Computergrundlagen Programmiersprachen, Interpreter, Compiler, Linker

Axel Arnold

Institut für Computerphysik
Universität Stuttgart

Wintersemester 2010/11

Was sind Programme?

- In dieser Vorlesung: Python, C, LaTeX, bash
- Sind das alle Typen von Programmiersprachen?

Wie kann man Programmiersprachen klassifizieren?

- Kein Prozessor versteht Python-, C- oder Shell-Befehle
- Was muss alles passieren, um ein Programm laufen zu lassen?

Wie wird aus unserem Programm etwas, dass der Prozessor ausführen kann?

Programmiersprachen

Imperative Sprachen

Beispiele: Python, C, Shell, BASIC...

- Die meisten Sprachen sind imperativ
- Programme erklären, *wie* ein Problem gelöst werden soll
- Umsetzung des von Neumann-Rechners: Befehle und Schleifen
- **prozedural** heißen Sprachen, die Prozeduren kennen
  Beispiele: alle außer einfachem BASIC

Deklarative Sprachen

- Keine von Neumann-artigen Befehle, kein innerer Zustand
- Rekursion anstatt Schleifen
- **funktional**, basierend auf Funktion im mathematischen Sinn
  Beispiele: Haskell, Erlang, Scheme...
- **logisch**, basierend auf Fakten, Axiomen und logischer Ableitung
  Beispiele: Prolog

Funktionale Sprachen: Haskell

- Beispiel: Berechnung der Fakultät

```
fakultaet :: Integer -> Integer
-- rekursive Definition
fakultaet 0 = 1
fakultaet n = n * fakultaet(n - 1)
-- oder als Produkt
fakultaet n = product [1..n]
```

  - Rekursive Definition der Fakultät, impliziter Abbruch durch Spezialisierung
  - oder mit Hilfe eines endlichen Produkts

- Variablen im mathematischen Sinn
  $x = x + 1$ ist daher unzulässig bzw. falsch
- Beschreibt, *was* berechnet wird, nicht wie

Logische Sprachen: Prolog

- Beispiel: Ahnentafel

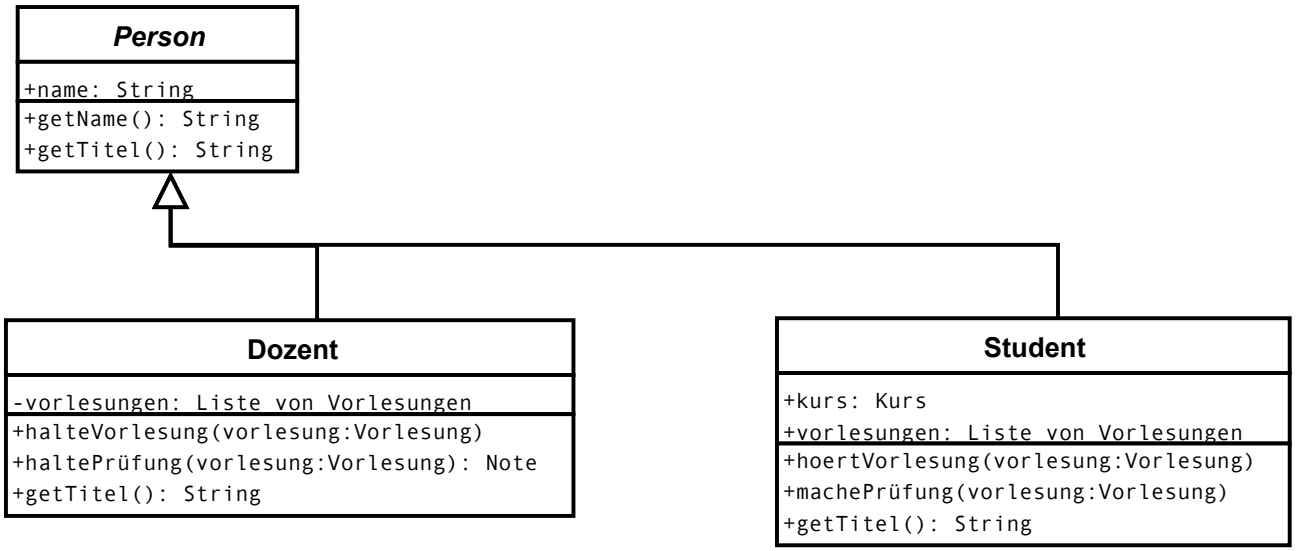
```
ahn(X,Z) :- elter(X,Z).
ahn(X,Z) :- elter(X,Y), ahn(Y,Z).

elter(Axel,Alfred).
elter(Alfred,Alexander).

?- ahn(Axel,Alexander).
yes.
```

