Organisatorisches/Einstieg

Florian Adamsky, B. Sc. (PhD cand.)

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

Softwareentwicklung im WS 2014/15

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Outline

Table of Contents

Motivation: Warum Programmieren lernen?

- YouTube-Video: What Most Schools Don't Teach
- Programmieren ist die Sprache des 21. Jahrhunderts

Entscheidungsproblem

Theorem (Entscheidungsproblem)

Gibt es ein System von Axiomen, aus denen alle Gesetze der Mathematik mechanisch ableitbar sind?

David Hilbert (1900)

Entscheidungsproblem

Theorem (Entscheidungsproblem)

Gibt es ein System von Axiomen, aus denen alle Gesetze der Mathematik mechanisch ableitbar sind?

David Hilbert (1900)

Negativbeweise durch:

- 1936: Alonzo Church mit seinem *Lambda-Kalkül*
- 1936: Alan Turing mit der *Turing Maschine*

□

λ-Kalkül

- minimale universelle Programmiersprache
- entwickelt von Church und Stephen Kleene
- bestehst aus:

Funktionsabstraktion

$\lambda\, x\, .\, A$ anonyme Funktion die x als Parameter bekommt und A als Funktionskörper hat

Funktionsapplikation

$F\, A$ die Funktion F wird auf den Ausdruck A angewendet

Abbildung: Alonzo Church by Princeton University

Turingmaschine

- mathematisches Modell das aus folgenden Komponenten besteht:
  - unendlich langem Speicherband mit Feldern
  - programmierbaren Lese- und Schreibkopf $q1$
- Turingmaschine ist die Basis der Von-Neumann-Architektur

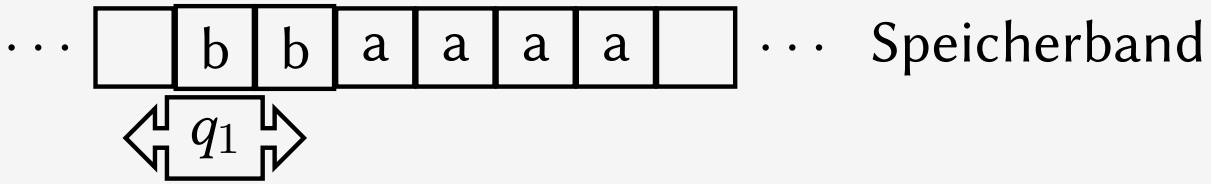


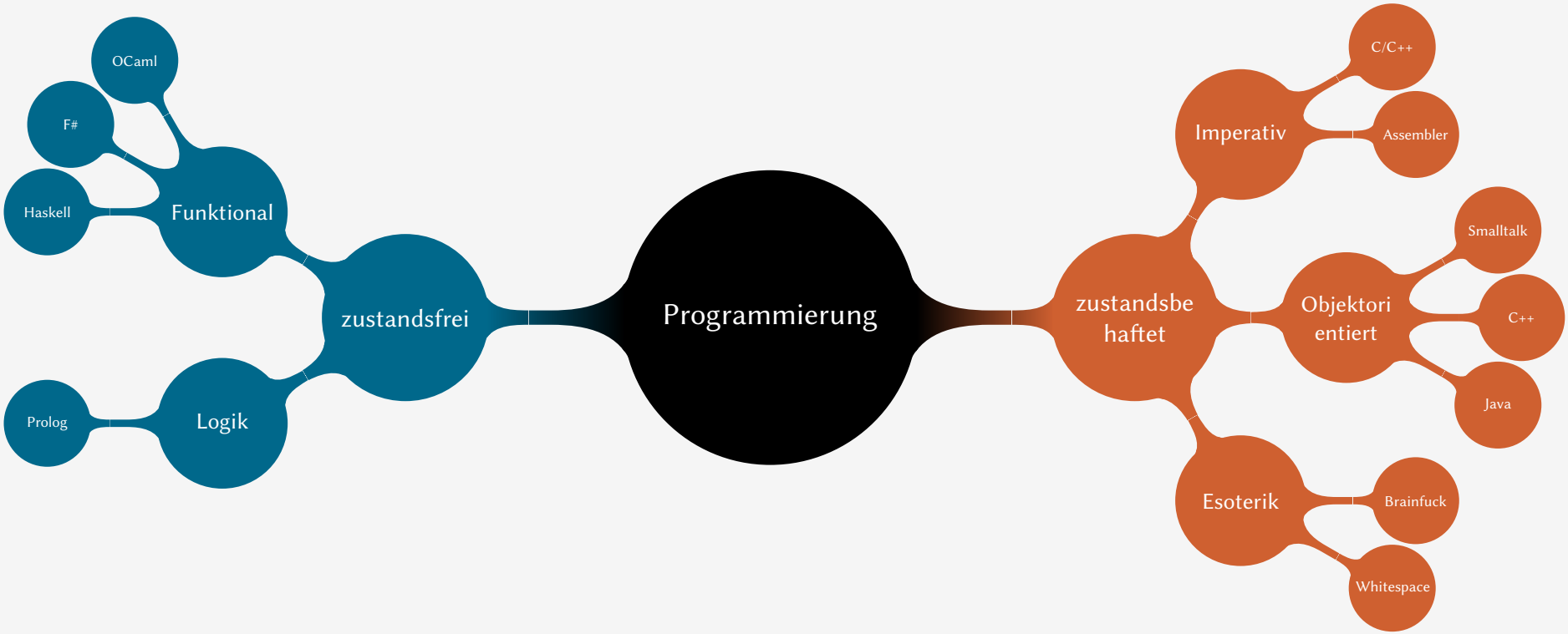
Abbildung: Darstellung auf Basis von Sebastian Sardina

