Konzepte von Programmiersprachen

1. Einführung

Ziele der Vorlesung:

- **Entwurfsalternativen von Programmiersprachen erkunden**
- **Deklarative Programmierung am Beispiel von Haskell erlernen**
- **Programmiertechniken und ihre Verbindung zu Sprachkonzepten kennenlernen und verstehen**

Höhere Programmiersprachen

- **Ziele beim Entwurf:**

  Ausdrucksmächtigkeit, Schlichtheit, Eleganz

- **Weitere Anforderungen:**
  - **Universalität**
  - **Implementierbarkeit**
  - **Verifizierbarkeit**
  - **Effizienz -> ... bei Ausführung & ... bei Programmierung**

Konzepte

- **Werte, Speicher, Bindungen**
- **Abstraktion und Kapselung**
- **Typsysteme**
- **Ablaufsteuerung und Nebenläufigkeit**
- **...**

Ziele

- **Erlernen neuer Sprachen**
- **Auswahl geeigneter Sprachen**
- **Entwicklung verlässlicher wohlstrukturierter Programme**
- **Unterscheidung verschiedener Paradigmen**

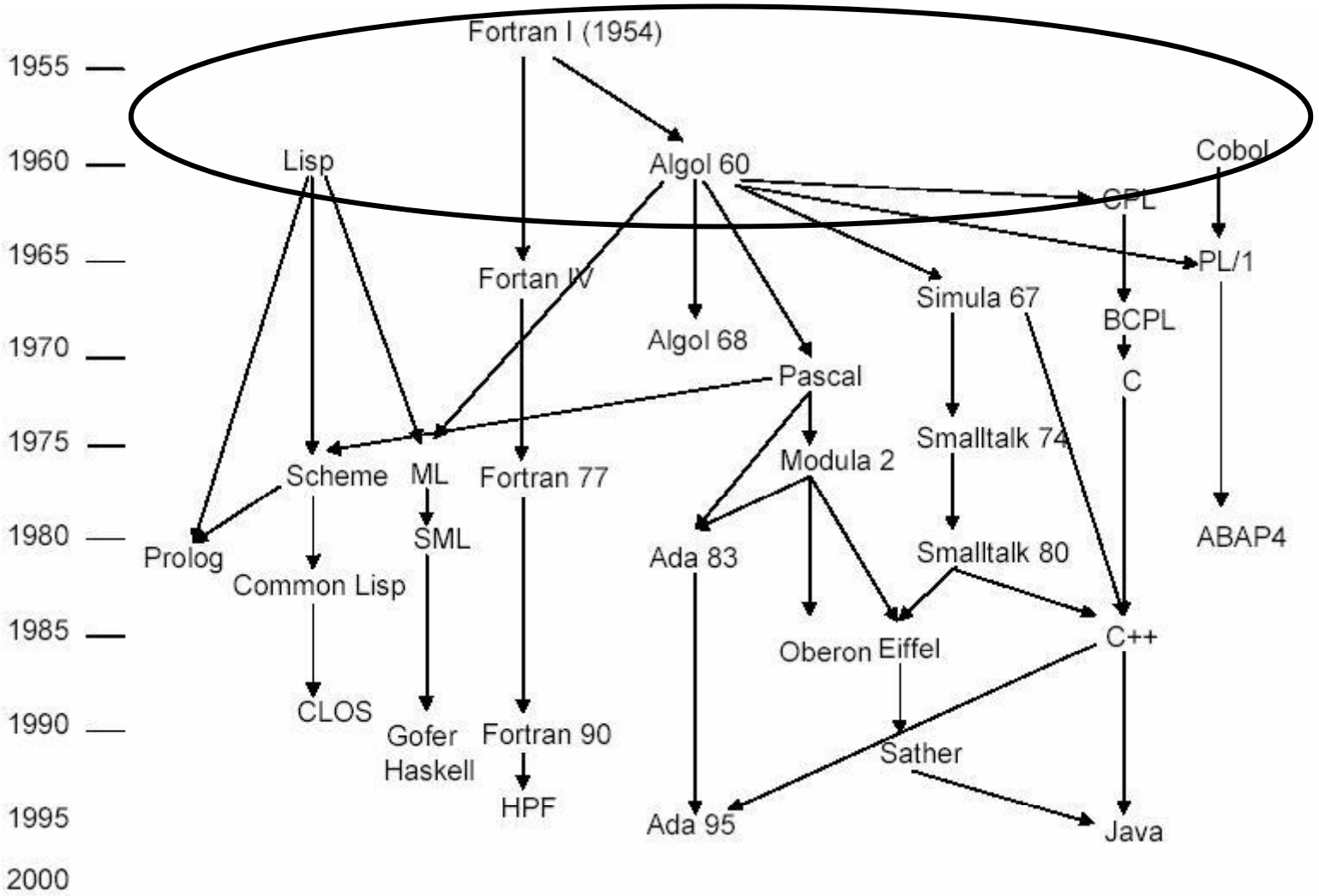
Einige Zitate

- **Karl der Große (742-814):**
  „To know another language is to have a second soul.“

- **Ronald Searle (1920-):**
  „You can never understand one language until you understand at least two.“

- **Ralf Hinze, Uni Bonn: Seit 1950 sind Hunderte von Programmiersprachen entwickelt worden.**
  ...
  Programmiersprachen sind das Medium der Informatik.
  Ein tiefes Verständnis der zugrundeliegenden Konzepte ist für alles, was mit Informatik zu tun hat, unabdingbar.

Historie


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran I (1954)
Lisp
Algol 60
Cobol
CPL
PL/1