- Eingabe sind
  - **Fakten** `p(X)`: X hat Eigenschaft p
  - **Regeln** `p(X) :- p1(V,...,X,...), p2(W,...,X,...)`: X hat Eigenschaft p, wenn es V,W,... gibt, so dass `p1(V,...,X,...)` und `p2(W,...,X,...)` gilt.
- Entwickelt für die Forschung an künstlicher Intelligenz

Objektorientierte Sprachen

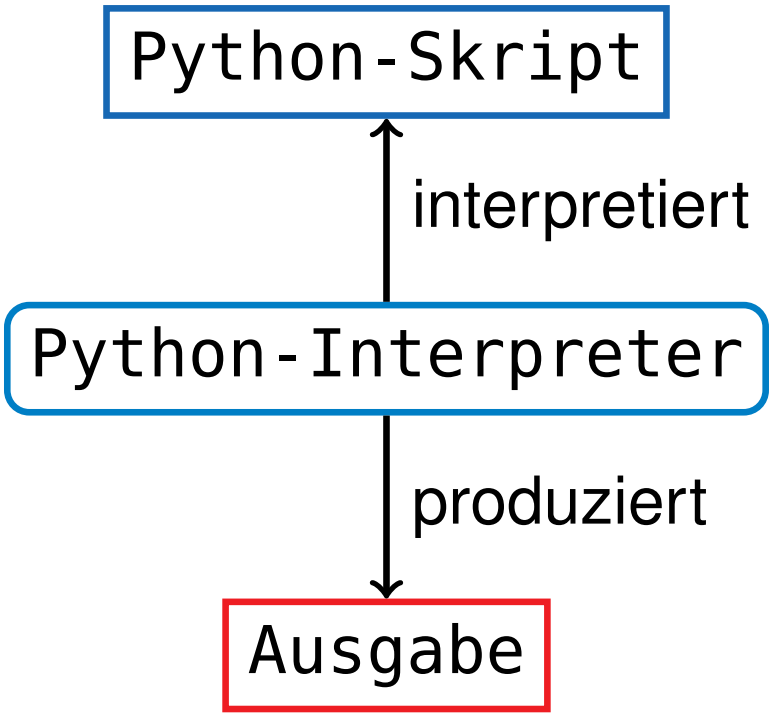


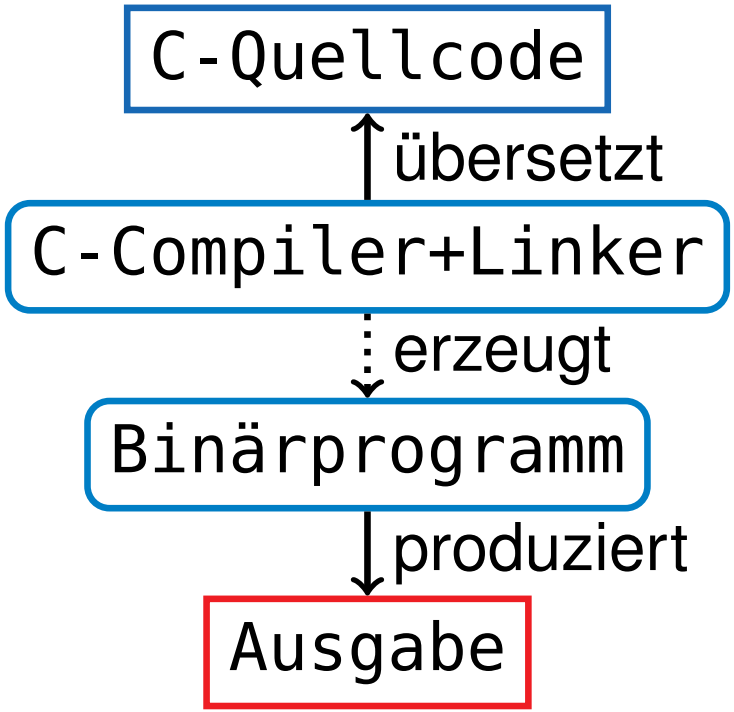
- Objektorientierung (OO) ist ein sprachunabhängiges Programmiermodell
- Ziel ist die bessere Wartbarkeit und Austauschbarkeit von Software durch starke Isolation von Teilproblemen
- Speziell darauf ausgelegt sind z.B. C++, Java, Python,...
- UML (Unified Modelling Language) beschreibt OO-Modelle

