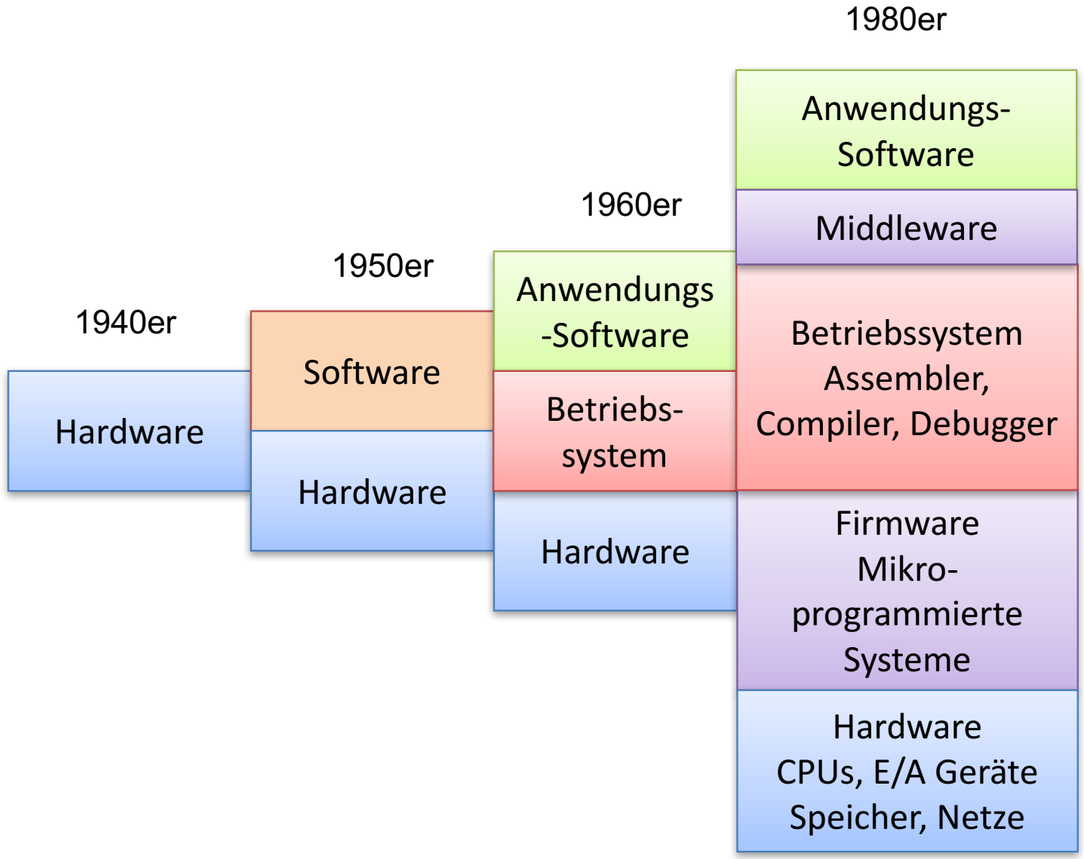
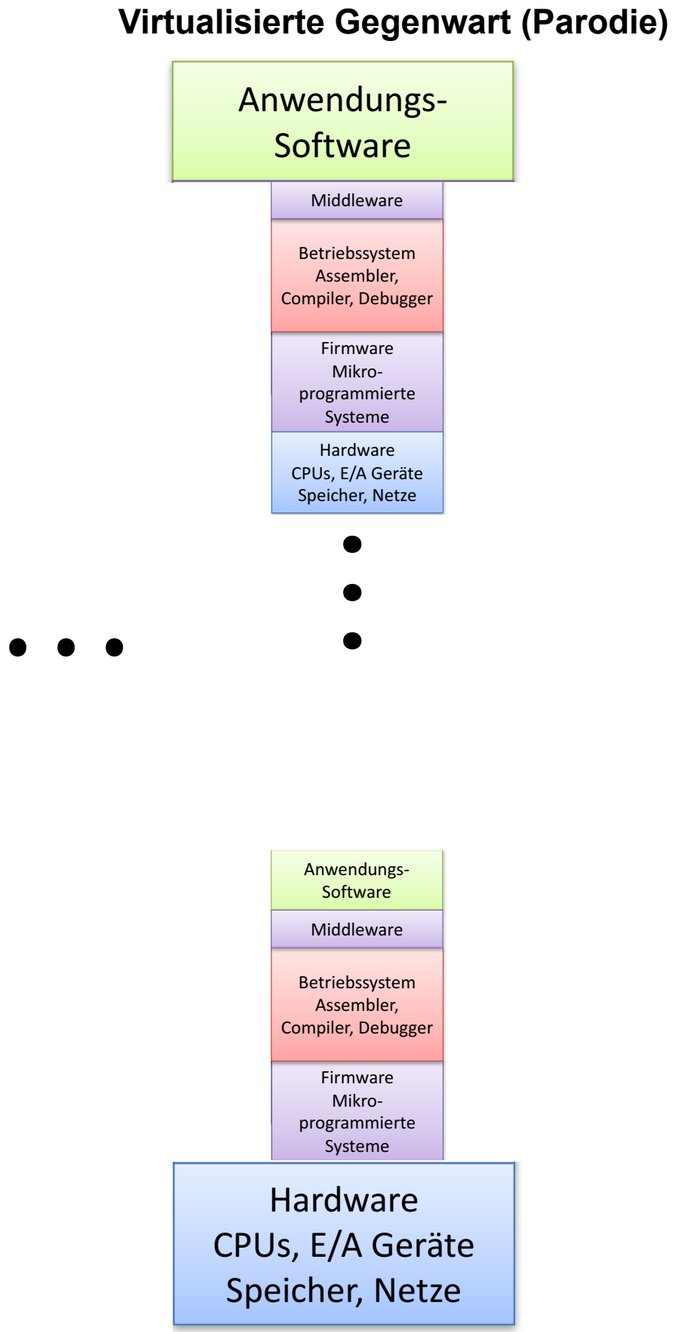
PT1