Fortan IV
Simula 67
BCPL
Algol 68
C
Pascal
Smalltalk 74
Modula 2
Scheme
ML
Fortran 77
SML
ABAP4
Prolog
Ada 83
Smalltalk 80
Common Lisp
C++
Oberon
Eiffel
CLOS
Gofer
Haskell
Fortran 90
Sather
HPF
Ada 95
Java

Fortran (Formula Translator)

- Entwicklung 1954-56 bei IBM unter John Backus
- Standard für numerische Berechnungen (scientific computing)
- Hauptinnovationen:
  - math. Notation für arithmetische Ausdrücke
  - symbolische Namen für Variablen, Unterprogramme, Felder, Dekl.
- Nachfolger heute noch weitverbreitet:
  Fortran 90, HPF (High Performance Fortran), Fortran 2003

```
C Hello World in Fortran
      WRITE (6,7)
 7    FORMAT (13HHello, world!)
      STOP
      END
```

COBOL (Common Business Oriented Language)

- **Entwicklung 1960 von Komitee unter Grace Murry Hopper**
- **Konzeption für Geschäftsanwendungen**
- **Hauptinnovation: Anlehnung an die natürliche Sprache**

```
* Hello World in COBOL.
IDENTIFICATION DIVISION.
   Program-Id. Hello-World.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
PARA1.
   DISPLAY „Hello, world!“.
   STOP RUN.
```

E.W. Dijkstra: „The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.“

LISP (List Processing)

- **Entwicklung Ende der 50er Jahre am MIT unter John McCarthy**
- **Anwendungsgebiete: Künstliche Intelligenz, Symbolische Berechnungen**
- **Hauptinnovationen: rekursive Funktionen, Funktionen höherer Ordnungen, Programme als Daten, S-Ausdrücke, Speicherverwaltung**

```
(PRINT (LIST ,HELLO ,WORLD))

(write-string „Hello, world!“)
```

