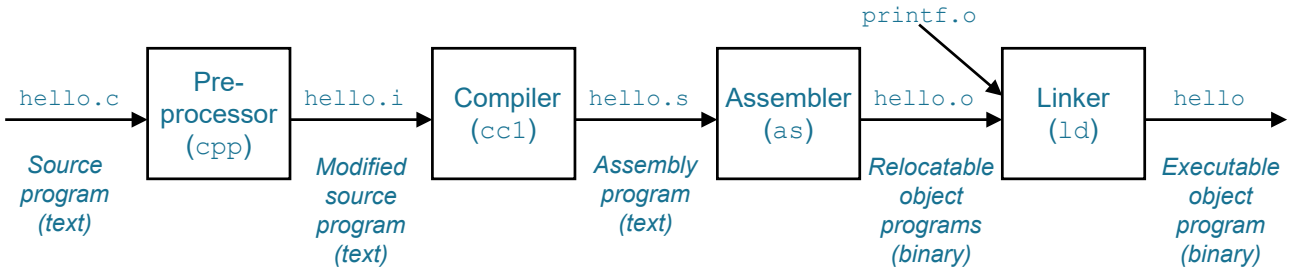
Überblick: C-Compiler

- Beispiel: GCC – GNU Compiler Collection

```
unix> gcc -Wall -std=c11 -o hello hello.c
```

4 Phasen:

• Preprocessor	Aufbereitung
• Compiler	Übersetzt C in Assemblercode
• Assembler	Übersetzt Assemblercode in Maschinensprache
• Linker	Nachbearbeitung / Kombination verschiedener Module



Assemblercode: CPU-abhängig

```
section .data
  hello_world db 'Hallo Welt!', 0x0a
  hello_world_len equ $ - hello_world
section .text
  align 4
  sys:
    int 0x80
    ret
  global _start
  _start:
    push hello_world_len
    push hello_world
    push 1
    mov eax, 4
    call sys
    push 0
    mov eax, 1
    call sys
```

```
# Kompilieren mit "gcc -nostdlib -s hallo.s"
    .section    .rodata
    .align 2
.s:
    .string "Hallo Welt!\n"

    .section    ".text"
    .align 2
    .globl _start
_start:
    li 0,4          # SYS_write
    li 3,1          # fd = 1 (stdout)
    lis 4,.s@ha     # buf = .s
    la 4,.s@l(4)
    li 5,12         # len = 12
    sc              # syscall

    li 0,1          # SYS_exit
    li 3,0          # returncode = 0
    sc              # syscall
```

```
.text

.global _start

_start:

        /* print */
        mov     r2, $11
        ldr     r1, =hw
        mov     r0, $1
        mov     r7, $4
        svc     $0

        /* exit */
        sub     r0, r0, r0
        mov     r7, $1
        svc     $0

.data

hw:
.ascii "Hallo Welt\n"
```

```
        ld hl,text              ; address of text into hl
loop:   ld a,(hl)               ; load akku by content at address in hl
        cp 0                    ; compare to zero
        ret z                   ; return if zero
        call &bb5a              ; otherwise print char in akku
        inc hl                  ; hl=hl+1
        jr loop                 ; relative jump to loop
text:   db "Hello World!",0     ; text with 0 as end mark
```

C-Code => Assembler-Code => Maschinencode

Maschinencode: CPU-abhängig

```
$ objdump -d factorial

factorial:     file format elf64-x86-64

Disassembly of section .init:

00000000004003f0 :
  4003f0:   48 83 ec 08          sub    $0x8,%rsp
  4003f4:   e8 73 00 00 00       callq  40046c
  ..
Disassembly of section .plt:

0000000000400408 :
  400408:   ff 35 e2 0b 20 00    pushq  0x200be2(%rip)        # 600ff0
  40040e:   ff 25 e4 0b 20 00    jmpq   *0x200be4(%rip)        # 600ff8
  400414:   0f 1f 40 00          nopl   0x0(%rax)
```

Maschinencode

Assemblercode

[stackoverflow, David Hoelzer]