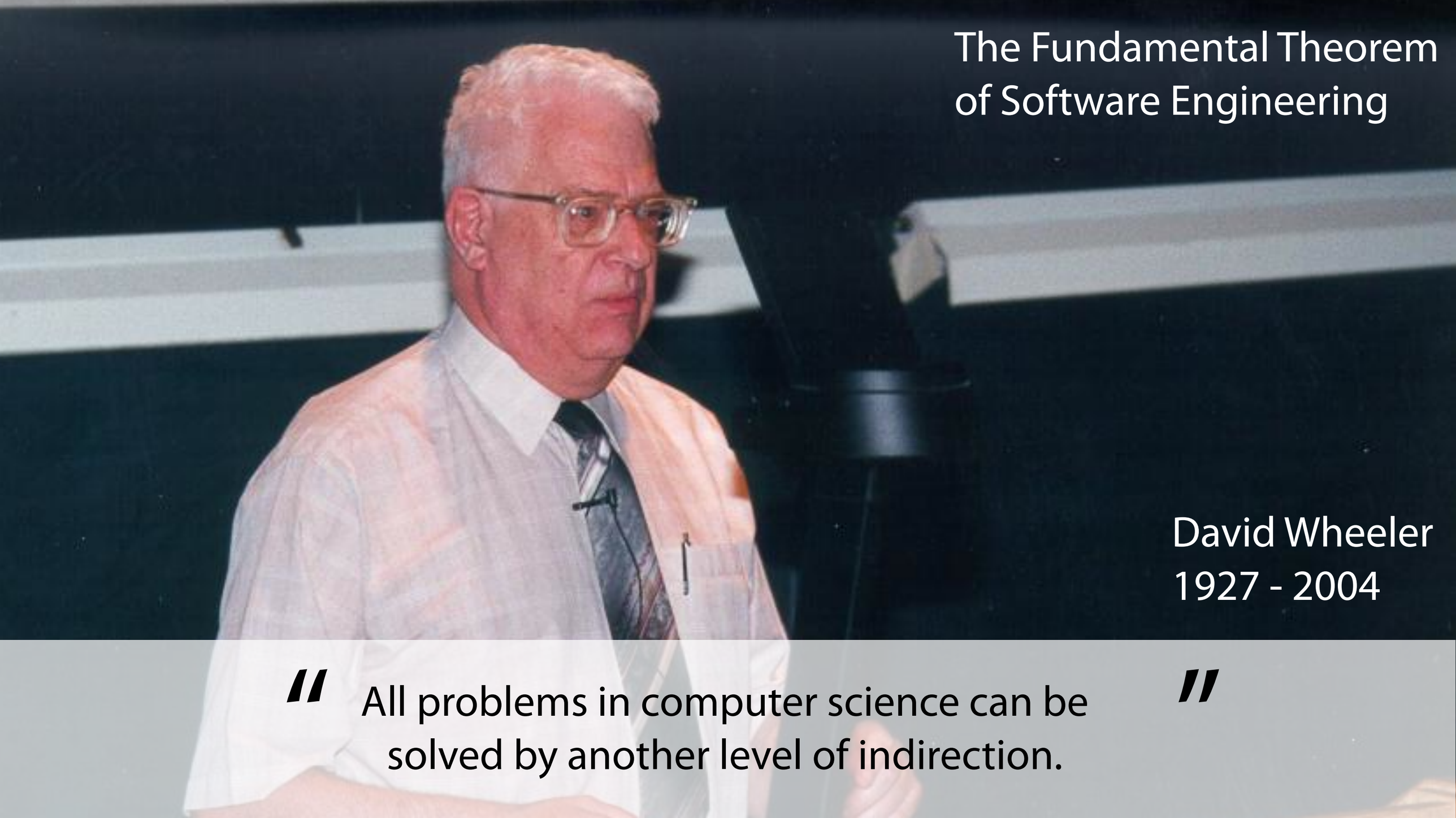
Unit 05 (cont.): Werkzeuge

Sven Köhler
Hasso-Plattner-Institut
2017-12-05





Softwareevolution



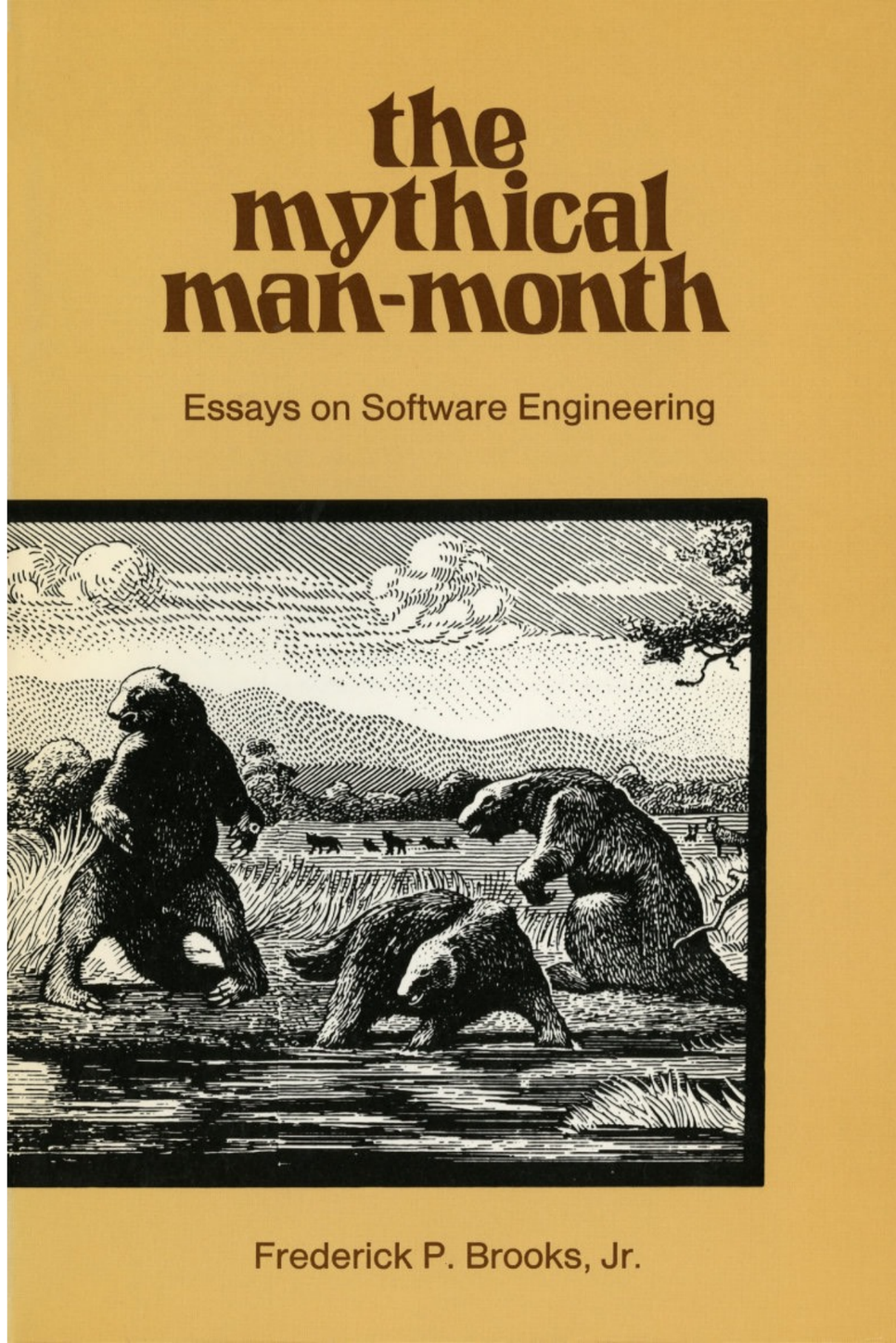
The Fundamental Theorem of Software Engineering