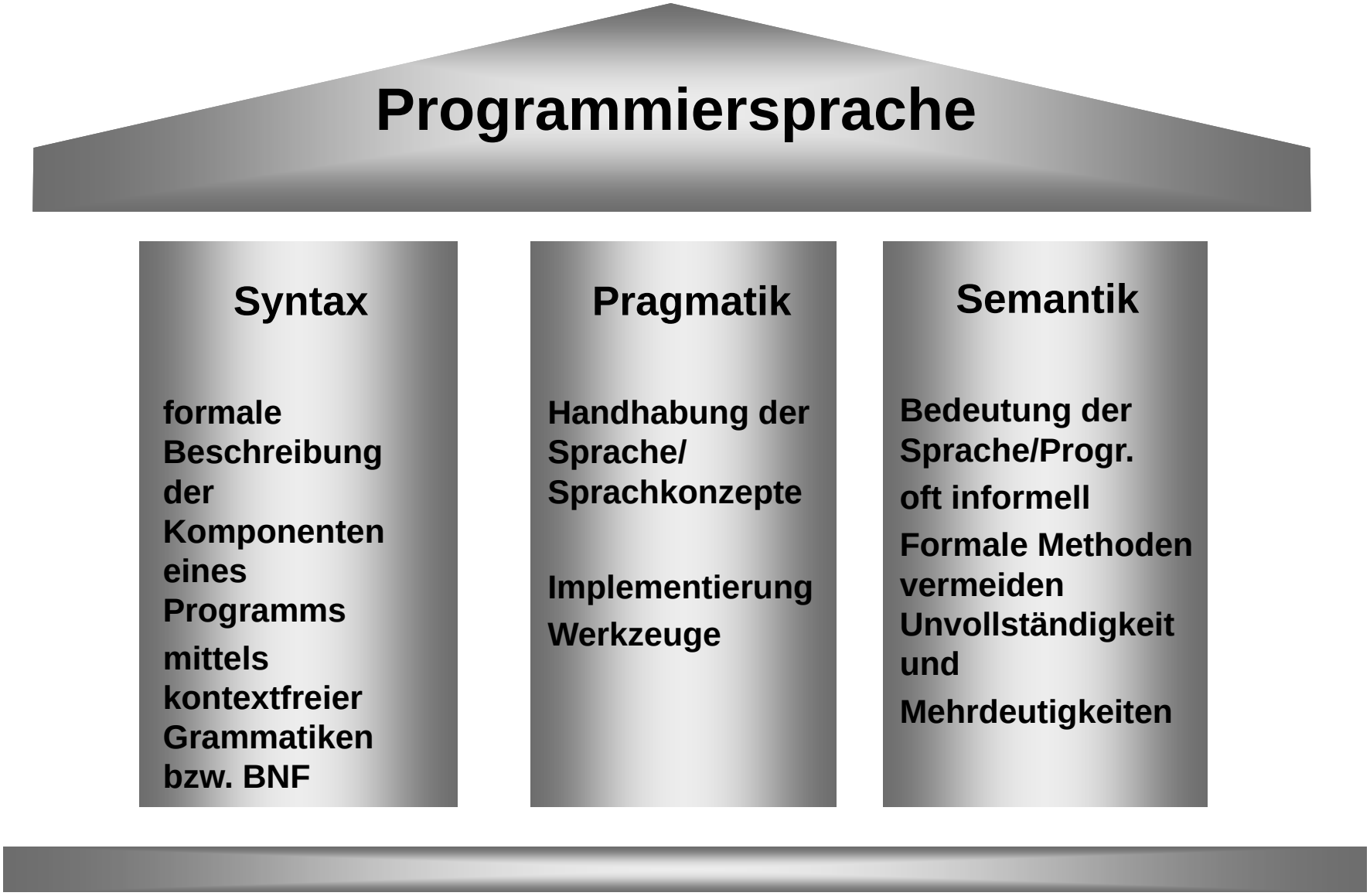
Alan Perlis: „A Lisp programmer konws the value of everything, but the cost of nothing.“

E.W. Dijkstra: „Lisp has jokingly been called `the most intelligent way to misuse a computer. I think that description is a great compliment...“

Algol (Algorithmic Language)

- Entwicklung von 1958-1963 von einem Komitee, dem u.a. John Backus, John McCarthy und Alan Perlis angehörten
- Ausrichtung: „general purpose“ Sprache, vor allem für wissenschaftliche Anwendungen
- Hauptinnovationen:
  - einfache Syntax für Anweisungen (;)
  - Blockstruktur (begin – end)
  - rekursive Funktionen (Stackprinzip)
  - einfaches Typsystem
  - Nebenprodukt: Backus-Naur-Form (BNF)

```
program HELLOWORLD;
begin
   print „Hello, world!“
end
```