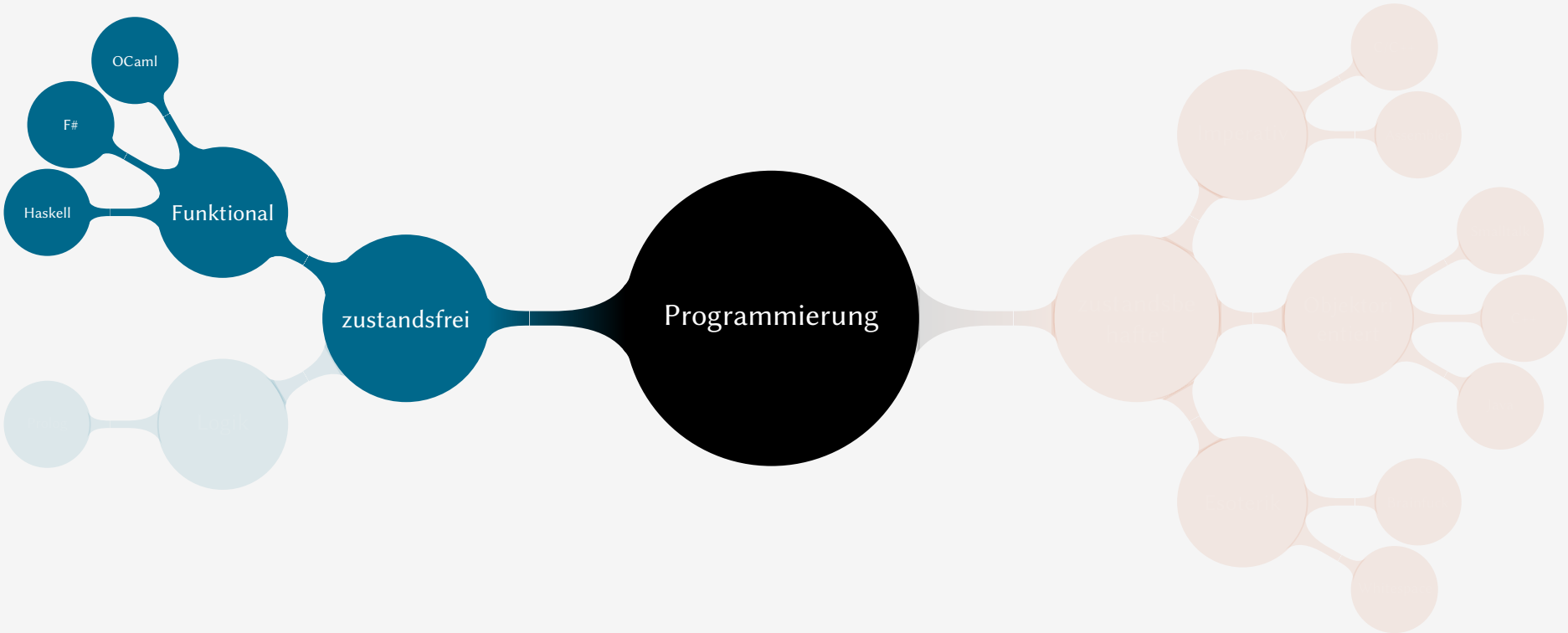
Abbildung: Alan Turing

Definitionen

Programmierparadigma ist ein Programmierstil dem verschiedene Prinzipien zugrunde liegen. Eine spezielle Art über Probleme nachzudenken.

Programmiersprache formale Sprache die ein oder mehrere Programmierparadigmen unterstützt (Multiparadigmen-Sprache).





Funktionale Programmierung

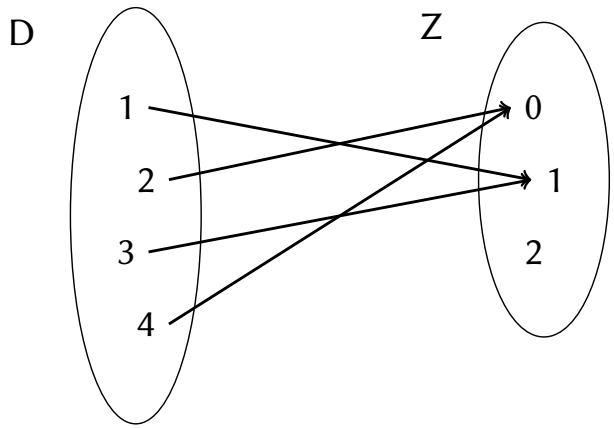
- Basiert auf dem λ-Kalkül
  - Formales Modell das auf Alonzo Church (1930) zurückgeht
- Funktionen entsprechen der mathematische Definition
  - Funktionen sind nur abhängig von den Argumenten

Funktionale Programmierung

- Basiert auf dem λ-Kalkül
  - Formales Modell das auf Alonzo Church (1930) zurückgeht
- Funktionen entsprechen der mathematische Definition
  - Funktionen sind nur abhängig von den Argumenten

Definition (Funktion)

$f: D \to Z: x \mapsto f(x)$, d.h. jedem Element $x \in D$ ist genau ein Element durch $f(x) \in Z$ zugeordnet.



Beispiel für funktionale Programmierung

Example (Imperative – C#)

```
int sum = 0;
foreach (int i in mylist) {
    sum += (i + 1);
}
```

Beispiel für funktionale Programmierung

Example (Imperative – C#)

```
int sum = 0;
foreach (int i in mylist) {
    sum += (i + 1);
}
```

Example (Funktional – Lisp)