Terminologie

- **Klasse**: beschreibt Eigenschaften und Methoden von Objekten
  Beispiel: Klassen sind z.B. Dozent, Student, Person
- **Objekt**: eine Instanz einer Klasse. Ein Objekt hat stets eine Klasse, aber es kann viele Instanzen geben
  Beispiel: Axel Arnold ist ein Dozent, Christian Holm auch
- **Eigenschaften**: Datenelemente von Objekten
  Beispiel: Dozent hat Name, Titel und zu haltende Vorlesungen
- **Methoden**: Funktionen einer Klasse
  Beispiel: Personen können ihren Namen sagen
- **Datenkapselung**: Eigenschaften werden nur durch Funktionen der Klasse verändert
  Beispiel: der Name einer Person kann nicht geändert werden
- **Vererbung**: Klassen können von anderen abgeleitet werden, erben dadurch deren Eigenschaften und Funktionen
  Beispiel: Jede Person hat einen Namen

Interpreter- und Compilersprachen

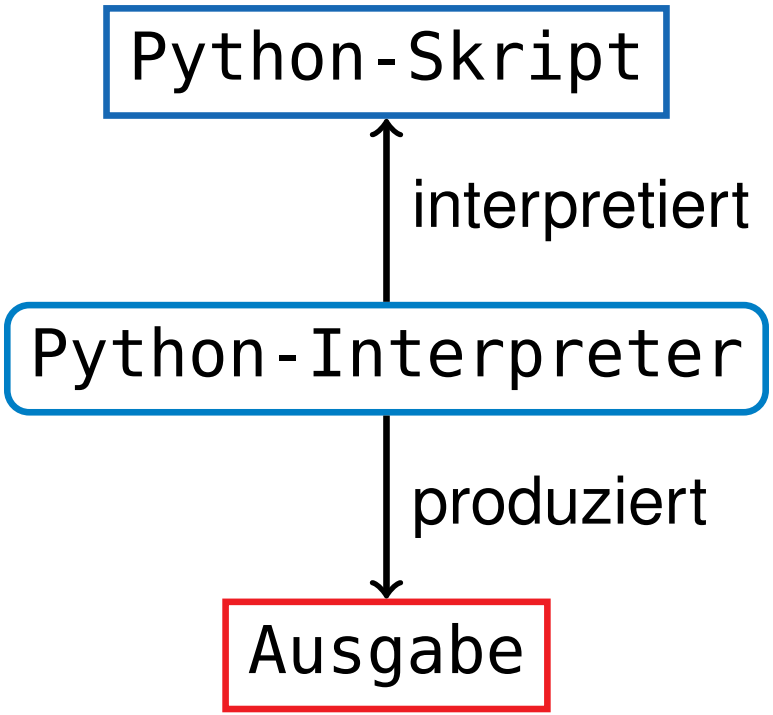


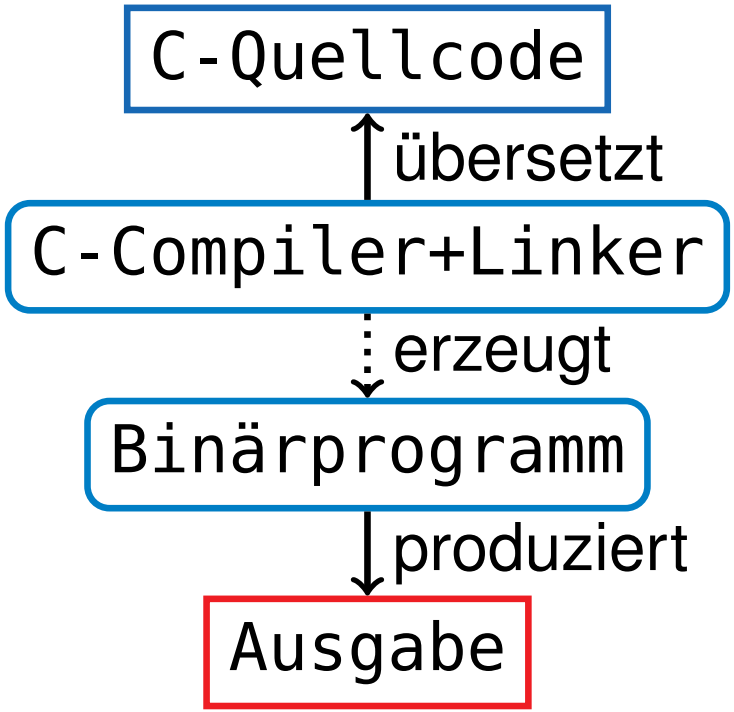


Interpretersprachen

- Z.B. Python, Java, C#, Basic, PHP, Shellsprachen
- Das Programm wird vom Interpreter gelesen und ausgeführt
- Es wird nie in Maschinencode für den Prozessor übersetzt
- Ausnahme: Just-in-Time (JIT) Compiler

