

# Deskriptive Programmierung

SS 2015

**Jun.-Prof. Dr. Janis Voigtländer**  
**Institut für Informatik III**  
**Universität Bonn**

## Zeiten im SS 2015

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
8					
9					
10	Vorlesung		Übung		Vorlesung
11					
12					
13					
14					
15					
16					

- **Übungen:** **Mi 10:15 – 11:45** (Beginn: vorauss. **22.04.2015**)
  - Kriterien für **erfolgreiche Teilnahme** (und damit Zulassung zur Prüfung):
    - Lösen einer Eingangs-Programmieraufgabe
    - regelmäßige Einreichung von Lösungen für gekennzeichnete Aufgaben
    - 50% der bei diesen erreichbaren Punkte (zu zwei Stichtagen)
    - außerdem 25% pro Übungsblatt
    - Genaues/Details, siehe Aushang vor Prüfungsamt!
- **Webseite(n)** zur Vorlesung als Hauptkommunikationsmedium:
  - „News of the Day“ (**Bitte regelmäßig checken!**)
  - **Folien** (als PDF-Dateien) zum Download (jeweils nach der Vorlesung)
  - weiterführende **Literaturangaben**, Links etc.
  - Links zu benötigter **Software**

<http://www.iai.uni-bonn.de/~jv/teaching/dp/>

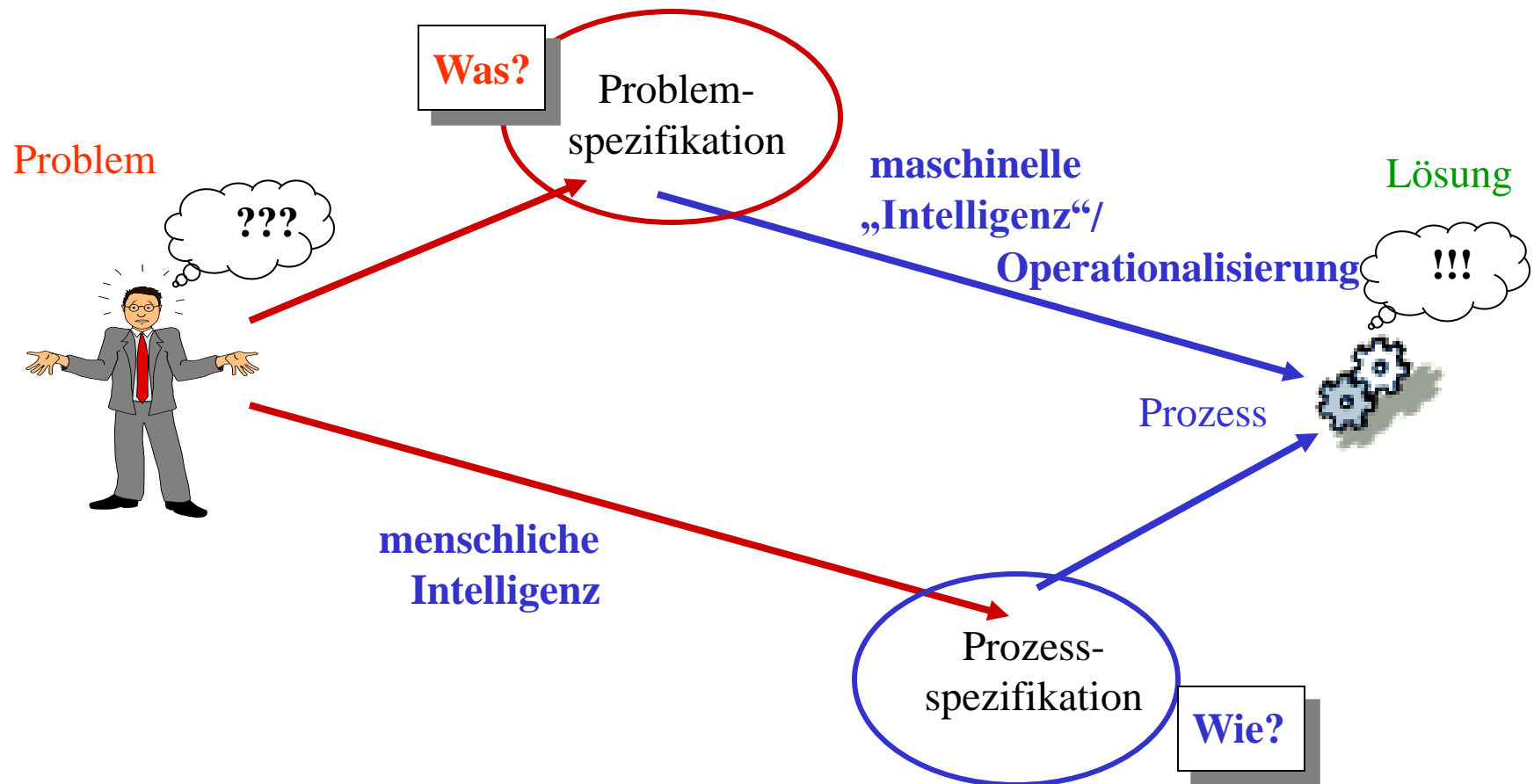
[https://ecampus.uni-bonn.de/goto\\_ecampus\\_crs\\_607168.html](https://ecampus.uni-bonn.de/goto_ecampus_crs_607168.html)