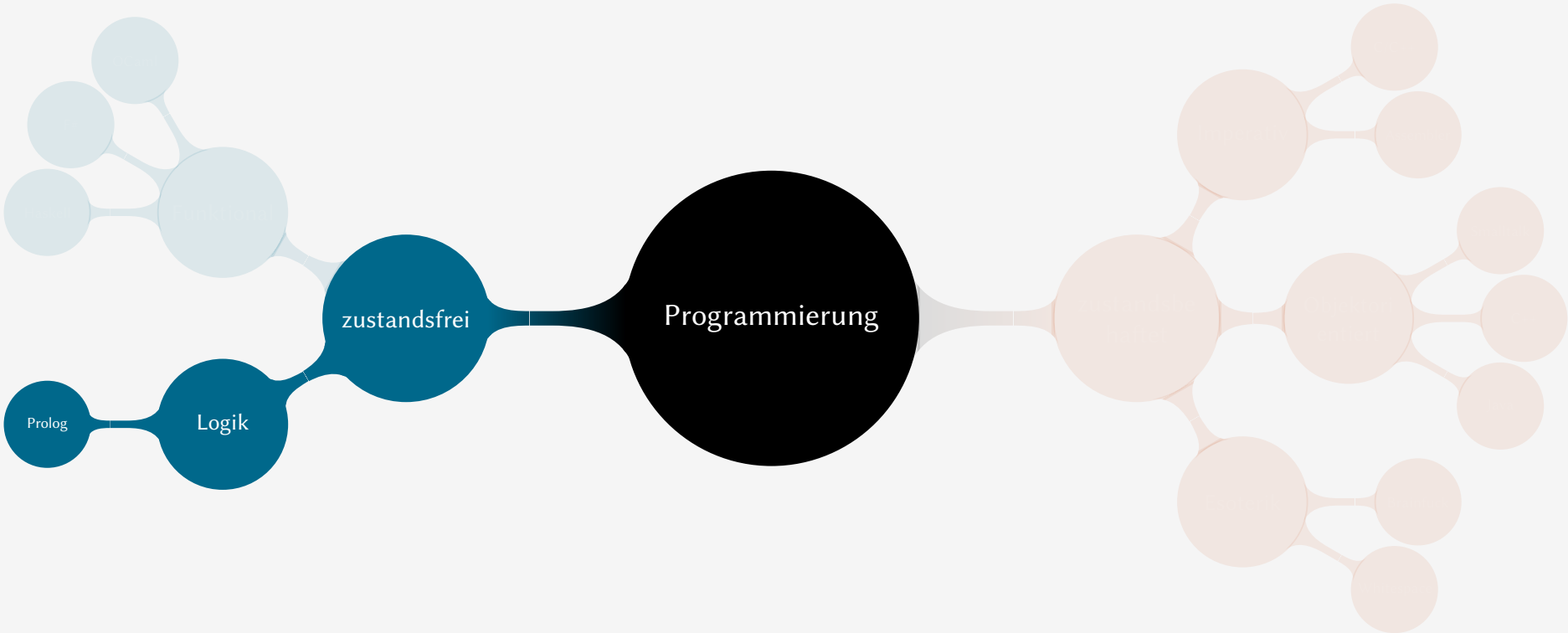
```
(reduce '+ (mapcar '1+ mylist))
```

Prinzipien der funktionalen Programmierung

- First-Class Funktionen
- Higher-Order Functions
- Lazy Evaluation
- Rekursion

Programmiersprachen

Common Lisp, Racket, Clojure, Scheme, Haskell, F#, OCaml, R, Erlang, Mathematica, . . .

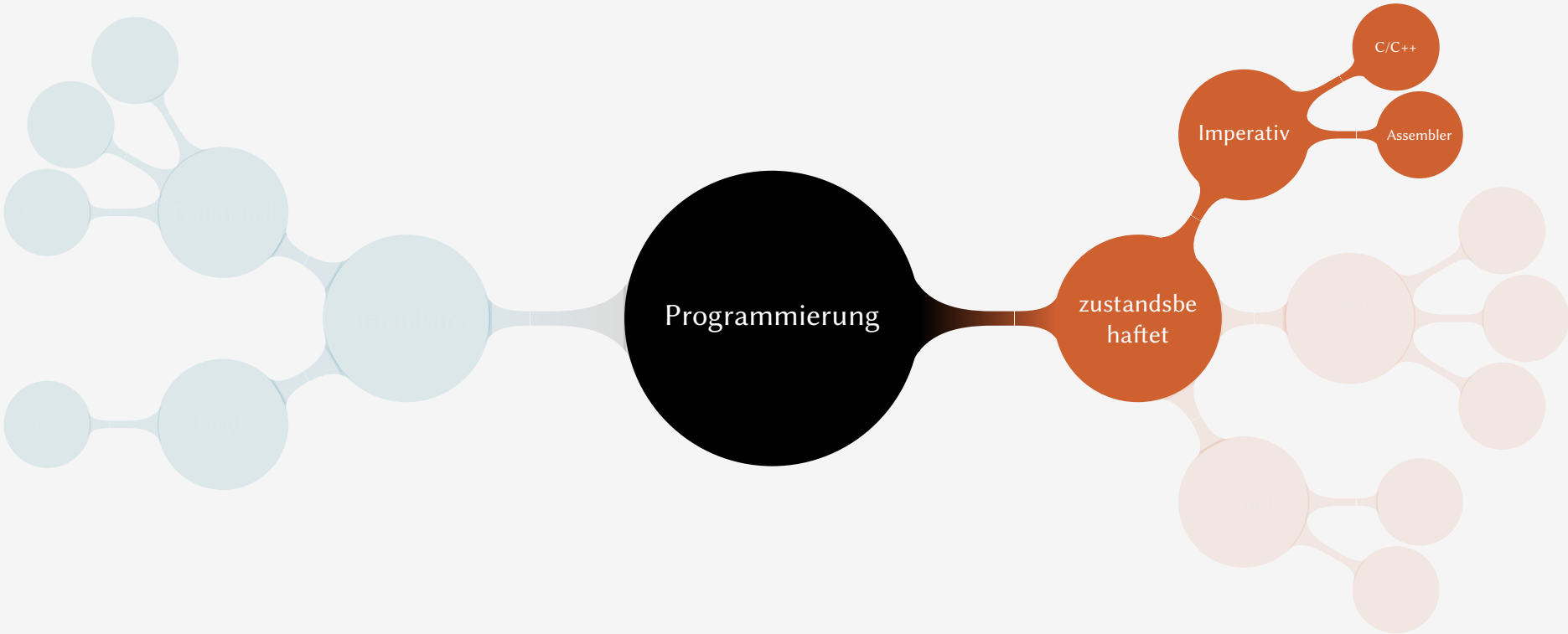


Logik Programmierung

- basiert auf mathematischer Logik
- besteht daher aus einer Menge aus Axiomen

Example (Prolog)

```
grossvater(X,Y) :- vater(X,Z), vater(Z,Y).

vater(adam,tobias).
vater(tobias,frank).

?- grossvater(adam,frank).
true.
```