Interpreter- und Compilersprachen





Compilersprachen

- Z.B. C/C++, Fortran, Pascal/Delphi
- Compiler übersetzt in maschinenlesbaren Code
- Nicht portabel, erschwerte Fehlersuche, deutlich schneller
- Interpreter selbst müssen compiliert werden

Maschinensprache

Aber was versteht nun der Prozessor?

Zahlenkolonnen, z.B.

```
AD 34 12 18 69 2A 8D 34
12 AD 35 12 69 00 8d 35
12
```

- Maschinencode für den 6502-Mikroprozessor
- Addiert 42 zur 16-bit-Zahl an Speicherstellen 1234h und 1235h
- Wie hängt das mit `x += 42` zusammen?

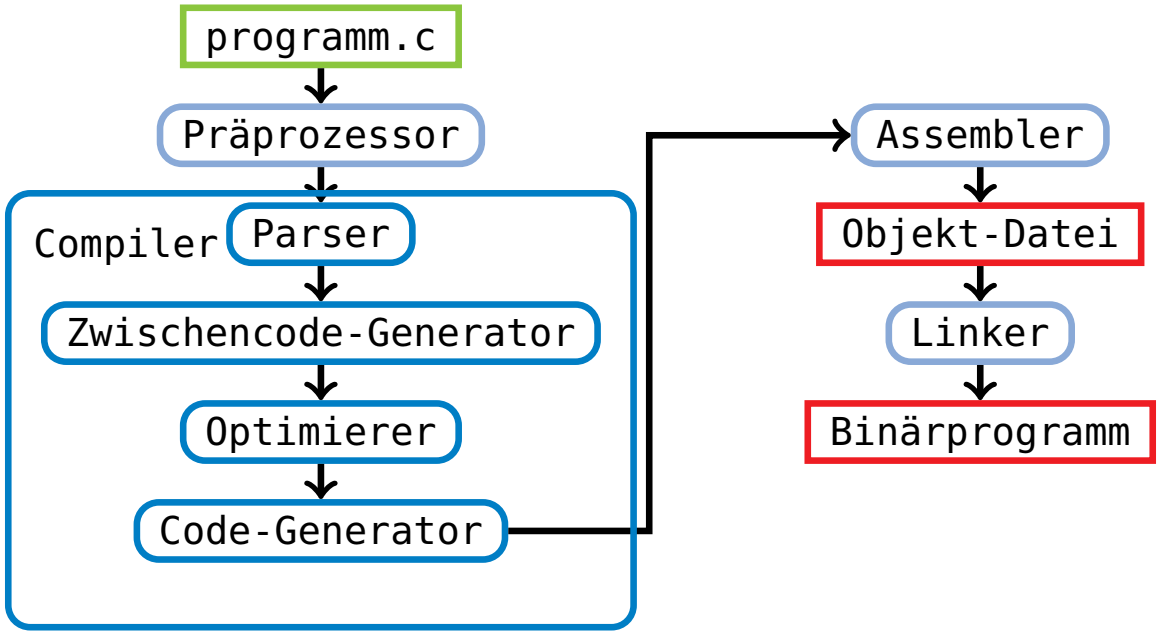
Wie wird aus einem Programm Maschinencode?

Assembler


```
                X     = $1234
AD 34 12              LDA X
18                    CLC
69 2A                 ADC #42
8D 34 12              STA X
AD 35 12              LDA X+1
69 00                 ADC #00
8d 35 12              STA X+1
```