# Deskriptive Programmierung

## Einführung und Motivation

# Ideal (und ein Stück weit, Historie) der deskriptiven Programmierung

„Befreiung“ des Menschen von der Notwendigkeit, zur Problemlösung führende Rechenprozesse explizit zu planen und zu spezifizieren: **„Was statt Wie“**



Die **deklarative Programmierung** ist ein Programmierparadigma, welches auf mathematischer, rechnerunabhängiger Theorie beruht.

Zu den deklarativen Programmiersprachen gehören:

- **funktionale** Sprachen (u.a. LISP, ML, Miranda, Gofer, Haskell)
- **logische** Sprachen (u.a. Prolog)
- **funktional-logische** Sprachen (u.a. Babel, Escher, Curry, Oz)
- **Datenflusssprachen** (wie Val oder Linda)

(aus Wikipedia, 07.04.08)

- In der Regel erlauben deklarative Sprachen in irgendeiner Form die Einbettung **imperativer** Programmteile, mehr oder weniger direkt und/oder „diszipliniert“.
- Andere Programmiersprachenkategorien, einigermaßen orthogonal zu dekl./imp.:
  - **Objektorientierte** oder **ereignisorientierte** Sprachen
  - **Parallelverarbeitende/nebenläufige** Sprachen
  - **Stark** oder **schwach**, **statisch** oder **dynamisch**, oder **gar nicht** getypte Sprachen

## Charakteristika deskriptiver Spezifikationen (vs. imperativer Programme)

- Deskriptive Programme (Spezifikationen) sind oft
  - signifikant **kürzer**
  - signifikant **lesbarer**
  - signifikant **wartbarer** (und **zuverlässiger**)als ihre imperativen „Gegenstücke“.
- Insbesondere funktionale Sprachen betonen Abstraktionen, die Seiteneffekte für Programmteile ausschließen oder gezielt (und flexibel) unter Kontrolle halten.  
(S. Peyton Jones: „Haskell is the world’s finest imperative programming language.“)
- Deskriptive Konzepte eignen sich besonders gut zur Realisierung/Einbettung domänenspezifischer Sprachen (DSLs).
- **aber:**
  - Deskriptive Sprachen sind noch **weniger verbreitet** als imperative Sprachen.
  - Die **Produktentwicklung** für das Arbeiten mit deskriptiven Sprachen ist nicht so weit fortgeschritten.
  - **Beschränkungen** in der Anwendung liegen oft (Annahmen über, oder tatsächlich) mangelhaft effiziente Operationalisierungsmethoden zu Grunde.

- Kommerzielle Anwender:
  - im Bankensektor (Trading, Quantitative Analysis), z.B. Barclays Capital, Jane Street Capital, Standard Chartered Bank, McGraw Hill Financial, ...
  - im Bereich Communication/Web Services, z.B. Ericsson, Facebook, Google
  - Hardware-Design/Verification, z.B. Intel, Bluespec, Antiope
  - System-Level Development, z.B. Microsoft
  - High Assurance Software, z.B. Galois