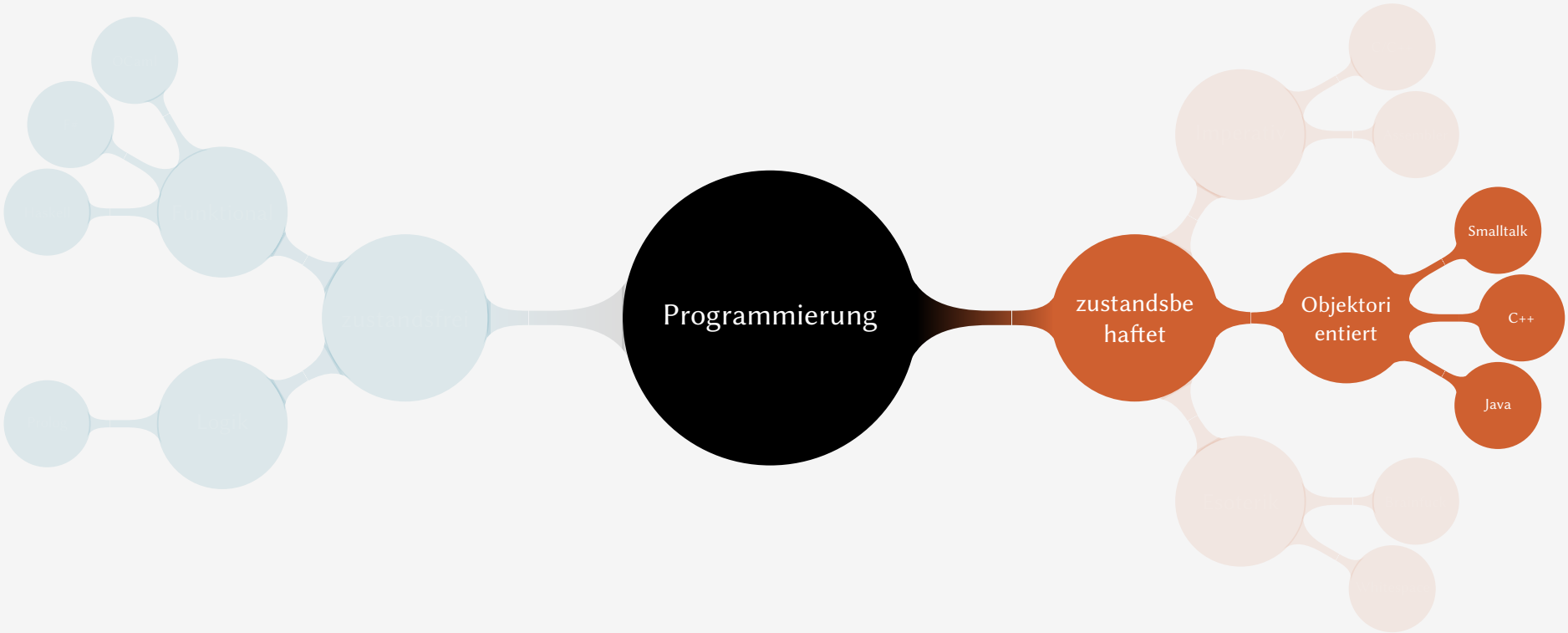


Imperative Programmierung

- längste bekannte Programmierparadigma
- Quellcode legt fest, was in welcher Reihenfolge ausgeführt wird
- vergleichbar mit einem Kochrezept
  - tue erst das, dann das

Programmiersprachen

C, Fortran, Ada, Cobol, Assembler



Objektorientierte Programmierung

- Abbildung der Wirklichkeit in Form von Objekten, Attributen und Methoden
- Versteckt die imperative Programmierung in Klassen

Example (Rechteck)

Attribute höhe, breite

Methoden vergrößern, berechneFlaechenInhalt, verschieben, . . .

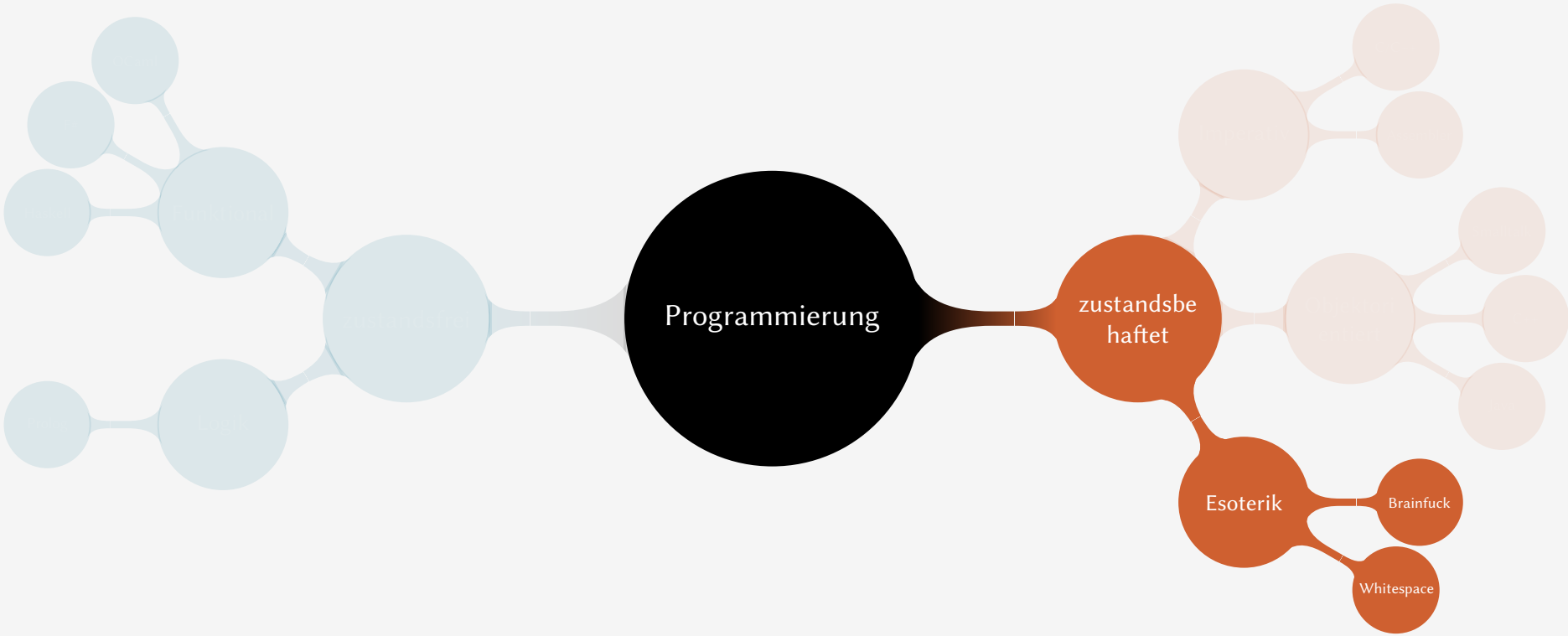
```
Rechteck quadrat(32, 32);
quadrat.vergroessern(100);
quadrat.berecheFlaecheninhalt();
quadrat.verkleinern(40);
```

Prinzipien der OO Programmierung

- Klassen, Objekte, Vererbung
- Datenkapselung
- Polymorphismus
- Komposition und Aggregation

Programmiersprachen

Simula, Smalltalk, Java, C#, C++, Python, Ruby, Perl, Lisp (CLOS), . . .