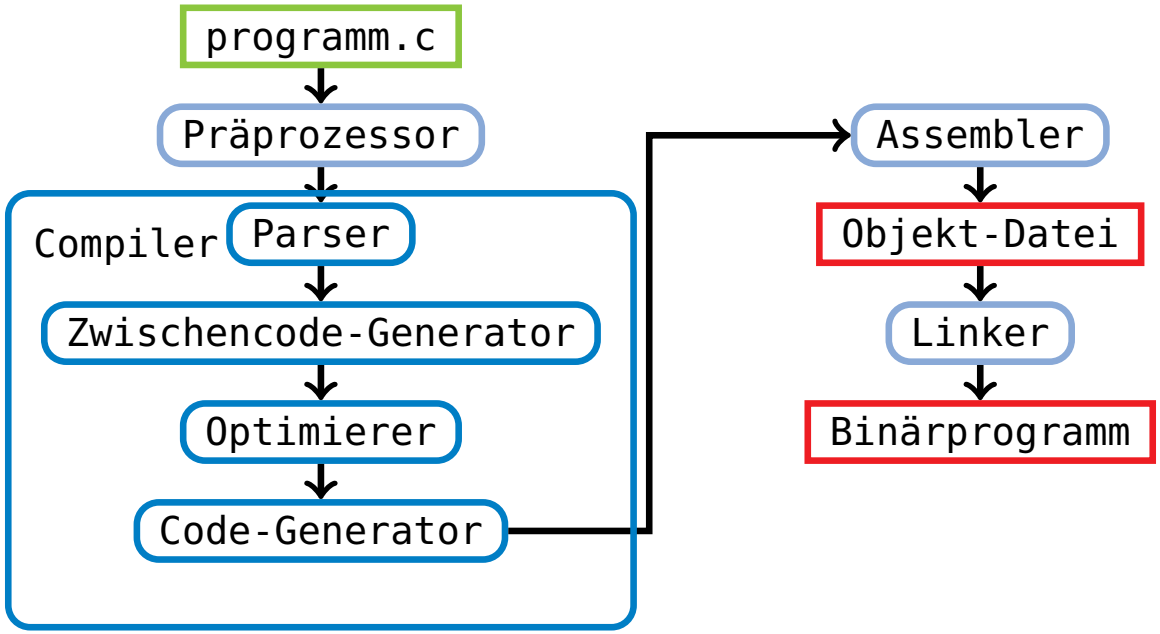
- Assembler ist die menschenlesbare Form der Maschinensprache
- Im Beispiel bezeichnet das *Label* `X` die Speicherzelle 1234h
- Assembler bezeichnet auch das Programm, das die Befehle in Zahlen übersetzt und Labels an Speicherzellen knüpft
- Zeitkritische Anwendungen werden auch heute noch manchmal in Assembler geschrieben

Vom Sourcecode zum Programm



- Ein (C-)Programm durchläuft viele Schritte bis zur fertigen ausführbaren Datei
- **Präprozessor**, **Compiler**, **Assembler** und **Linker** sind meist separate Programme
- meist mehrere Objektdateien aus verschiedenen Quelltextdateien

Vom Sourcecode zum Programm



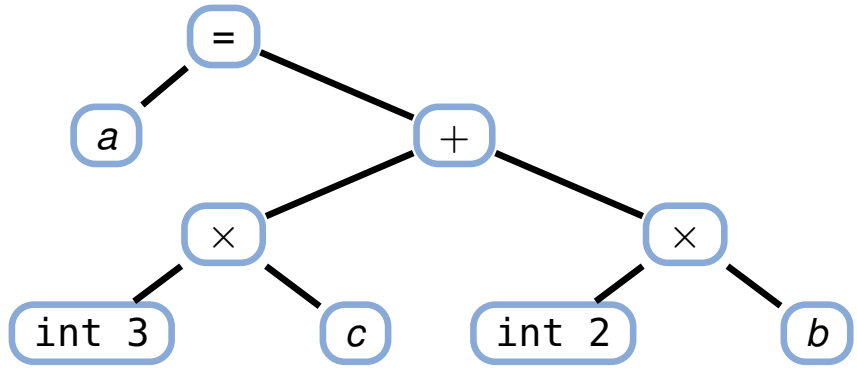
Präprozessor

- Im Prinzip sprachunabhängig, rein textbasiert
- Bindet weitere Quelltextdateien ein
- Erlaubt den Einsatz von *Makros* (Ersetzungen)
  Beispiel: `TEST(...)` wird überall durch `if (...)` ersetzt