David Wheeler
1927 - 2004

> "All problems in computer science can be solved by another level of indirection."

- Sammelt Erfahrungen aus der Entwicklung von IBM OS/360 während der 1960er
- OS/360 verzögerte sich, entwickelte zu hohe Hardwareanforderungen
- Zusätzliche Programmierer sorgen für weitere Verzögerung:
  - Müssen erst antrainiert werden (Lernkurve)
  - Erhöhter Verwaltungsoverhead
  - Spezifikation von Schnittstellen, Tests und Modulgrenzen nötig
  - Fehlersuche wird mit Projektgröße komplexer

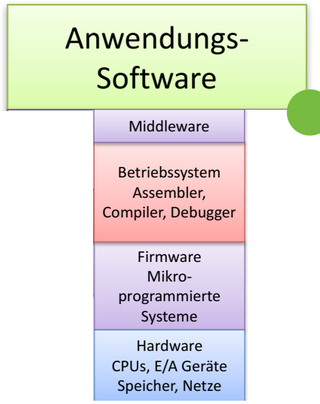
Die **Softwarekrise** führte zur **Softwaretechnik**.



Softwaretechnik (Software Engineering) ::

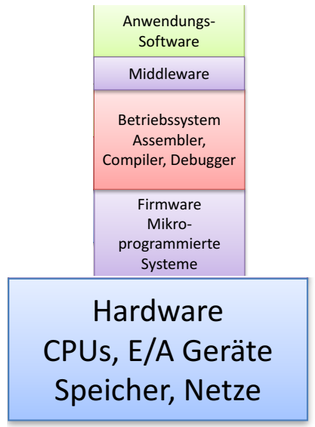
Menge verschiedener Teilgebiete, die das Ingenieurwesen
auf die Entwicklung komplexer Softwaresysteme anwenden.

Umfasst z.B. die Entwicklung von Hochsprachen, Objektorientiere Programmierung, Planungs- und Analyseprozesse, Softwarearchitektur, Validierung und Verifikation (Testen), Projektmanagement, Entwicklungs-prozesse, Dokumentationen, Nutzung von Werkzeugen und Bibliotheken.




```c
#include <stdio.h>

int main()
{
  printf("Hi!\n");
  return 0;
}
```



Jeder Quellcode ist Teil eines größeren Kontext.

Sowohl in der Ausführung,
als auch in zeitlicher Entwicklung und sich ändernden Teams.

PT1

Unit 05 (cont.): Werkzeuge

Sven Köhler
Hasso-Plattner-Institut
2017-12-05

Inhalt:

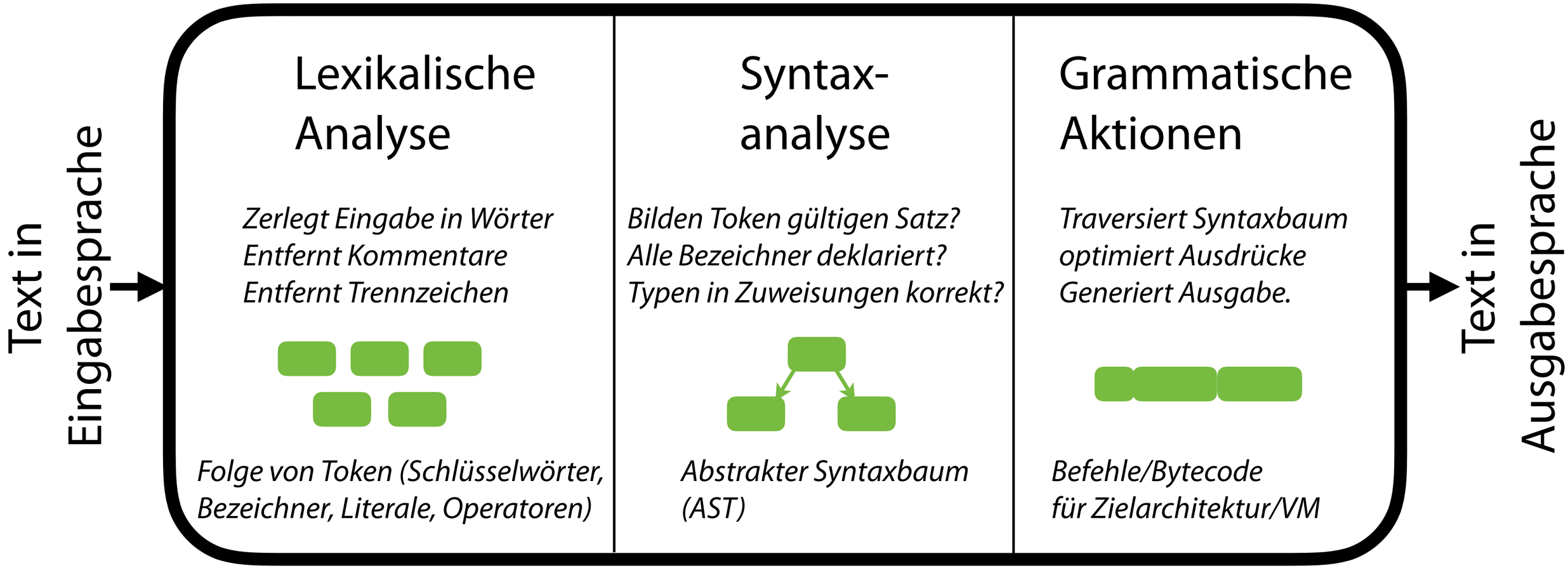
- Kommandozeileninterpreter
- Build-Managementprogramme (make)
- Versionsverwaltung (git)
- Compiler
- Debugger (gdb)
- Test (CUnit)