Esoterische Programmierung

- Sind nicht für den praktischen Einsatz gedacht
- Man versucht möglichst unverständliche Syntax zu erzeugen
- Viel Spaß auf `http://esolangs.org/wiki/Main_Page`

Programmiersprachen

Brainfuck, Whitespaces, Shakespeare Programming Language, Omgrofl,

Hello World in esoterischen Sprachen

Example (Brainfuck)

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<++
++++++++++++.>.+++.------.--------.>+.>.
```

Hello World in esoterischen Sprachen

Example (Brainfuck)

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<++
++++++++++++.>.+++.------.--------.>+.>.
```

Example (Omgrofl)

```
lol iz 72
rofl lol
lol iz 101
```

Hello World in esoterischen Sprachen

Example (Brainfuck)

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<++
++++++++++++.>.+++.------.--------.>+.>.
```

Example (Omgrofl)

```
lol iz 72
rofl lol
lol iz 101
```

Example (SPL)

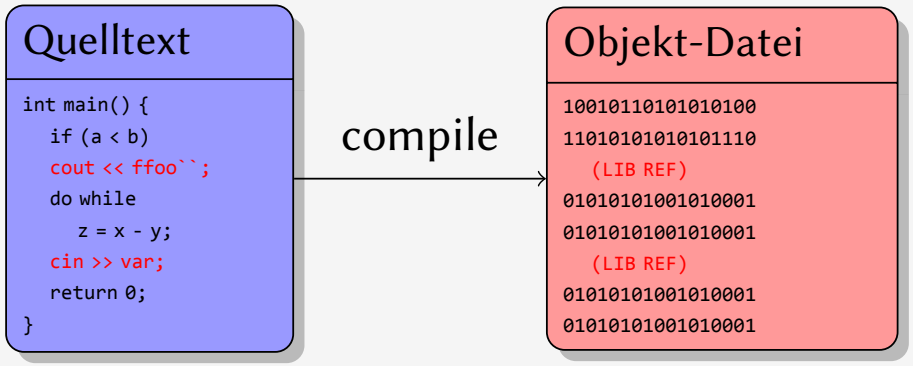
```
Romeo, a young man with a remarkable patience.
Juliet, a likewise young woman of remarkable grace.
```

Compilevorgang Schritt 1

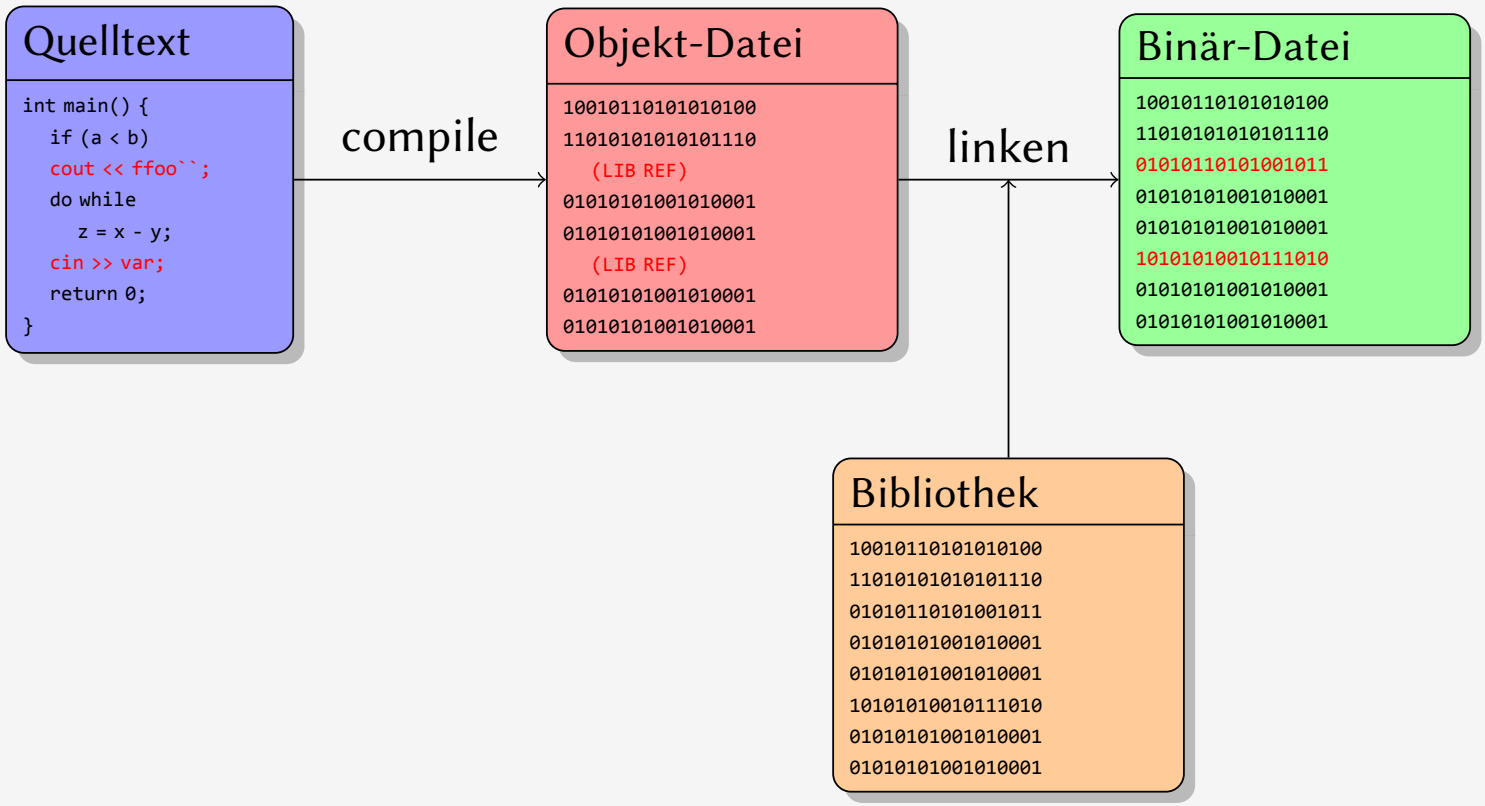
Quelltext