<http://cufp.org/>

<http://groups.google.co.uk/group/cu-lp>

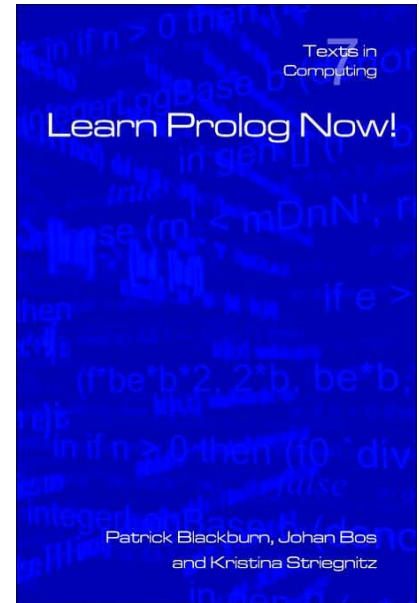
- „nicht-akademische“ Sprachen:
  - für spezielle Anwendungsgebiete, z.B. Erlang (Ericsson), reFLect (Intel)
  - für allgemeine Anwendungen, z.B. F# (Microsoft)
  - Einfluss auf Mainstream-Sprachen, z.B. Java, C#, und „sogar“ Visual Basic (allgemein: LINQ-Framework)





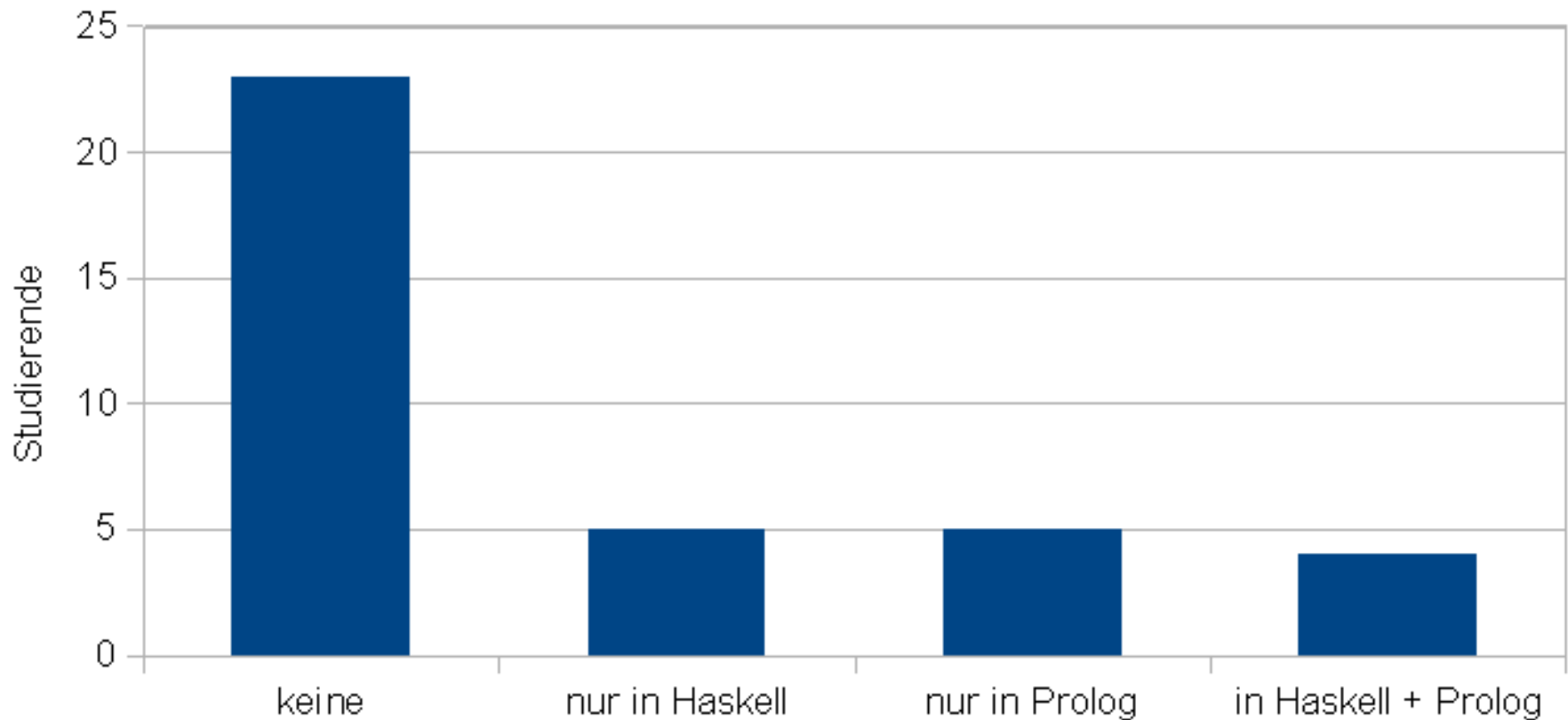
Marco Block, Adrian Neumann:  
„Haskell-Intensivkurs“  
Springer-Verlag, 2011