C-Code => Assembler-Code => Maschinencode

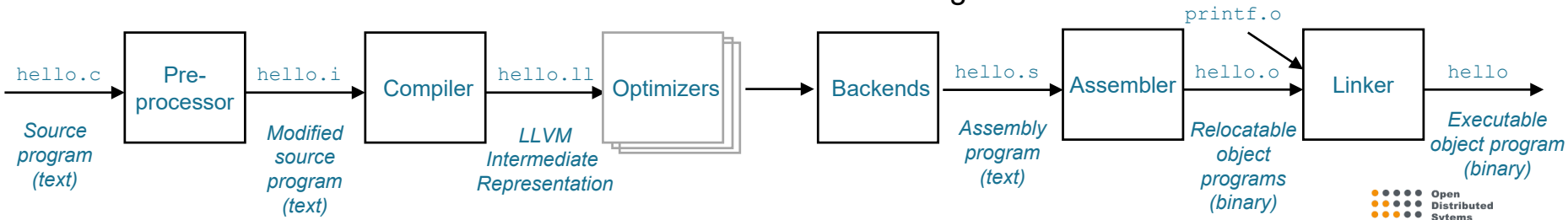
Überblick: C-Compiler

- Beispiel: Clang - a C language family frontend for LLVM

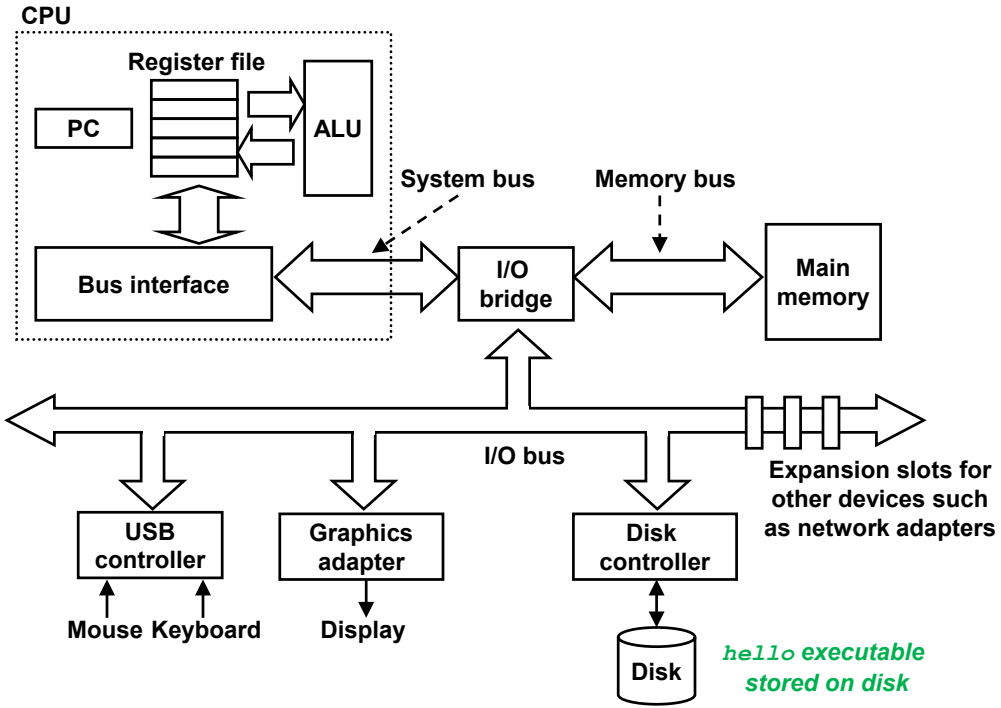
```
unix> clang -Wall -std=c11 -o hello hello.c
```

5+ Phasen:

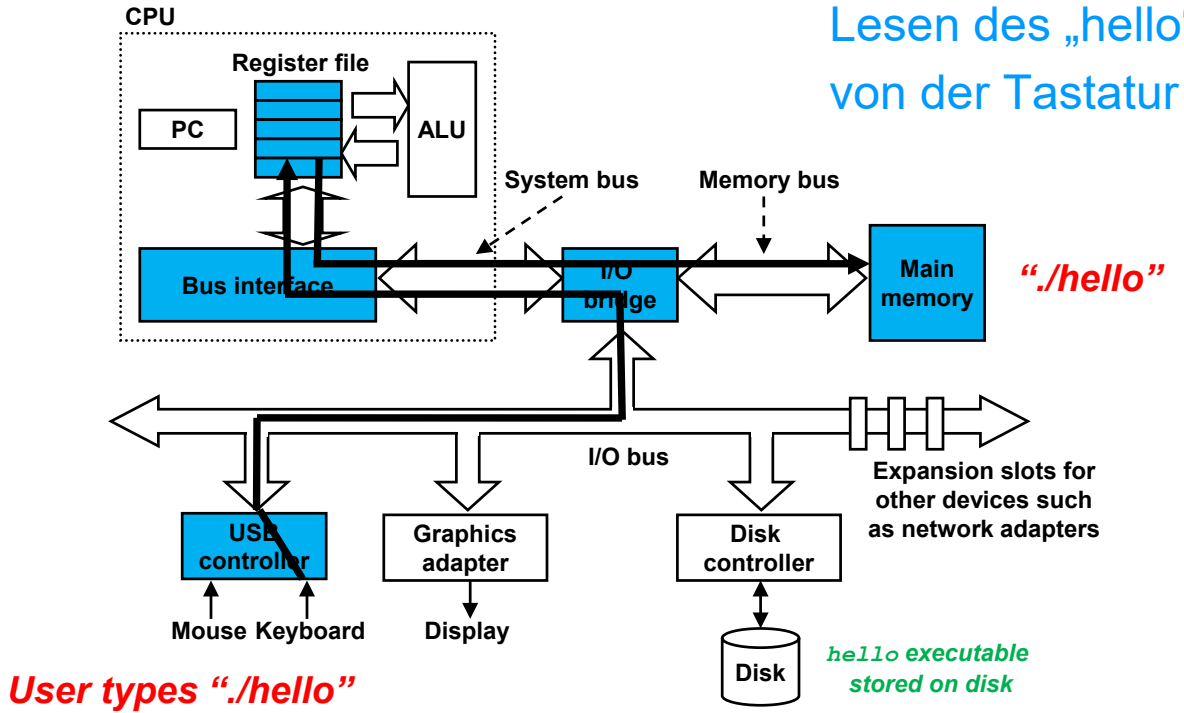
- Preprocessor — Aufbereitung
- Compiler — Übersetzt C in LLVM Intermediate Representation (IR)
- Optimizers — Optimieren der sprachunabhängigen LLVM IR
- Backend — Übersetzt LLVM IR in Assemblercode
- Assembler — Übersetzt Assemblercode in Maschinensprache
- Linker — Nachbearbeitung / Kombination verschiedener Module