Der Compiler

- **Parser** übersetzt den Quellcode in einen Syntaxbaum

$a = 2 * b + 3 * c$ ⟶

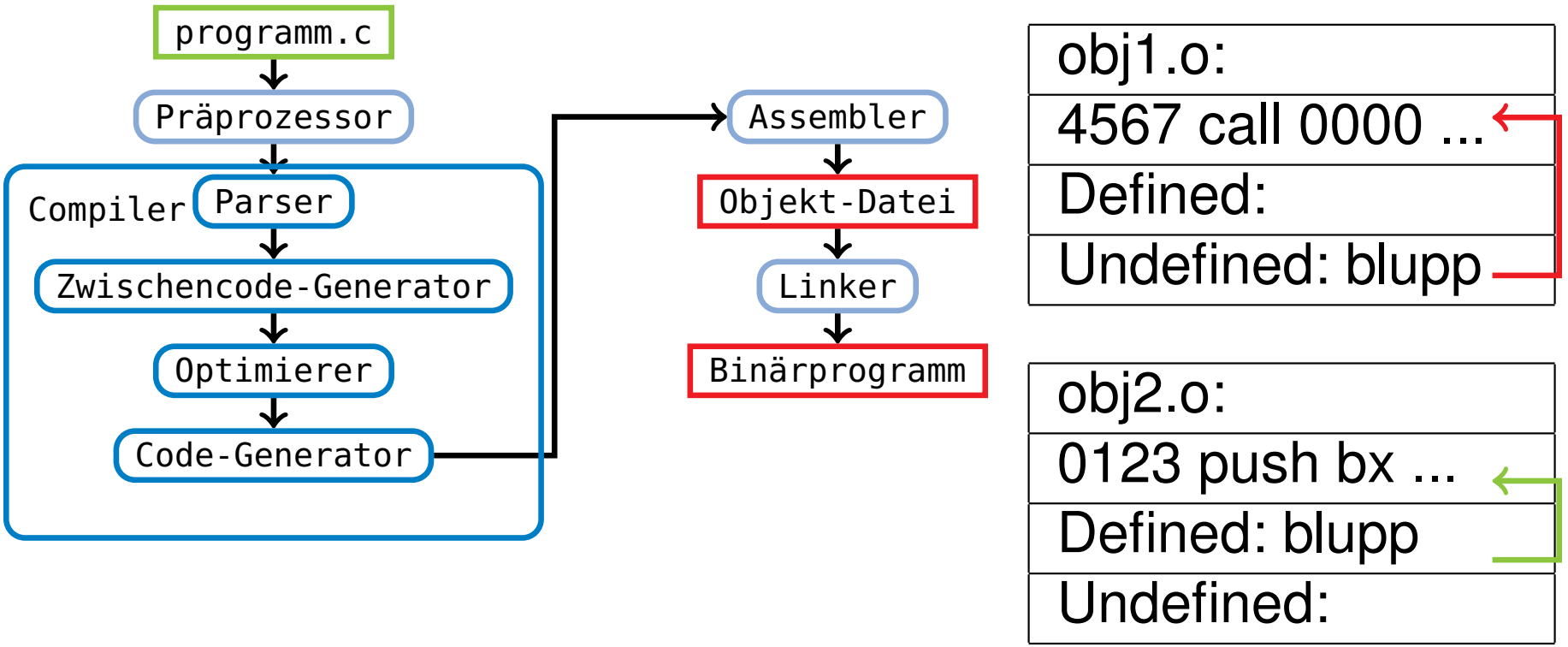


- **Zwischencode-Generator** erzeugt daraus Pseudocode, etwa:

$r1 = b$	$r2 = r1 \times 2$	$r3 = c$	$r4 = r3 \times 3$	$r5 = r2 + r4$	$a = r5$

- **Optimierer** versucht, diesen zu verbessern, z.B. durch
  - Prozessorregisterzuweisung
  - Einfügen kurzer Funktionen, Entrollen von Schleifen
  - Suche und Nutzen von gemeinsamen Termen, ...
- **Code-Generator** erzeugt Assembler aus dem Zwischencode