```
int main() {
   if (a < b)
   cout << ffoo``;
   do while
      z = x - y;
   cin >> var;
   return 0;
}
```

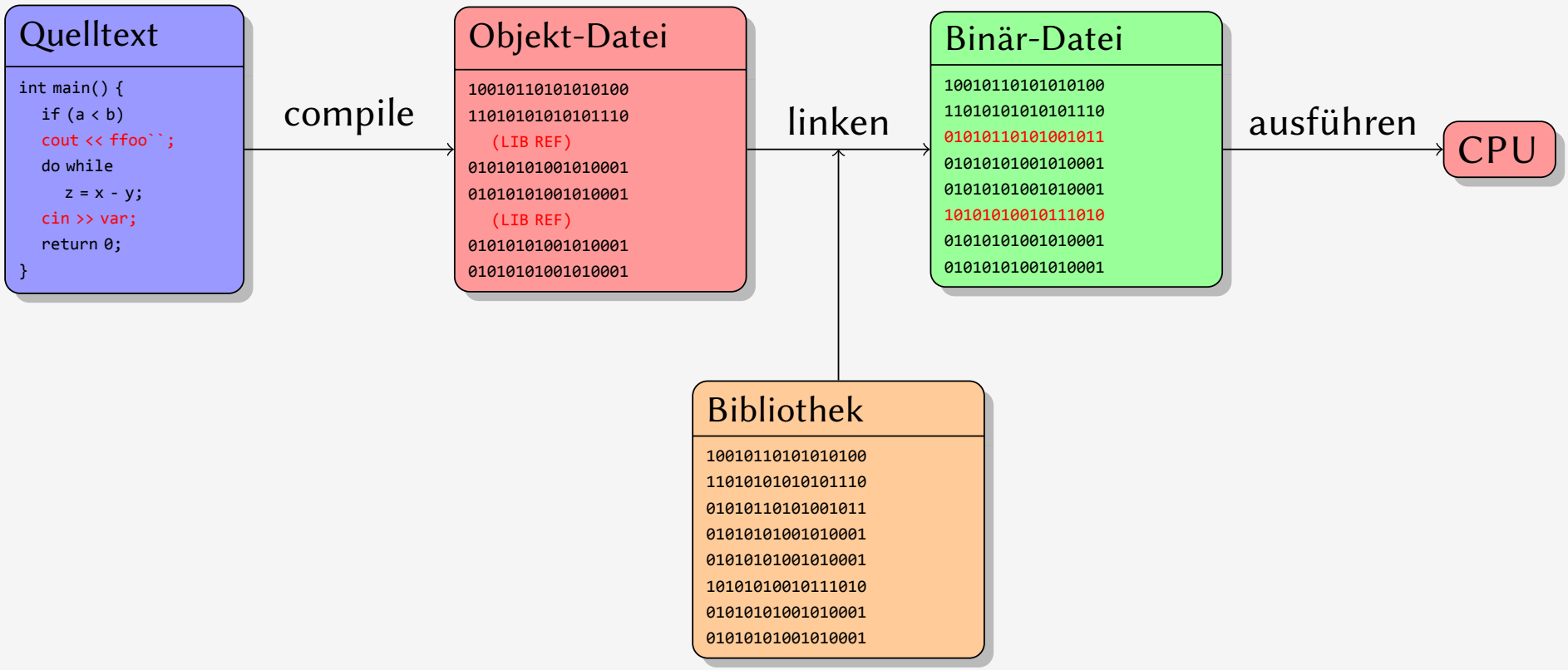
Compilevorgang Schritt 1



Compilevorgang Schritt 1



Compilevorgang Schritt 1



Compiler Vor- und Nachteile

Vorteile

- sehr schnell, da Compiler viele Optimierung vornehmen kann

Nachteile

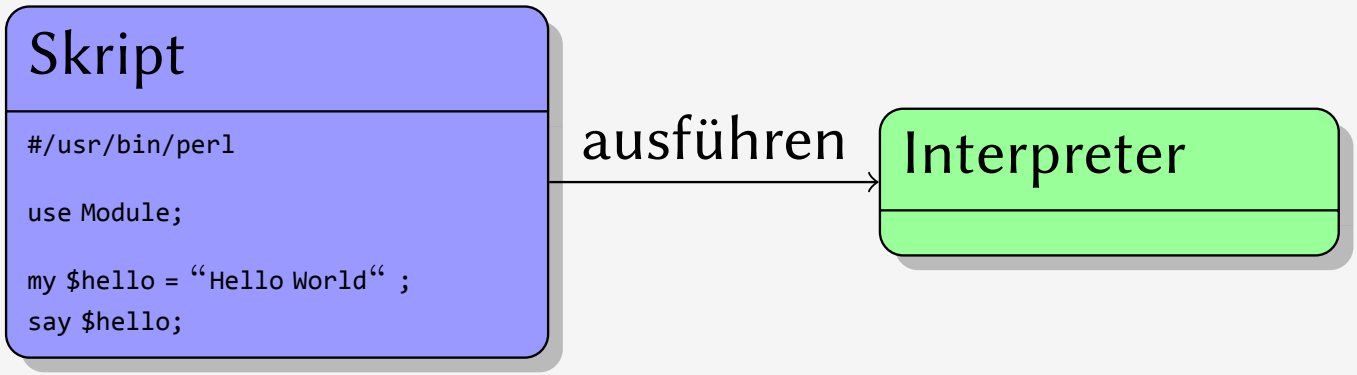
- schwerer zu programmieren
- Systemabhängig

Interpreter

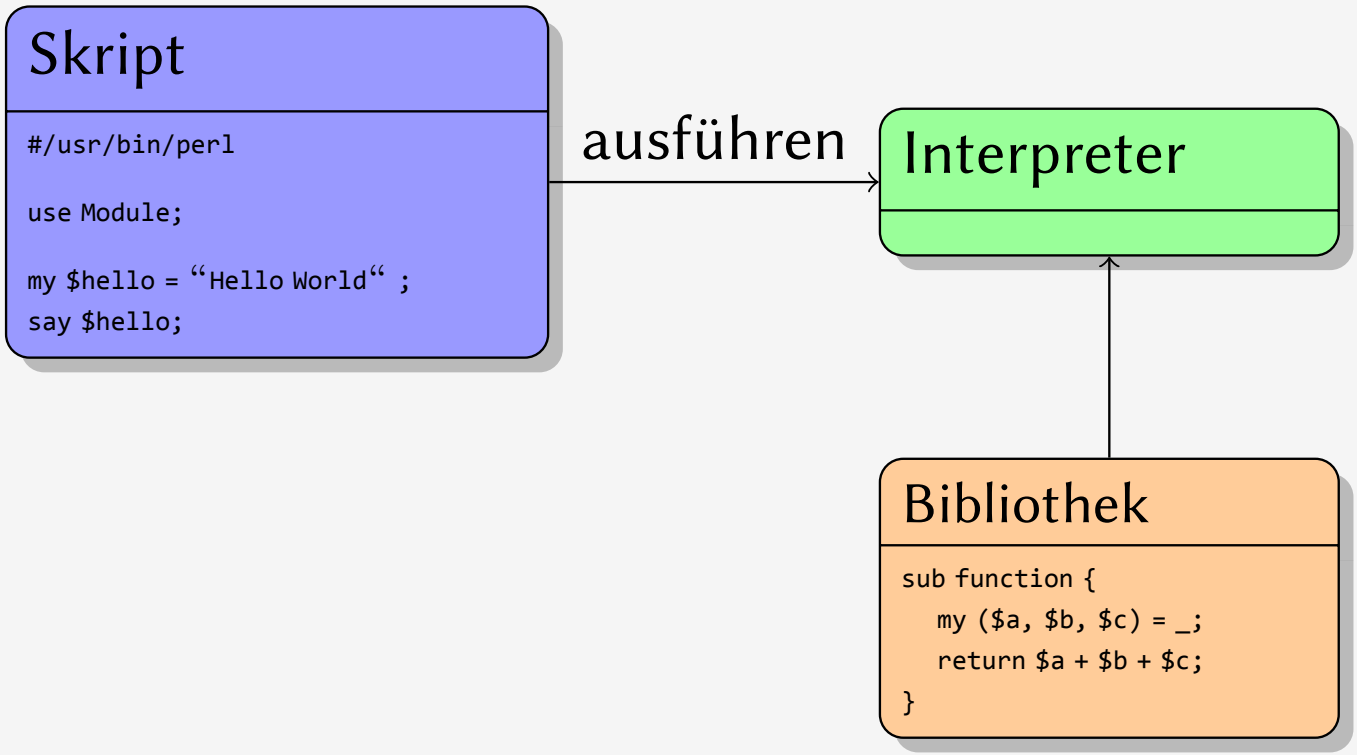
Skript