[https://xkcd.com/303/]

1 Compiler

Compiler ::

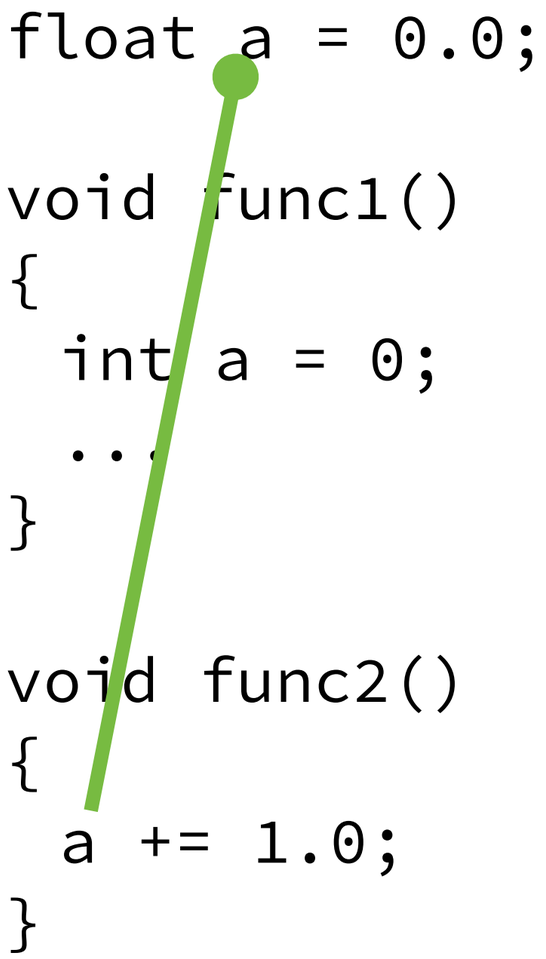
überführt Text einer Eingabesprache in eine Ausgabesprache.
Arbeitet in drei Hauptphasen:



Symboltabelle ::

Compiler erzeugt während Syntaxanalyse **Symboltabellen** für Bezeichner (Name -> (Typ, Speicherort)), überprüft Dopplungen.

Legt für jeden Gültigkeitsbereich (Kontext) eine eigene (überlagernde) Symboltabelle an.



Optimierungen ::

Der Compiler kennt die Zielplatform und kann equivalenten, "besseren" Code erzeugen, z.B.:

- Berechnung und Propagierung von Konstanten

  *4 * 2 -> 8*

- Algebraische Umformungen

  *x + x -> 2 * x -> x << 1*

- Statische Werte in Schleifenköpfen erkennen

  for (i = 0; i < strlen(s); i++)

- Umsortierung von Instruktionen
- Verwendung von Vektorinstruktionen

Der Compiler ist dein Freund!

-O1

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion2
-fif-conversion
-finline-functions-called-once
-fipa-pure-const
-fipa-profile
-fipa-reference
-fmerge-constants
-fmove-loop-invariants
-freorder-blocks
-fshrink-wrap
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-pta
-ftree-ter
-funit-at-a-time
```

-O2

```
-fthread-jumps
-falign-functions
-falign-jumps
-falign-loops
-falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps
-fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize
-fdevirtualize-speculatively
-fexpensive-optimizations
-fgcse
-fgcse-lm
-fhoist-adjacent-loads
-finline-small-functions
-findirect-inlining
-fipa-cp
-fipa-cp-alignment
-fipa-sra
-fipa-icf
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition
-freorder-functions
-frerun-cse-after-loop
-fsched-interblock
-fsched-spec
-fschedule-insns
-fschedule-insns2
-fstrict-aliasing
-fstrict-overflow
-ftree-builtin-call-dce
-ftree-switch-conversion
-ftree-tail-merge
-ftree-pre
-ftree-vrp
-fipa-ra
```

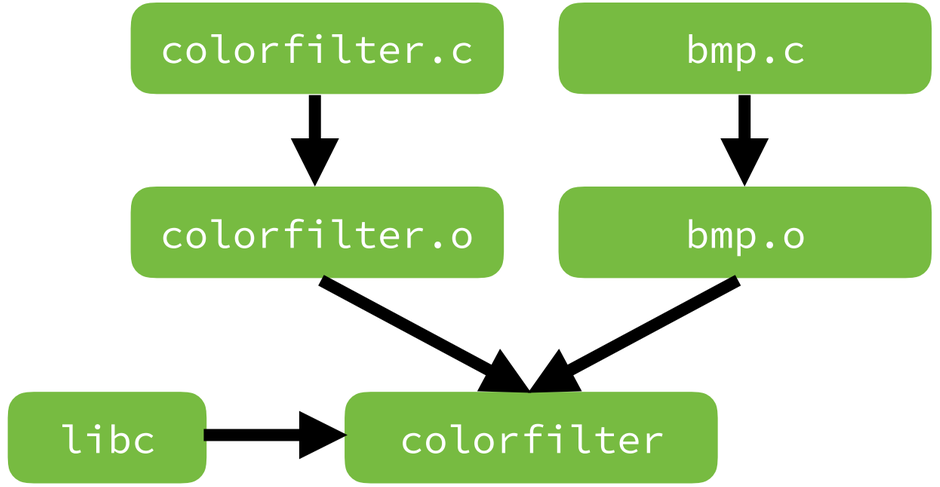
-O3