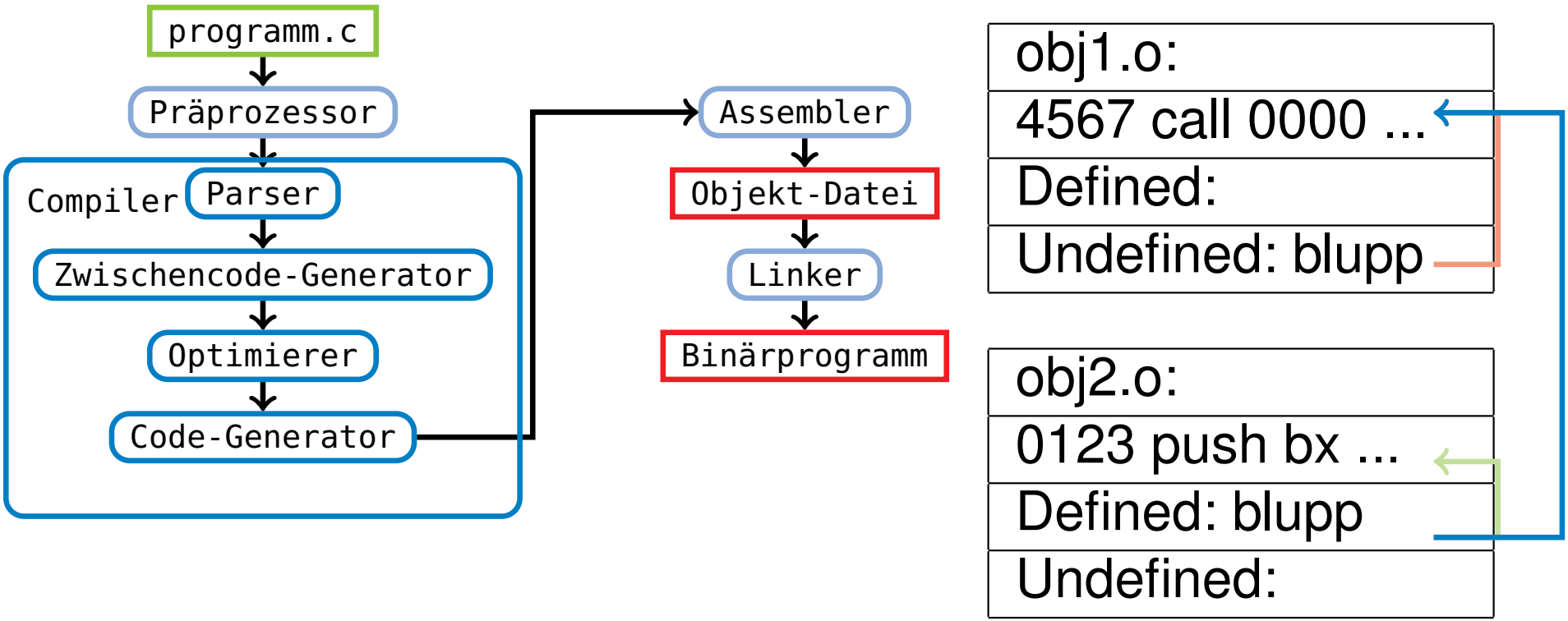
Der Assembler



Der **Assembler** erzeugt eine Objektdatei mit

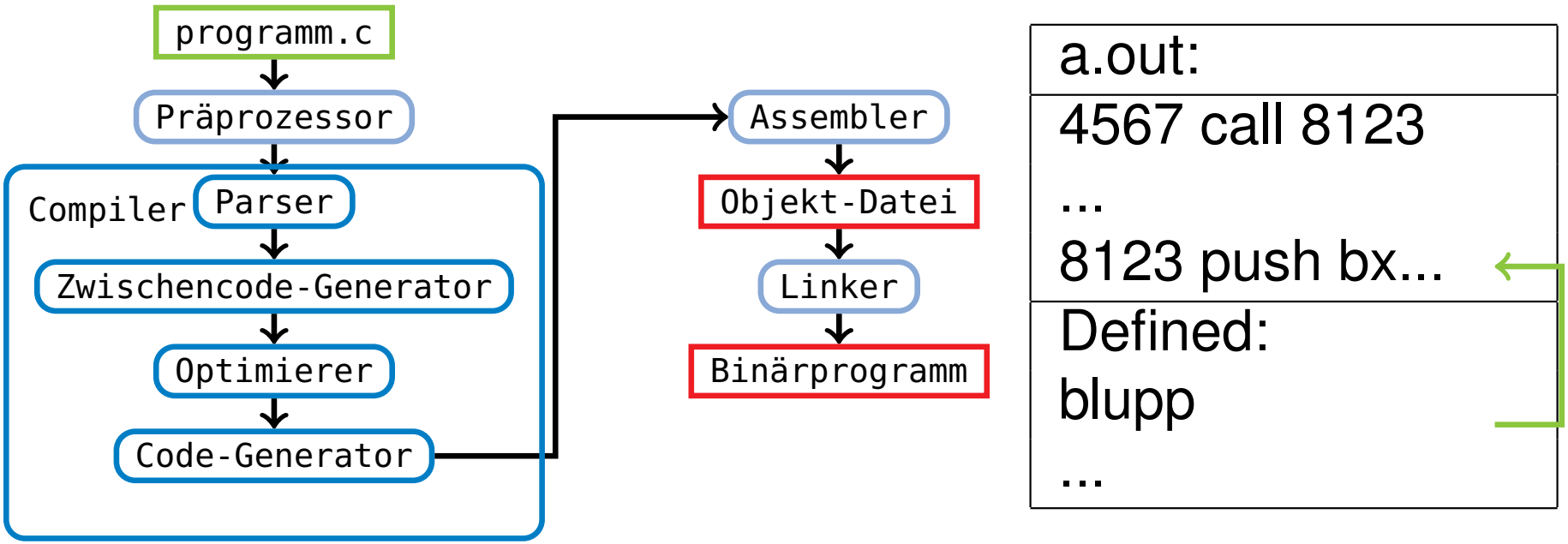
- Maschinencode
- den Speicherpositionen von globalen Variablen und Funktionen
- Stellen, an denen Stellen Information über solche Positionen aus andere Objektdateien benötigt wird