Programmiersprache
Syntax
formale Beschreibung der Komponenten eines Programms
mittels kontextfreier Grammatiken bzw. BNF
Pragmatik
Handhabung der Sprache/ Sprachkonzepte
Implementierung
Werkzeuge
Semantik
Bedeutung der Sprache/Progr.
oft informell
Formale Methoden vermeiden Unvollständigkeit und
Mehrdeutigkeiten

Syntaxbeschreibung

kontextfreie Grammatik = < Nonterminale, Terminale, Startsymbol, Regeln >

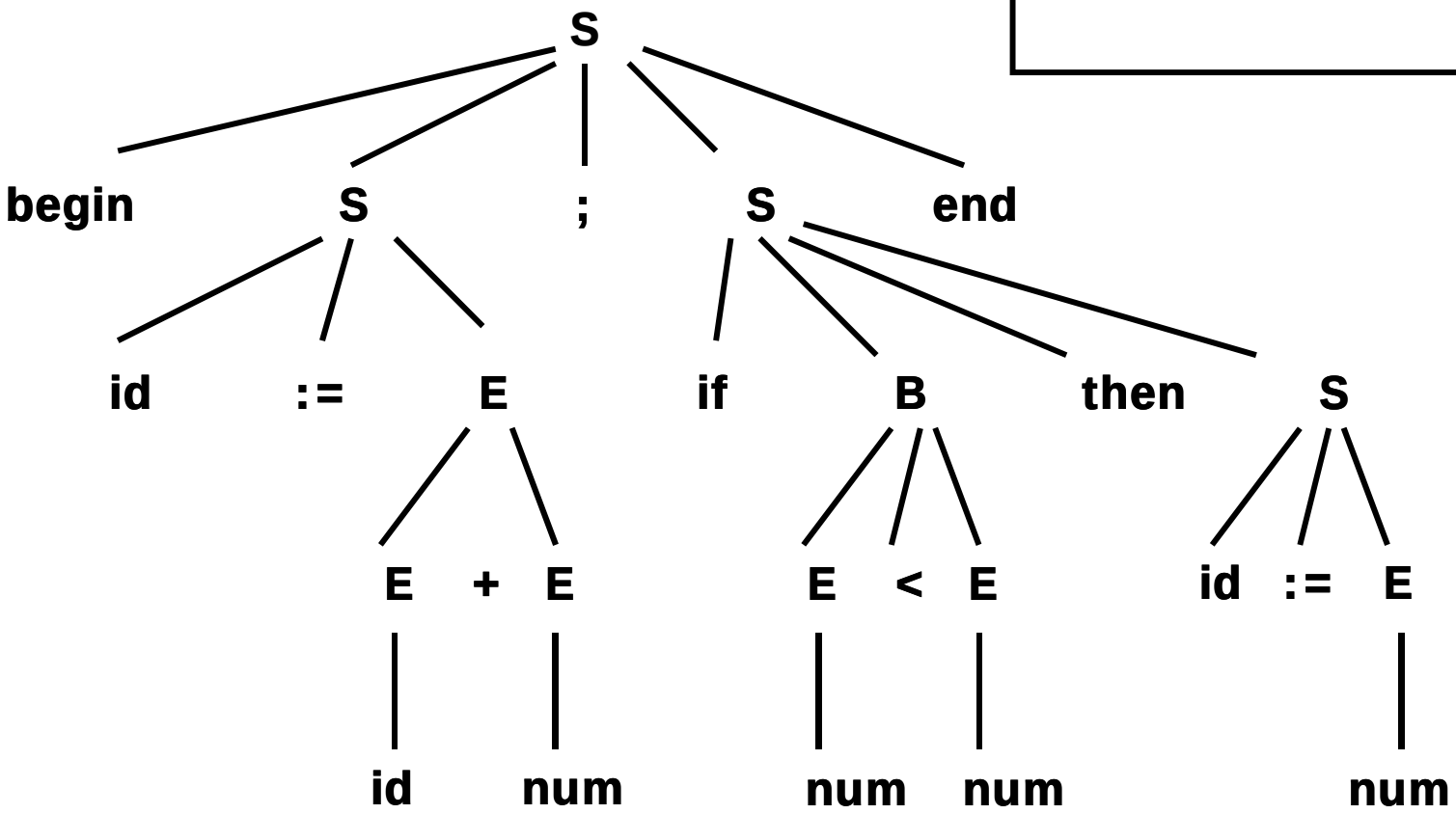
```
% Anweisungen (Statements)
 S  ->     id := E                  % Wertzuweisung
    |      begin  S ; S end         % Hintereinanderausführung
    |      if  B then  S            % bedingte Anweisung

% Ausdrücke (Expressions)
 E ->      E + E                    % Summe
    |      id                       % Bezeichner
    |      num                      % Zahl

% Boolesche Ausdrücke (Boolean expressions)
 B ->      E < E                    % Vergleichsausdruck
```