```
-finline-functions
-funswitch-loops
-fpredictive-commoning
-fgcse-after-reload
-ftree-loop-vectorize
-ftree-loop-distribute-patterns
-fsplit-paths
-ftree-slp-vectorize
-fvect-cost-model
-ftree-partial-pre
-fipa-cp-clone
```

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Programme enthalten häufig:

- Mehrere Übersetzungseinheiten (Module)
- Verweise auf Systemfunktionen (open, …)
- Verweise auf externe Bibliotheken

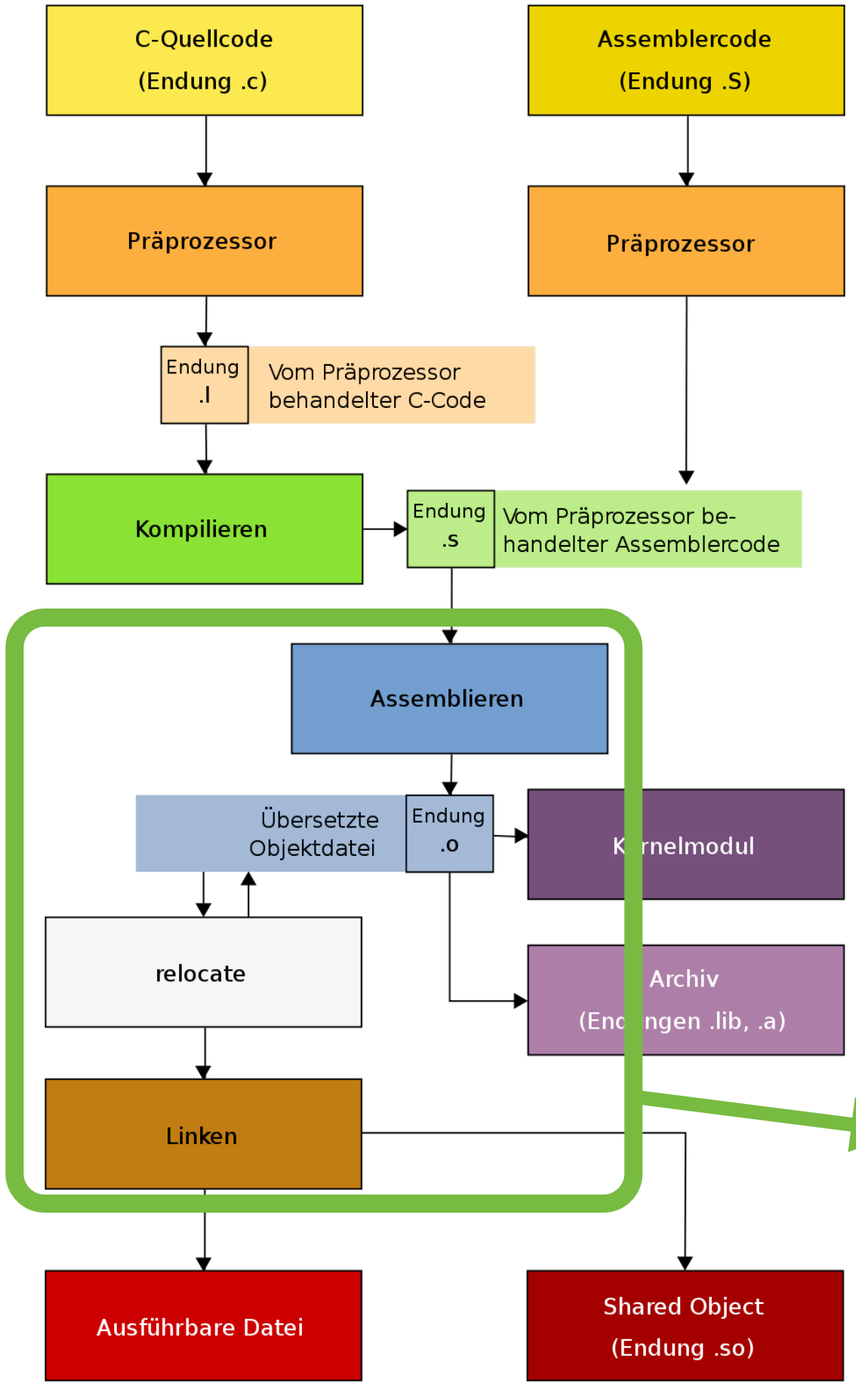


Compiler generiert Tabellen mit relativen Adressen für Variablen, Funktionen.

Der **Linker** (Binder)

- kombiniert verschiedene Module