Der Linker



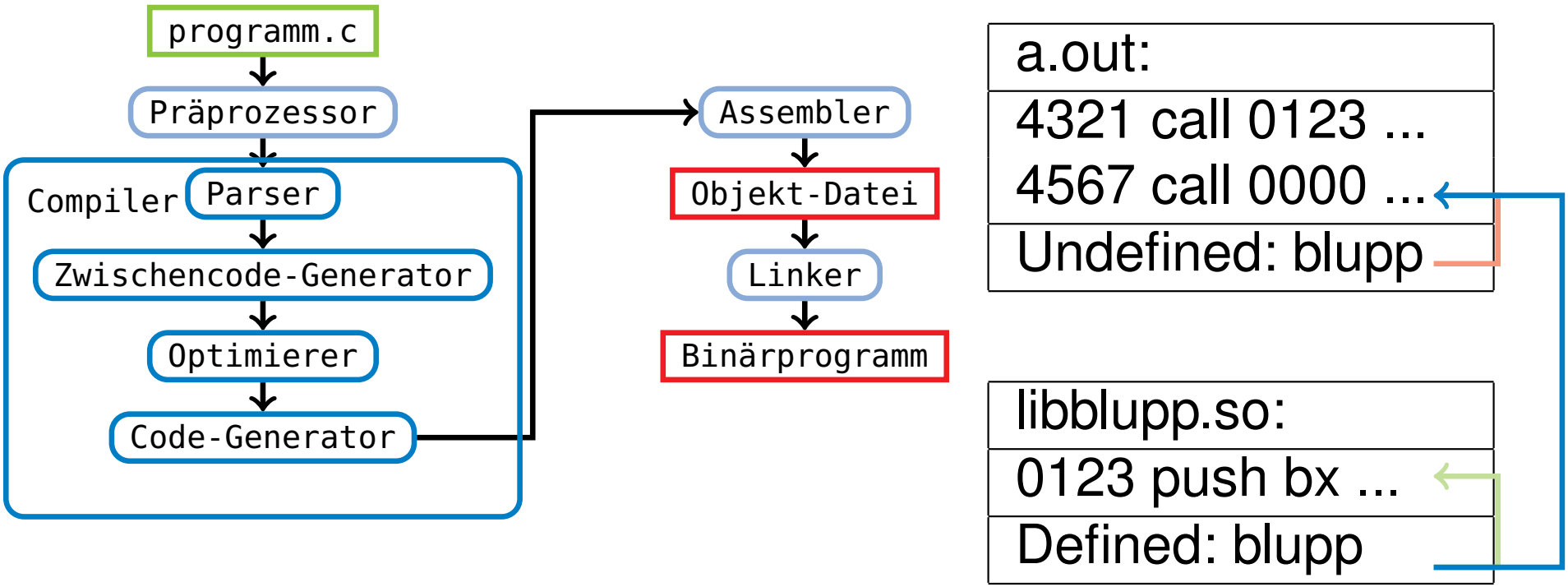
- Verbindet mehrere solcher Objektdateien zu einer ausführbaren Datei (Binary) oder einer **Bibliothek**
- Bibliotheken sind Sammlungen von Objektdateien
- Der Linker verbindet Zugriffe zwischen Objektdateien
- Nur wenn alle Zugriffe aufgelöst wurden, entsteht ein Binary

Der Linker



- Verbindet mehrere solcher Objektdateien zu einer ausführbaren Datei (Binary) oder einer **Bibliothek**
- Bibliotheken sind Sammlungen von Objektdateien
- Der Linker verbindet Zugriffe zwischen Objektdateien
- Nur wenn alle Zugriffe aufgelöst wurden, entsteht ein Binary

Dynamisches Linken



- Beim **dynamischen Linken** wird das Linken gegen die Bibliotheken erst beim Starten des Programms erledigt.
- Dadurch können Teile des Programms geupdated werden
- Und mehrfach benutzte Bibliotheken werden nur einmal geladen
- Spezielle Formate: dynamische Bibliotheken (.so, .dylib oder .dll)