```
#/usr/bin/perl

use Module;

my $hello = “Hello World“ ;
say $hello;
```

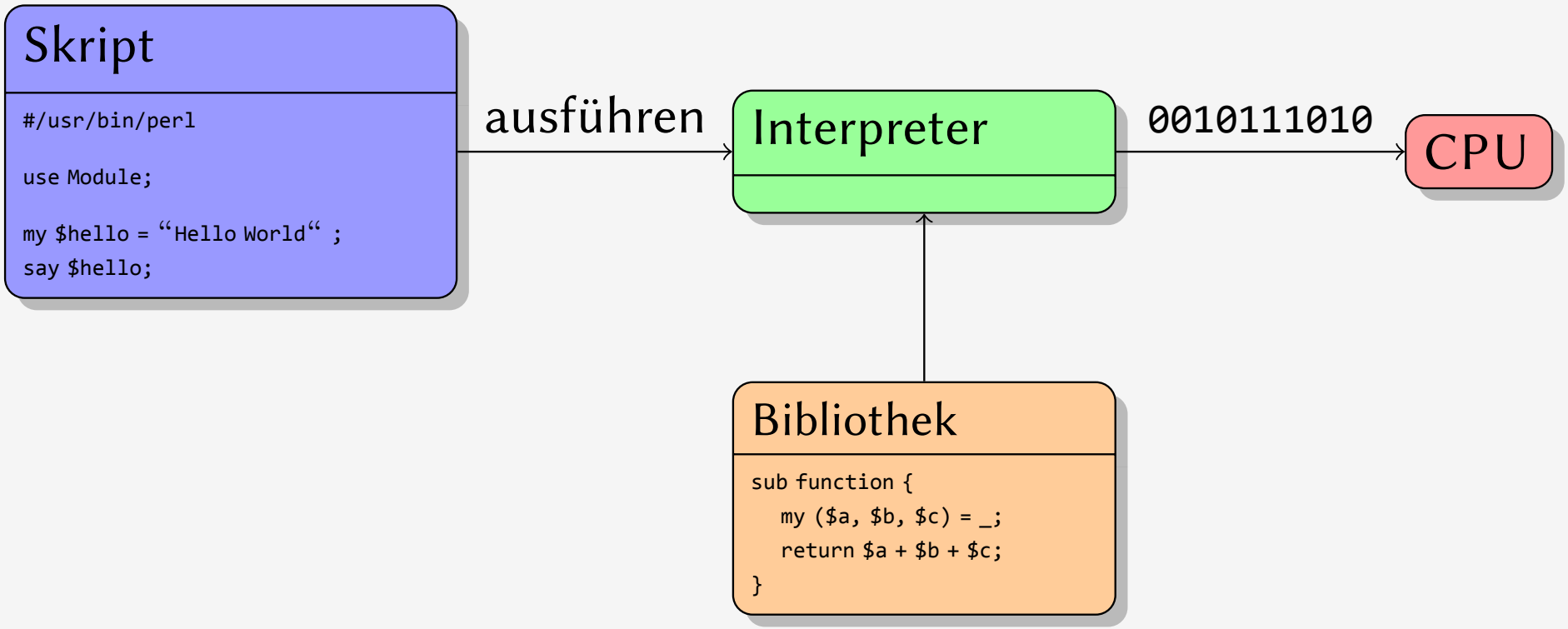
Interpreter



Interpreter



Interpreter



Interpreter Vor- und Nachteile

Vorteile

- kein Compilieren notwendig
- kein selbstständiges Speichermanagement notwendig
- einfacher zu programmieren
- Betriebssystem-unabhängig, wenn Interpreter für das Betriebsystem zur Verfügung steht

Nachteile

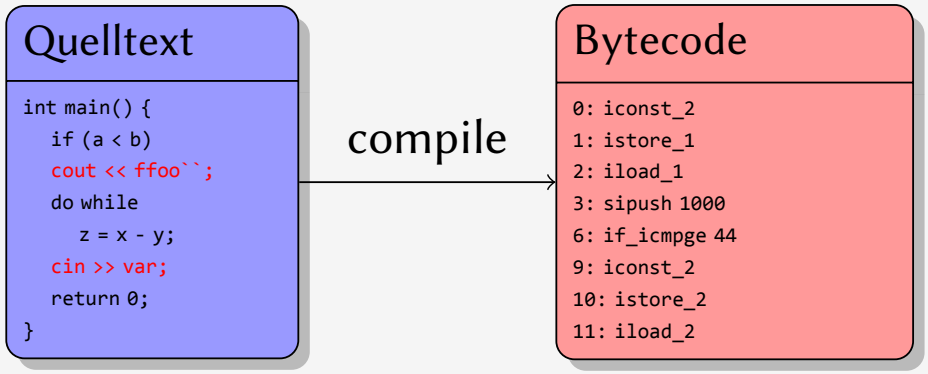
- deutlich langsamer als kompilierter Code

Virtuelle Maschine

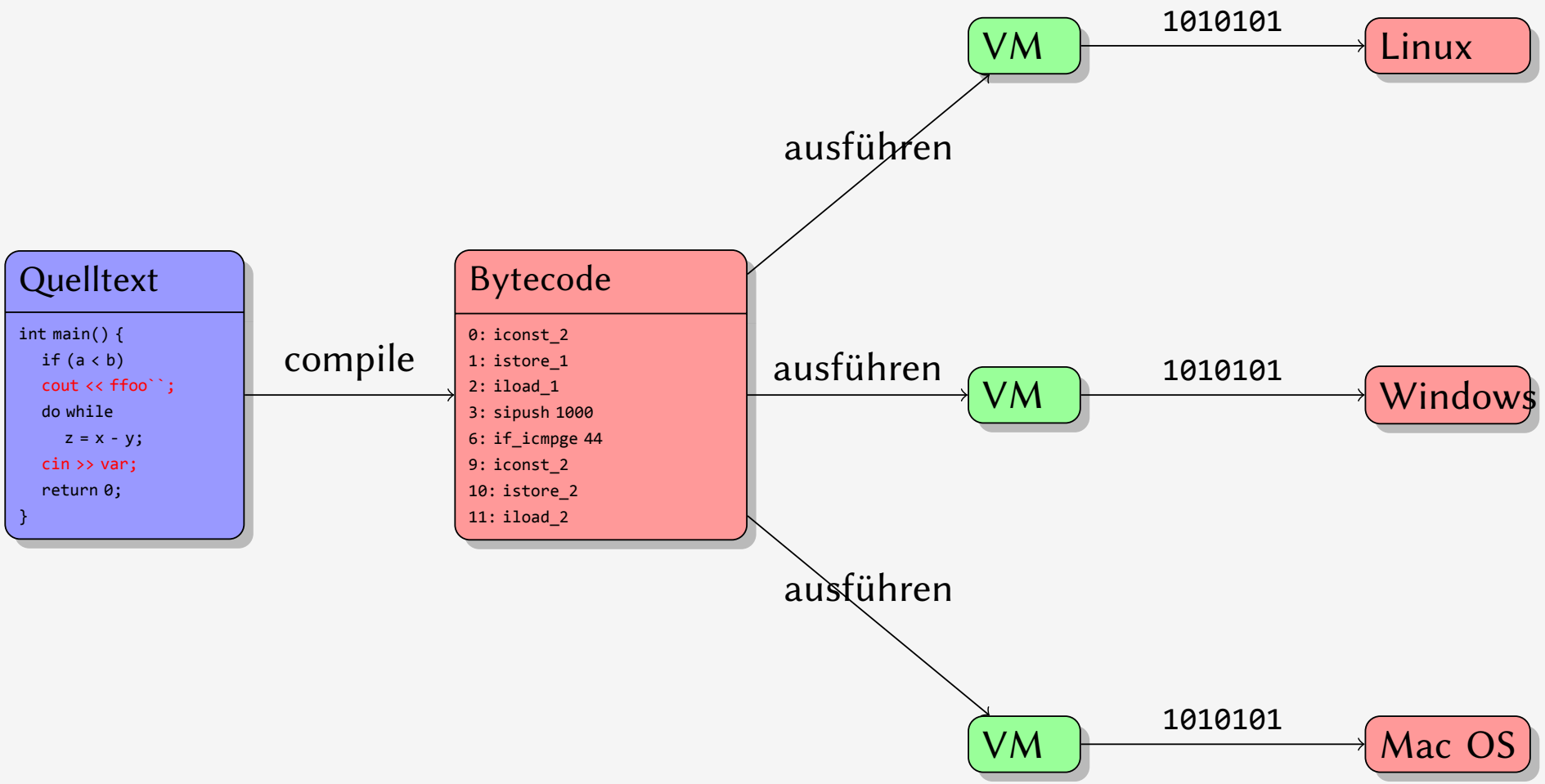
Quelltext

```
int main() {
  if (a < b)
  cout << ffoo``;
  do while
    z = x - y;
  cin >> var;
  return 0;
}
```

Virtuelle Maschine



Virtuelle Maschine



VM Vor- und Nachteile

Vorteile

- schneller als Interpreter-Sprache, weil vorcompiliert
- einfacher zu programmieren
- Betriebssystem-unabhängig, wenn VM für das Betriebsystem zur Verfügung steht

Nachteile

- langsamer als C

Table of Contents

Entwicklungsumgebung

IDE

- Code::Blocks
  `http://www.codeblocks.org/`
- Visual C++ Express
  `http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express`
- Eclipse CDT
  `http://www.eclipse.org/cdt/`

Texteditor

- Sublime Text 2
  `http://www.sublimetext.com/2`
- Notepad++
  `http://notepad-plus-plus.org/`
- Emacs
  `http://www.gnu.org/software/emacs/`