- passt Adressen überlappungsfrei an
- löst Verweise auf externe Symbole (Variablen, Funktionen) auf
- Bindet die Laufzeitumgebung (Speicherverwaltung, Bibliotheken) ein

Unnötige Symbole können entfernt werden (strip).



GNU Compiler Collection (GCC) ::

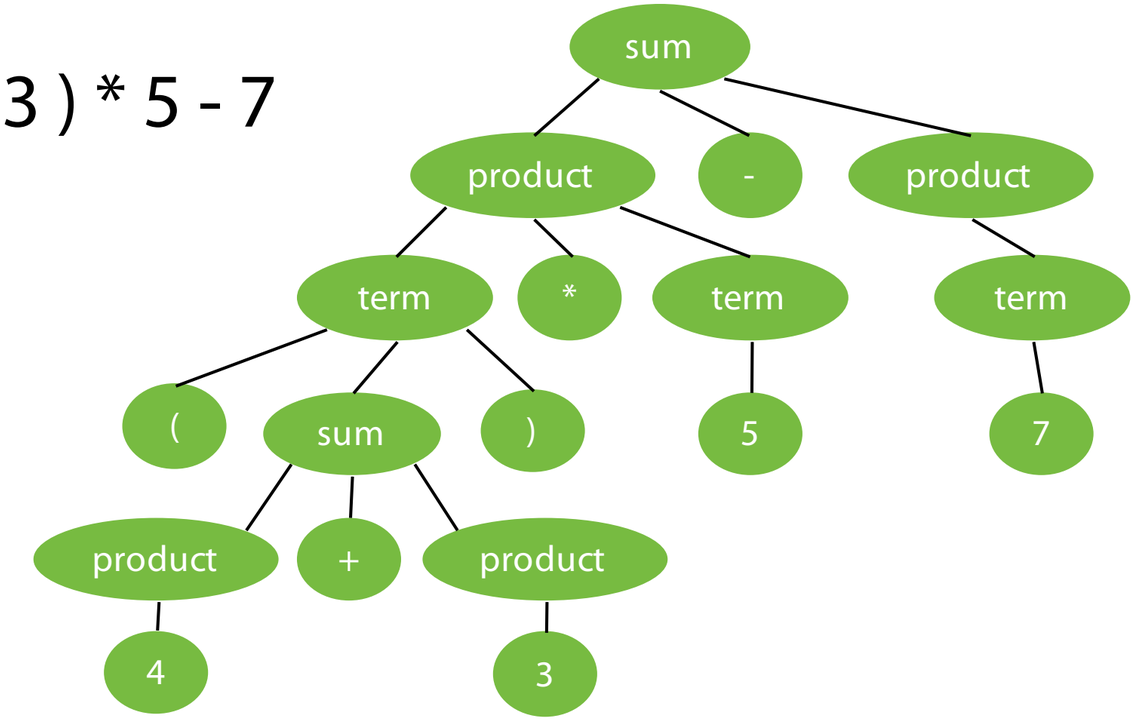
Sammlung von Programmen zur Übersetzung von C, C++, Objective-C, Fortran, Ada, Java, Go (**Frontends**) auf 50 Zielplattformen (**Backends**).

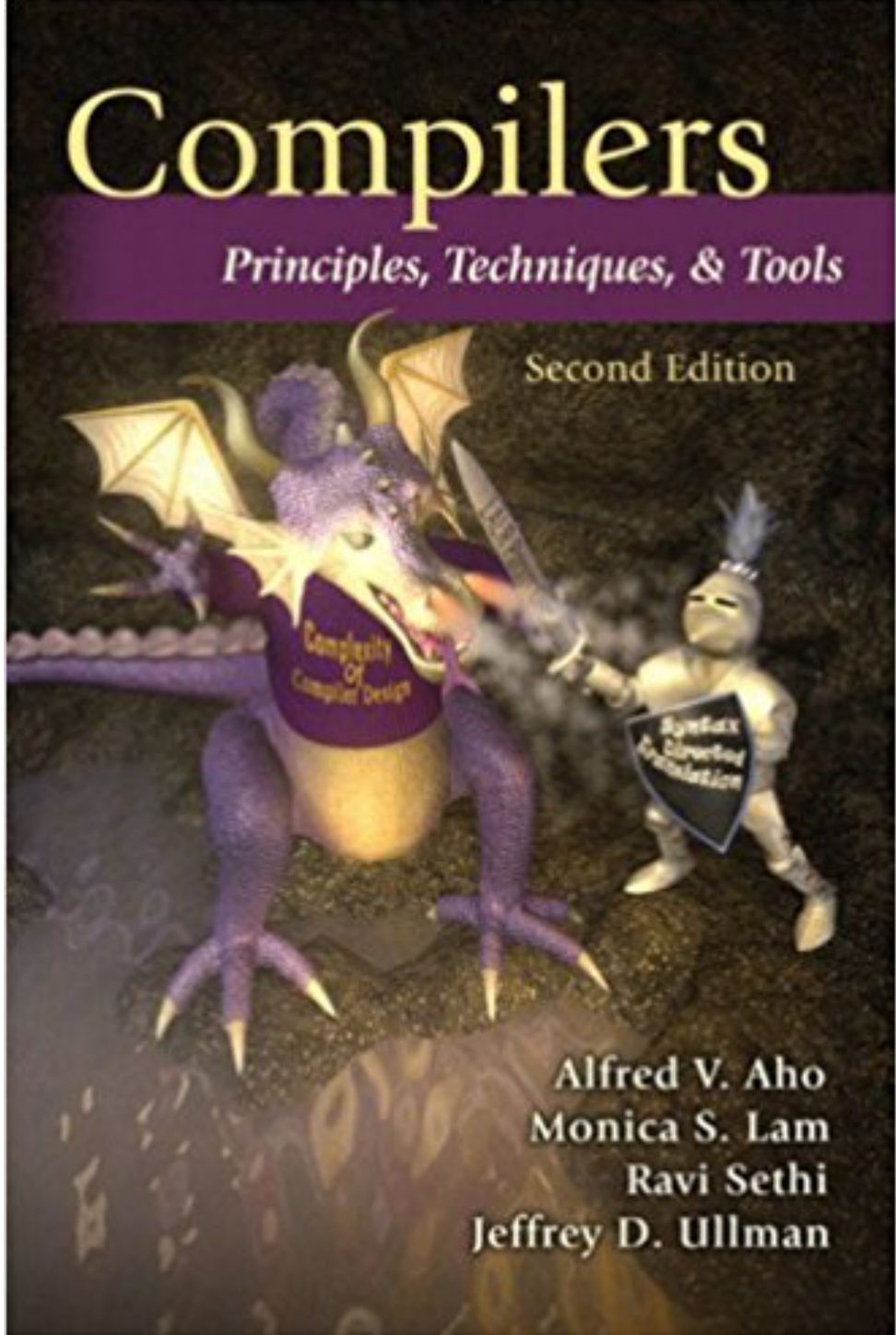
Teil des GNU-Projektes, unter General Public License.

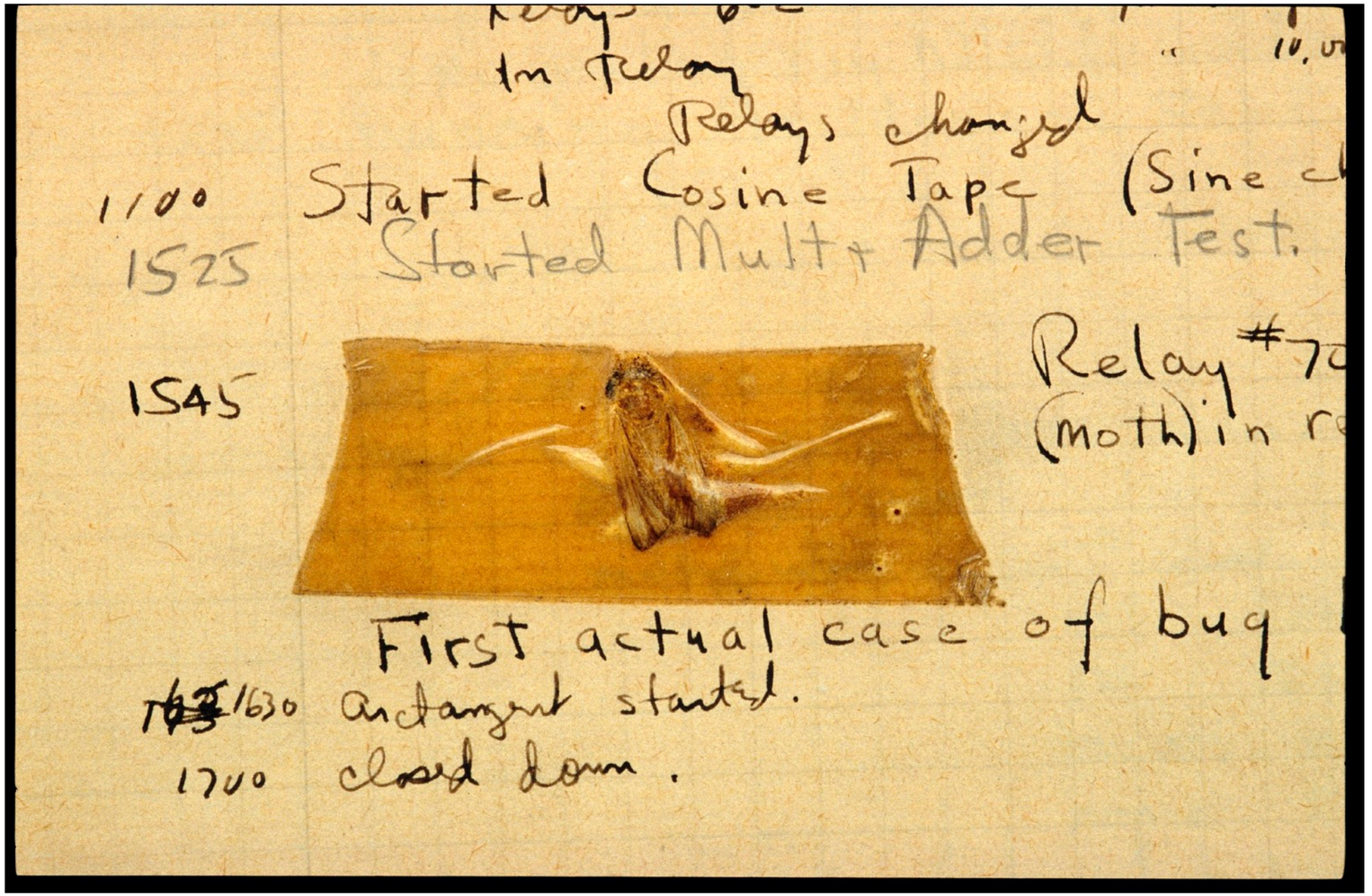
Eng verzahnt mit GNU Binutils.