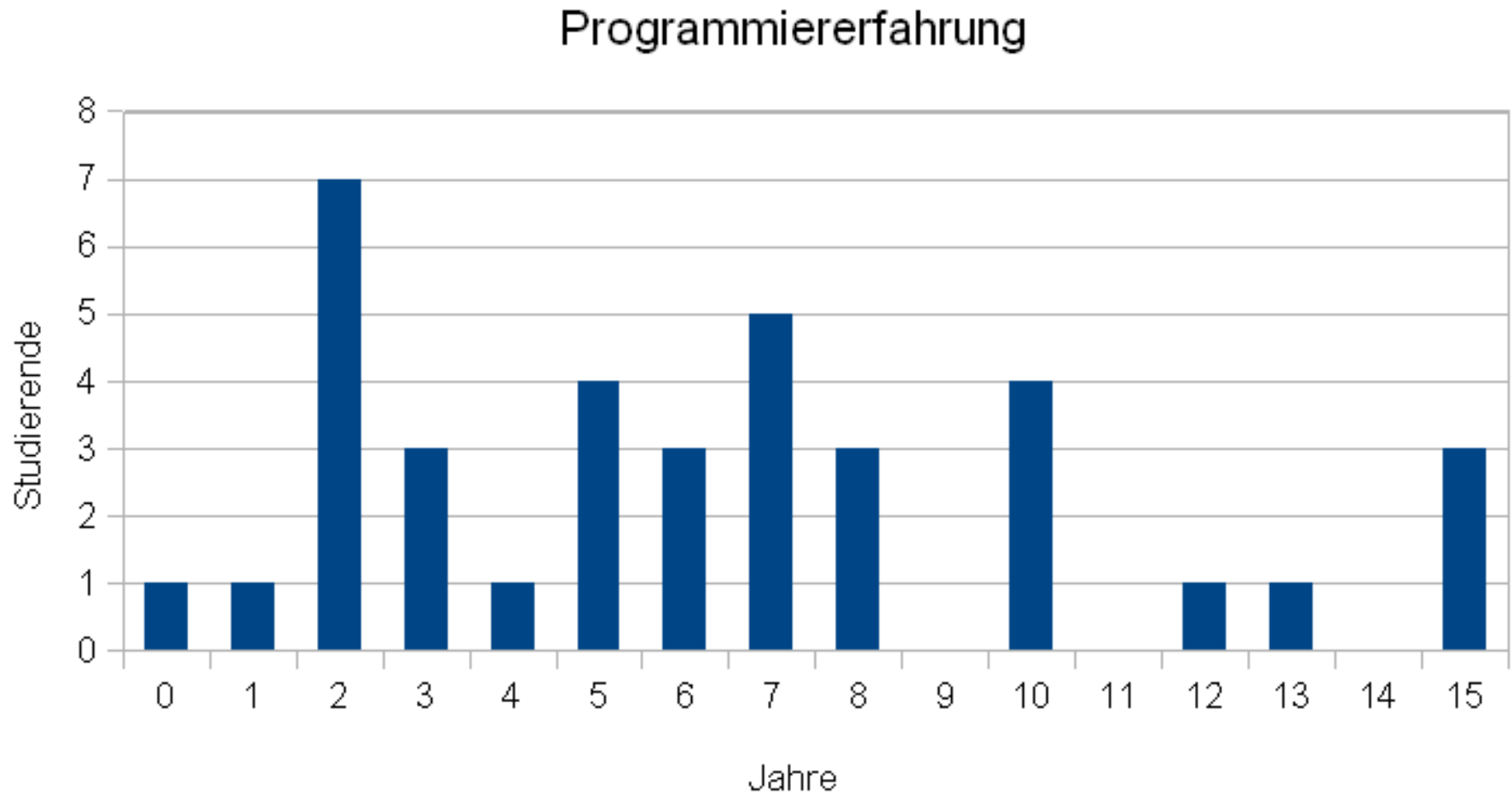
Patrick Blackburn, Johan Bos,  
Kristina Striegnitz:  
„Learn Prolog Now!“  
College Publications, 2006