Ausflug: Die Rechnerhardware



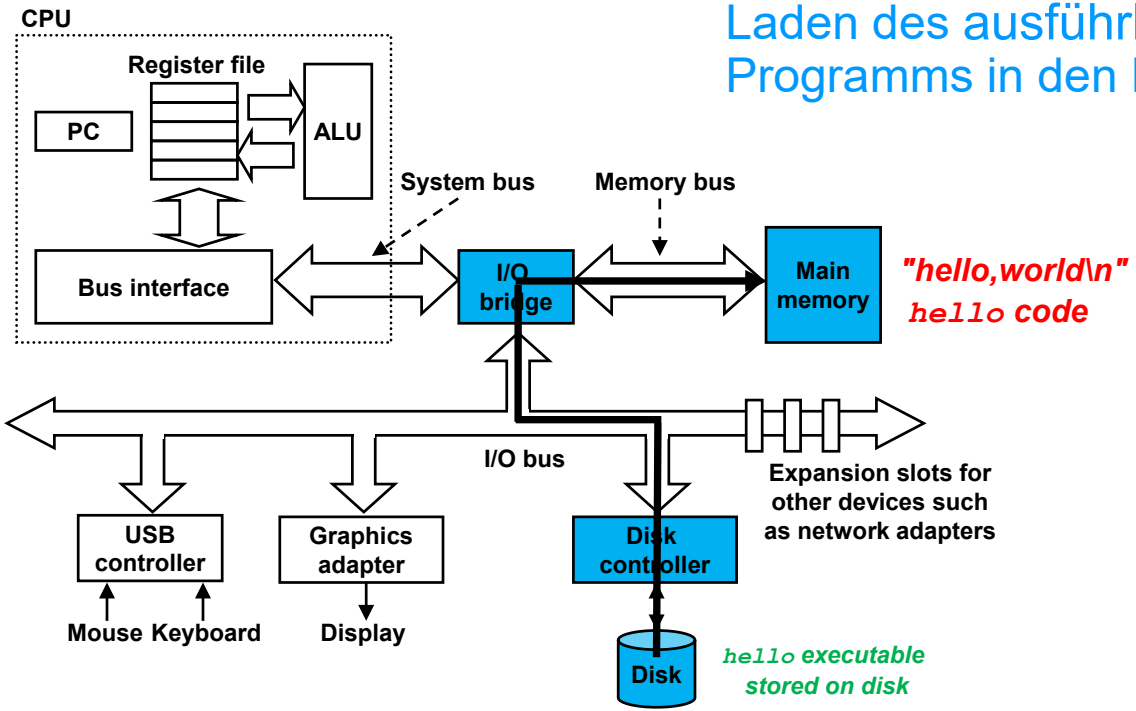
Ablauf: Ausführung



Lesen des „hello“ Kommandos von der Tastatur

Ablauf: Ausführung

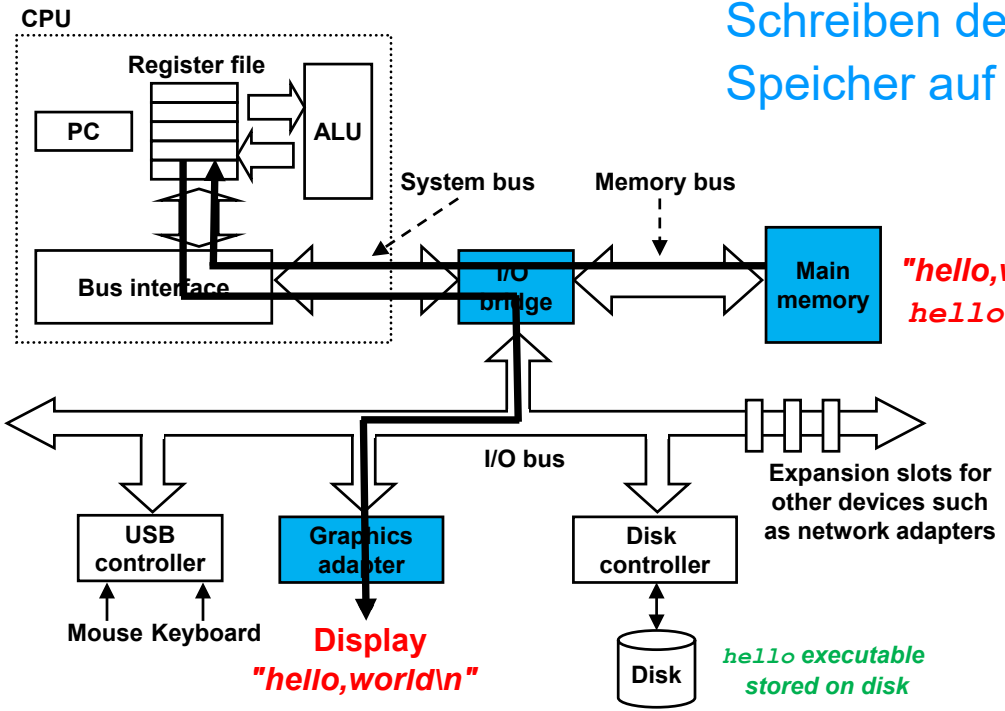
Technische Universität Berlin



Laden des ausführbaren Programms in den Hauptspeicher

Open Distributed Sytems ODS

Ablauf: Ausführung



Schreiben der Ausgabe vom Speicher auf das Display