```
sum     ::= product | sum '+' product
                    | sum '-' product ;
product ::= term | product '*' term
                 | product '/' term ;
term    ::=  '(' sum ')' | Number ;
```

(4+3)*5-7






In Relay
Relays changed
1100 Started Cosine Tape (Sine
1525 Started Mult + Adder Test.
1545
Relay #7
(moth) in r
First actual case of bug
1630 Arctangent started.
1700 closed down.

Debugger

Alle Menschen machen Fehler.
Alle Programmierer sind (noch) Menschen.
Alle Programmierer machen Fehler.

Ein Debugger hilft Programmierfehler zu finden.

beobachtet

Failure (Ausfall) – System ist unfähig die geforderte Funktionalität zu liefern
Error (Fehler) – Diskrepanz zwischen erwartetem und beobachtetem Wert

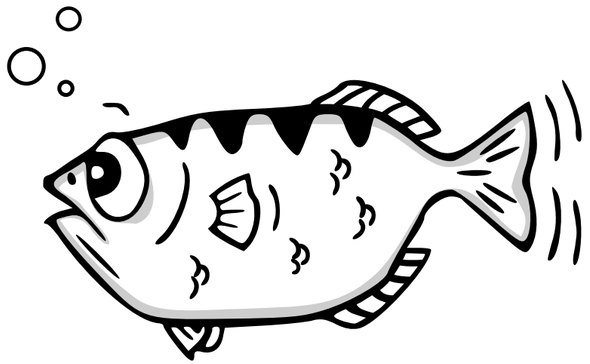
Fault (Fehler) – Falsche Anweisung, Datendatenstruktur, etc. wodurch das Programm in einen ungültigen Zustand eintritt. Kann zu einem Ausfall führen.

muss behoben werden

Zu viele Ausgaben, Verändert zeitliche Abhängigkeiten, Nebeneffekte, Buffer

Code wird unübersichtlich.

The GNU Debugger (gdb) ::



Ein interaktiver Debugger.
Aus dem GNU Projekt, orientiert an dbx (BSD, Solaris).
Unterstützt offizielle GCC-Sprachen, sowie Pascal, Modula-2, …

Beeinflusste durch seine Verbreitung andere Debugger (kgbd, pydb, lldb).
Über serielle Schnittstellen oder Netzwerk für eingebettete Systeme geeignet (gdbserver).

Bietet APIs zur Steuerung aus anderen Sprachen.
Grundlage vieler GUIs und Entwicklungsumgebungen.

Benötigt Debugsymbole in der ausführbaren Datei (erzeugt mit `cc -g`).
Erlaubt:

- Setzen von Breakpoints (Ausführung an bestimmter Stelle anhalten)
- Schrittweises Abarbeiten des Programmes
- Ausgabe des Backtrace (zuletzt gerufene Methoden)
- Variablen inspizieren und verändern
- Aufruf von Funktion außerhalb des Programmablaufs

Nutzt eine interaktive Shell zur Kommunikation und Steuerung (REPL).