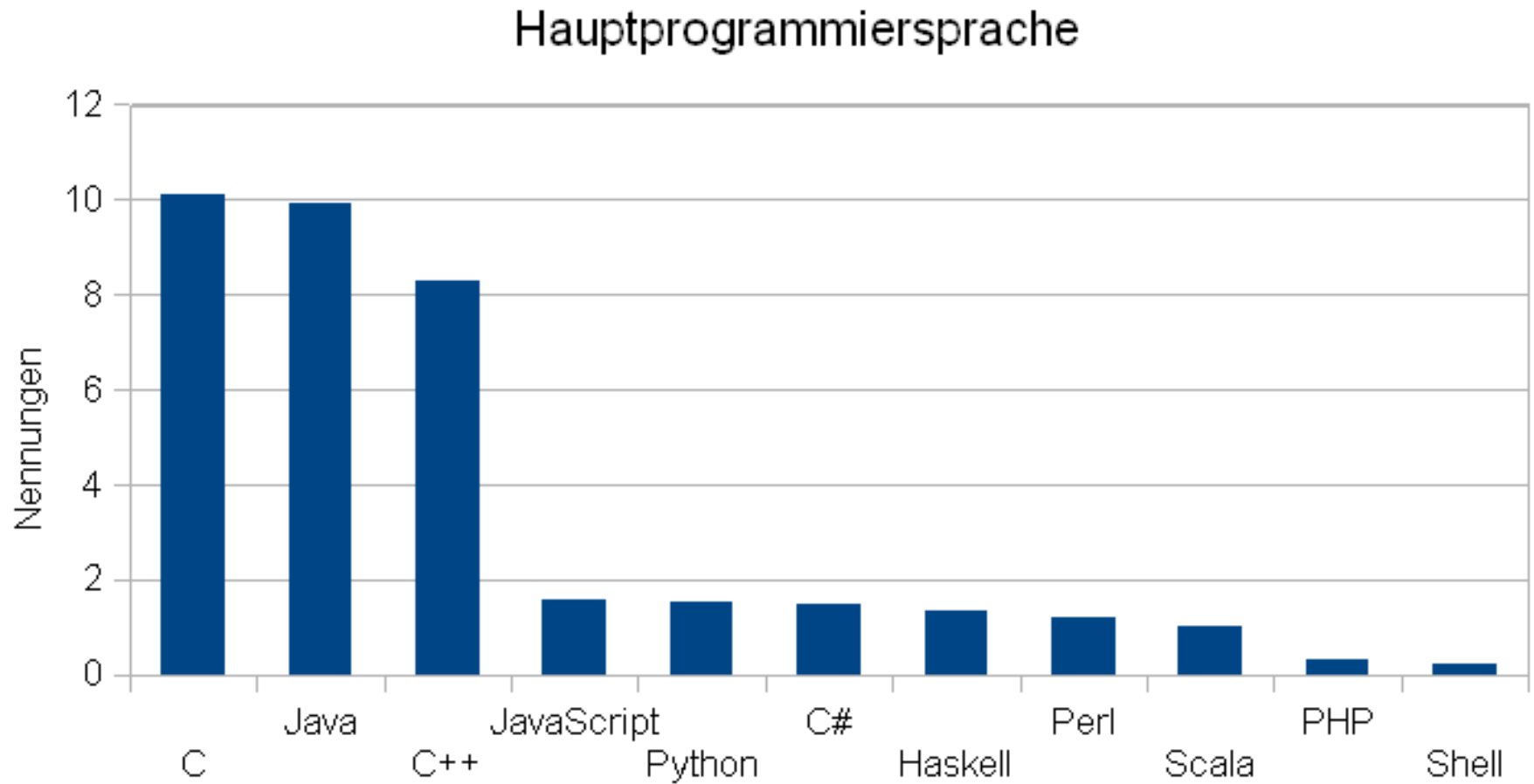


- In dieser Vorlesung: Haskell als funktionale, Prolog als logische Programmiersprache
- Obige Literatur nur Beispiele, weitere Hinweise auf Webseite zur Vorlesung