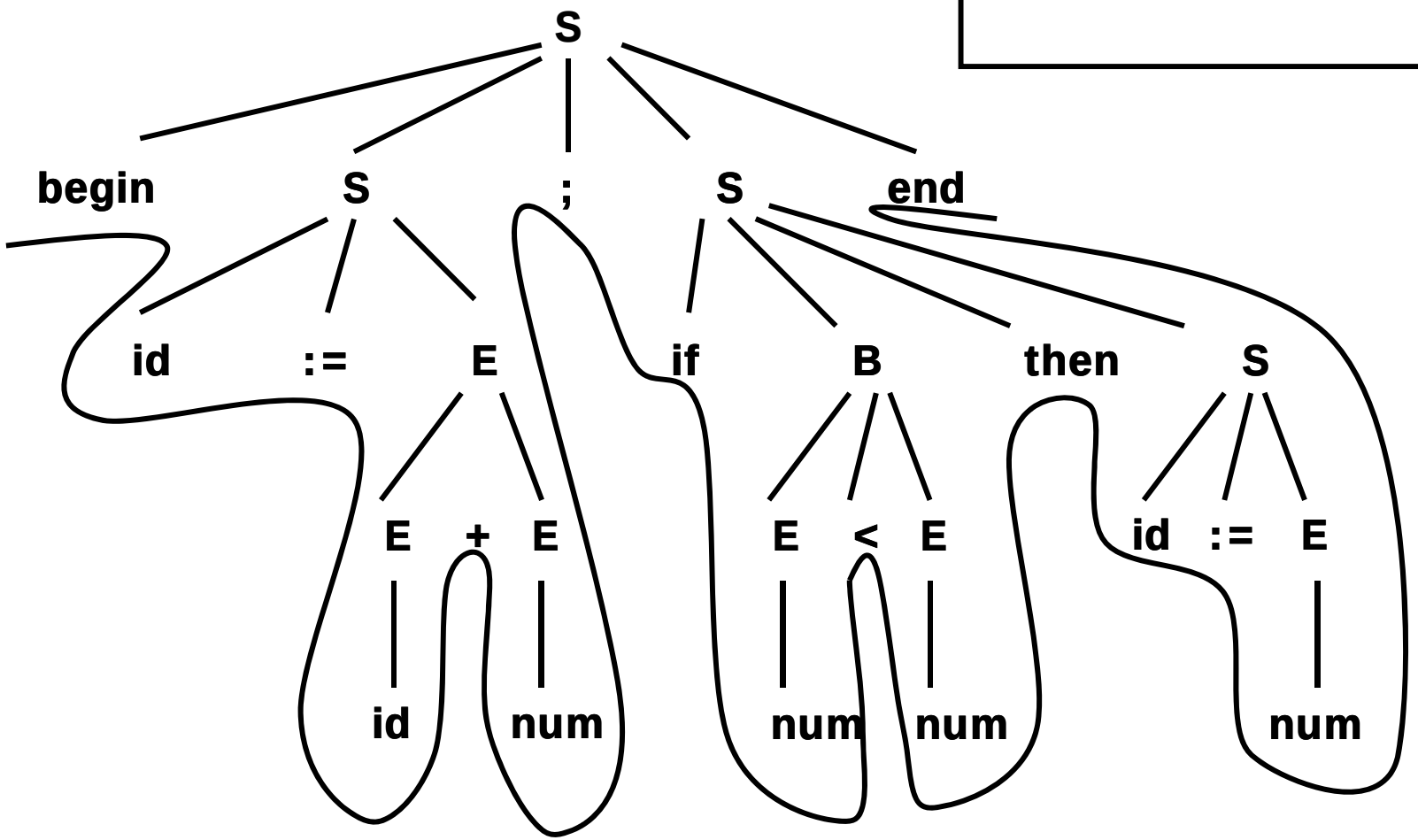
Syntaxbaum

```
S  ->   id := E
    |   begin  S ; S end
    |   if  B then  S
E ->    E + E   |  id  | num
B ->    E < E
```