```
$ gdb <executable>
GNU gdb (GDB) 8.0.1
Copyright (C) 2017 Free Software Foundation, Inc.
(gdb)
```

factorial.c

< DEMO >

run [args][pipe]	Führ das Programm mit den Argumenten args aus, nutz Pipesemantik der Shell für Eingabe/Ausgabe
break n	Setz Haltepunkt in Zeile n
break file:n	Setz Haltepunkt in Zeile n von Datei file
break fname	Setz Haltepunkt an Funktion fname
break x **if** cond	Setz Haltepunkt an x, halt nur an, wenn die Bedingung cond erfüllt ist
backtrace/**where**	Zeig aktuellen Funktionsaufrufverlauf
print expr	Zeig den C-Ausdruck expr an (Variable, Rechnung …)
step [n]	Geh eine (oder n) Zeile(n) weiter
next [n]	Geh (n) Zeile(n) weiter, überspringe Unterfunktionen
continue	Geh bis zum nächsten Haltepunkt/Watch-Event
finish	Geh bis zum Ende der aktuellen Funktion

list	Quelltext um den aktuellen Kontext
watch var	Halt den Verlauf an (run, continue, …), wenn sich der Wert von var ändert
set variable var **=** expr	Weis var den Wert von expr zu
frame	Nimm einen früheren Aufrufkontext
info locals	Zeig alle lokalen Variablen an
info registers	Zeig den Inhalt der Maschinenregister an
info breakpoints	Zeig aktivierte Breakpoints an
delete breakpoints [n]	Lösch breakpoint n (alle, wenn unspezifiziert)
stepi, **nexti**	Equivalent für einzelne Maschineninstruktionen
help command	Zeig die Hilfe zu command an
<enter>	Wiederhol den letzten Befehl

Cheat-Sheet II

NAME

gdb - The GNU Debugger

SYNOPSIS

gdb [-help] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps] [-tty=dev]
[-s symfile] [-e prog] [-se prog] [-c core] [-x cmds] [-d dir]
[prog[core|procID]]

DESCRIPTION

The purpose of a debugger such as GDB is to allow you to see what is going on ``inside'' another program while it executes--or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

You can use GDB to debug programs written in C, C++, and Modula-2. Fortran support will be added when a GNU Fortran compiler is ready.

man gdb

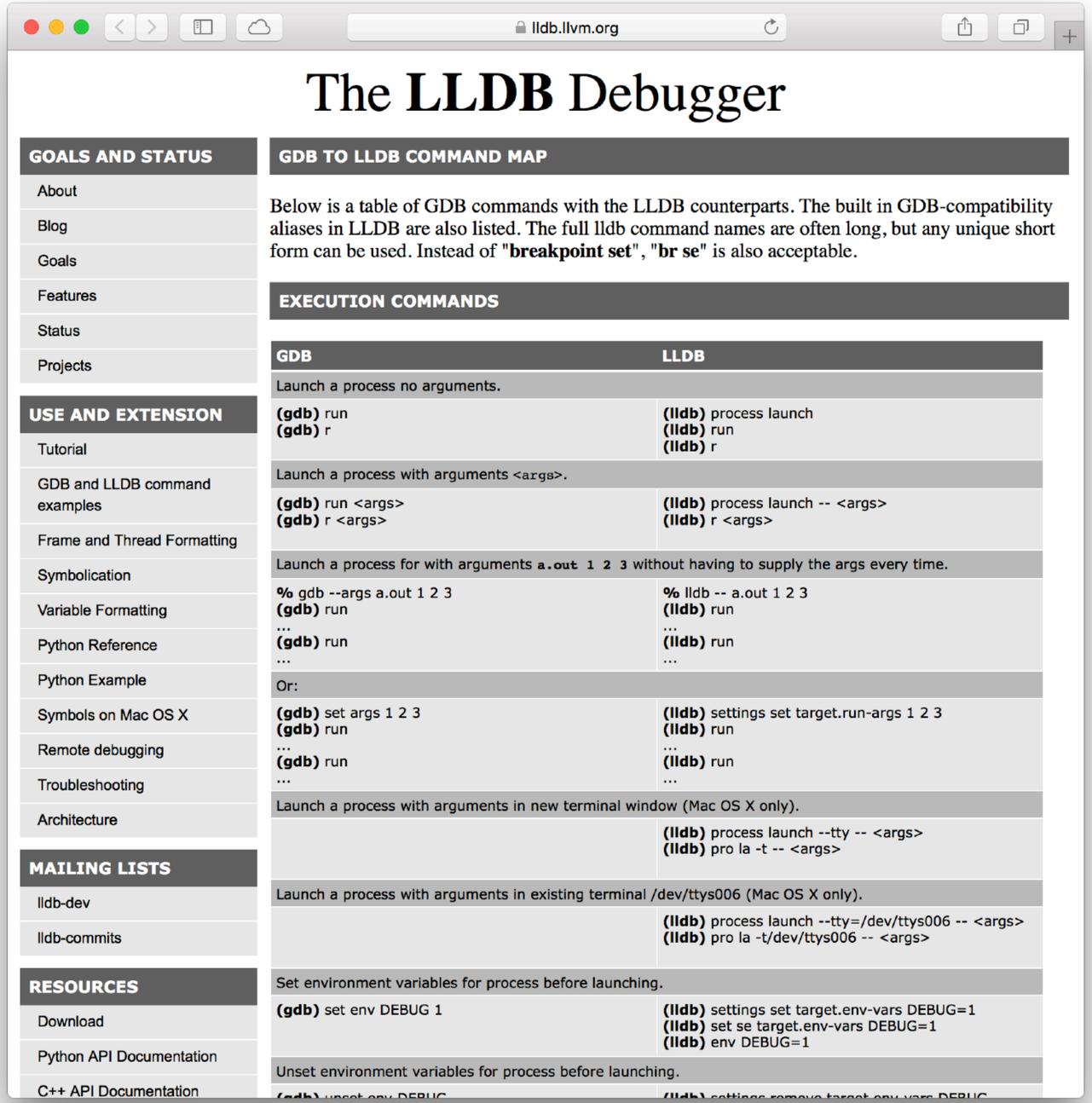
Post Mortem Analyse ::

gdb kann Programme nach dem Absturz analysieren.
Benötigt wird ein **Speicherabzug** (core dump) des letzten Zustands.

```
$ ulimit -c unlimited                  # Aktivieren auf GNU/Linux
$ sudo sysctl -w kern.coredump=1       # Aktivieren auf macOS
$ coredumpctl list                     # Anzeigen im systemd Cache
$ coredumpctl info match
$ gdb program core-dump-file
```

Und noch laufende Prozesse:

```
$ gdb -p pid
```



[https://lldb.llvm.org/lldb-gdb.html]

LLDB unterscheidet sich in den Befehlen.

ende