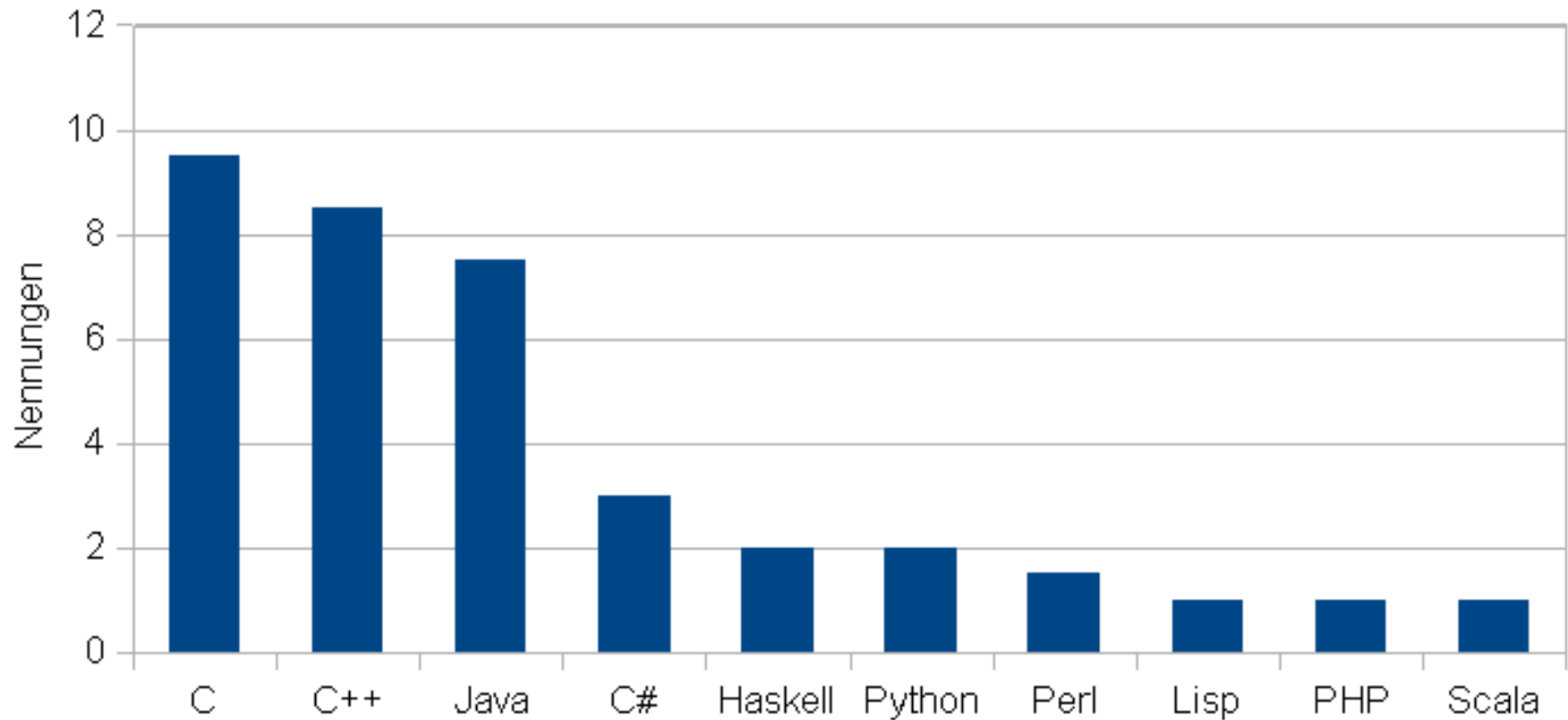
### Vorkenntnisse in Haskell oder Prolog



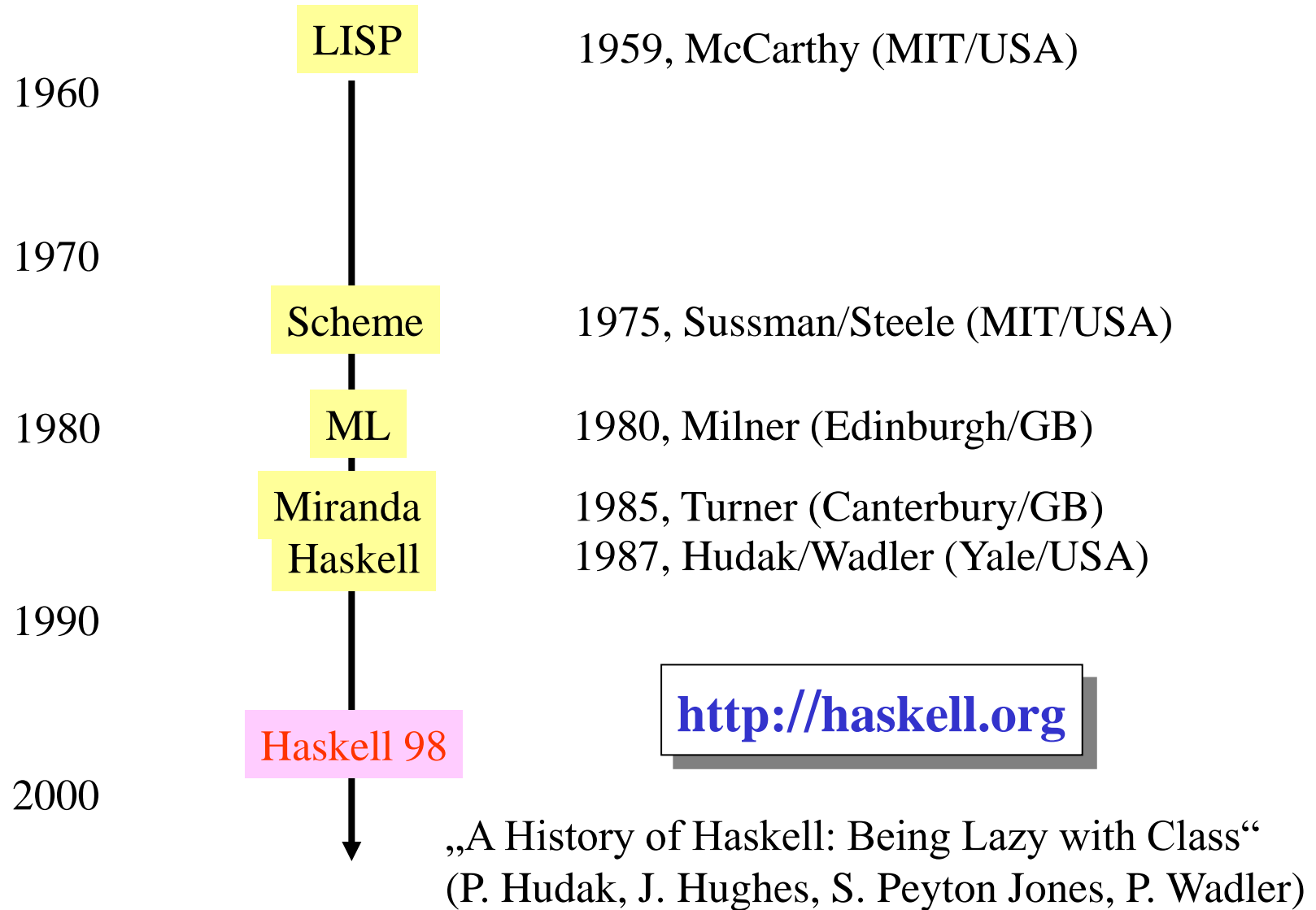




### "Lieblingsprogrammiersprache"



## Wichtige funktionale Sprachen im historischen Überblick



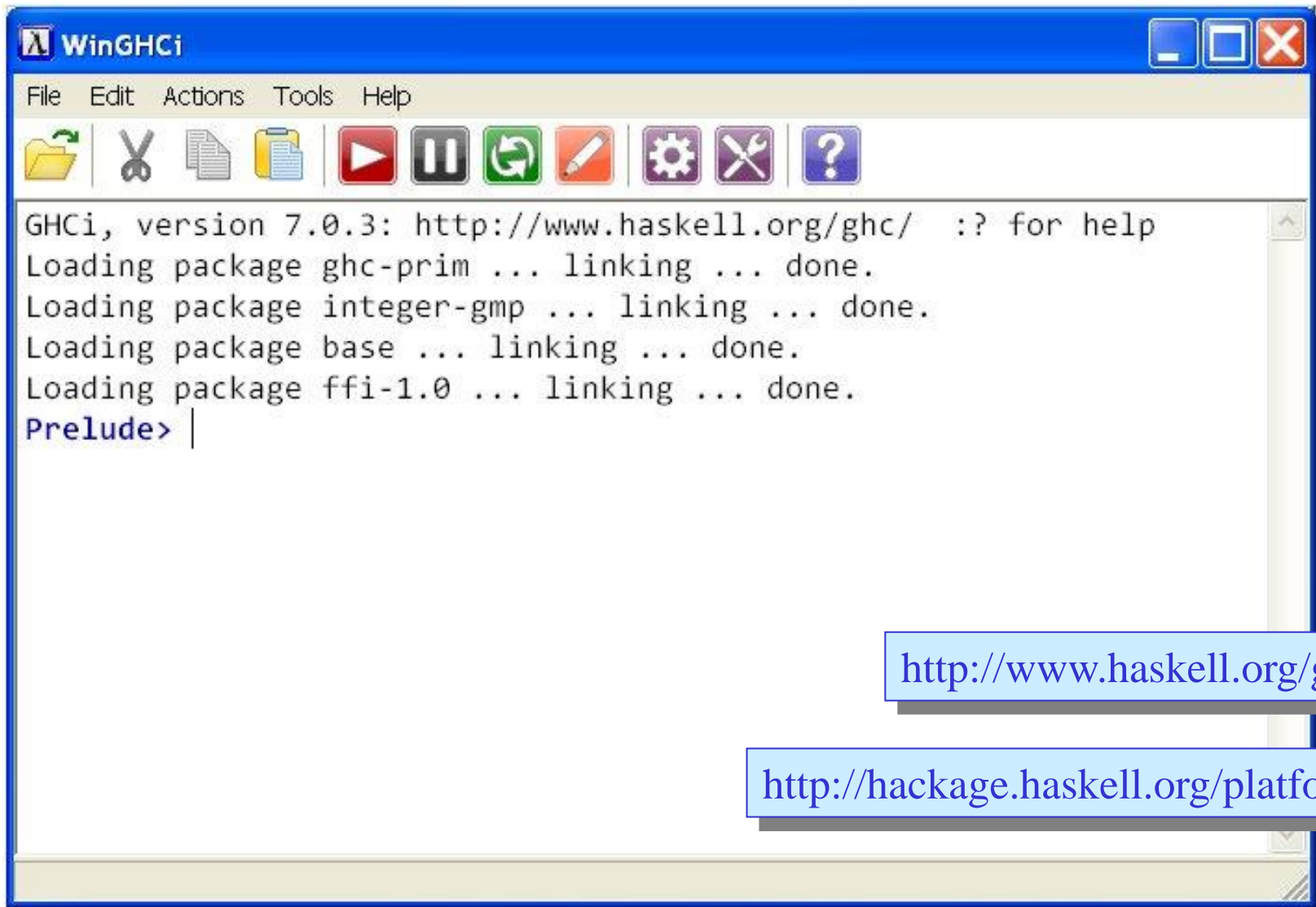
## Wofür steht „Haskell“?

- Namen von Programmiersprachen sind oft **Akronyme**  
(z.B. COBOL, FORTRAN, BASIC, ...)
- Der Name „Haskell“ dagegen leitet sich von einer **Person** her:

**Haskell** Brooks Curry  
(1900 - 1982)  
amerikanischer Logiker



## Verwendete Implementierung: GHC(i)



The screenshot shows a window titled "WinGHCi" with a menu bar (File, Edit, Actions, Tools, Help) and a toolbar with icons for file operations and execution. The main text area displays the following output:

```
GHCi, version 7.0.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Loading package ffi-1.0 ... linking ... done.
Prelude> |
```

<http://www.haskell.org/ghc/>

<http://hackage.haskell.org/platform/>



# Deskriptive Programmierung

**Beispiele in Haskell eingebetteter DSLs**

## Beschreibung von Grafiken mittels „gloss“

- eine einfache Bibliothek (siehe Installationsanleitung auf Übungsblatt)
- Grundkonzepte:

Float, String, Path, Color, Picture

text :: String → Picture

line :: Path → Picture

polygon :: Path → Picture

arc :: Float → Float → Float → Picture

circle :: Float → Picture

...

color :: Color → Picture → Picture

translate :: Float → Float → Picture → Picture

rotate :: Float → Picture → Picture

scale :: Float → Float → Picture → Picture

pictures :: [ Picture ] → Picture

- Verwendung in konkretem „Programm“:

```
module Main (main) where

import Graphics.Gloss

main = display (InWindow "Bsp" (100, 100) (0,0)) white scene

scene = pictures
  [
    circleSolid 20
    , translate 25 0 (color red (polygon [(0,0),(10,-5),(10,5)]))
  ]
```

- Let's play a bit. ...