Syntaxbaum - Front

```
S  ->   id := E
    |   begin  S ; S end
    |   if  B then  S
E ->    E + E   |  id  | num
B ->    E < E
```



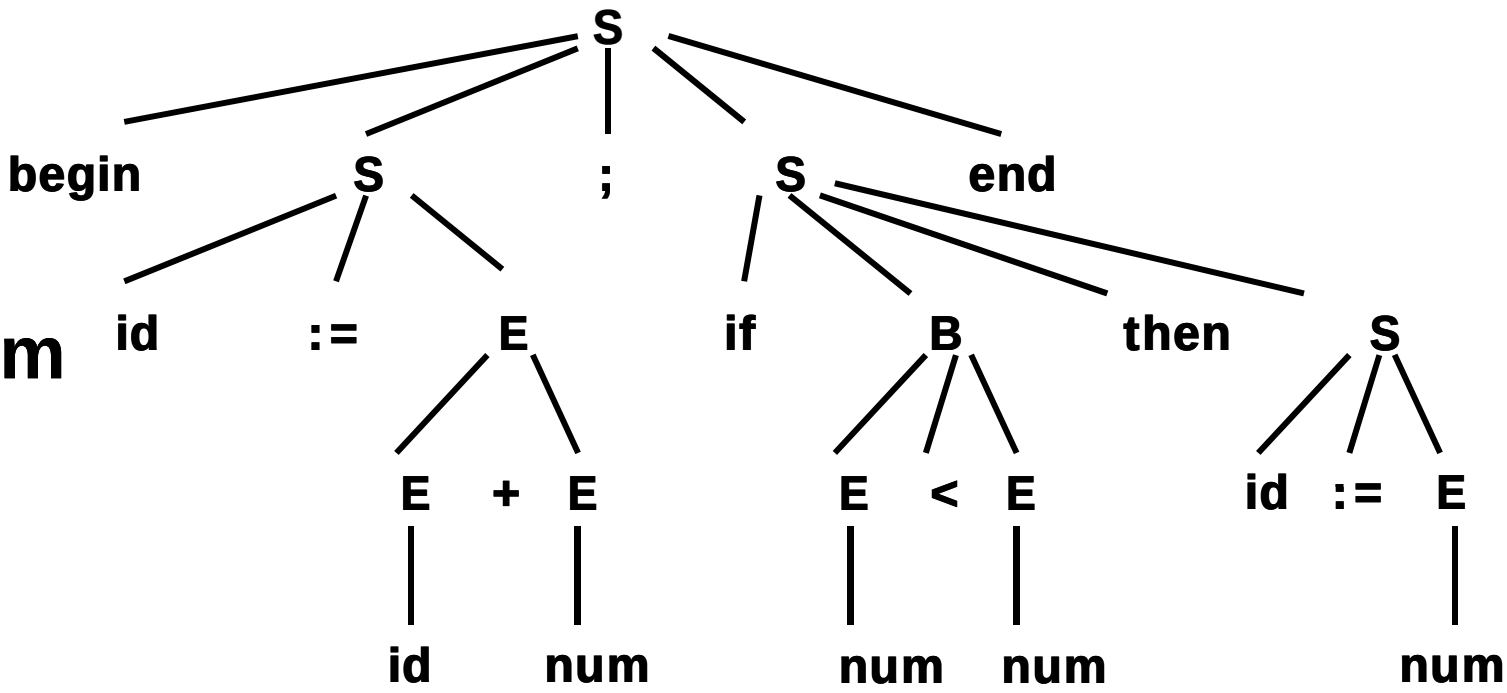

```
begin  id  :=  id  +  num  ; if  num  <  num  then  id  := num  end
```

Abstrakte Syntax

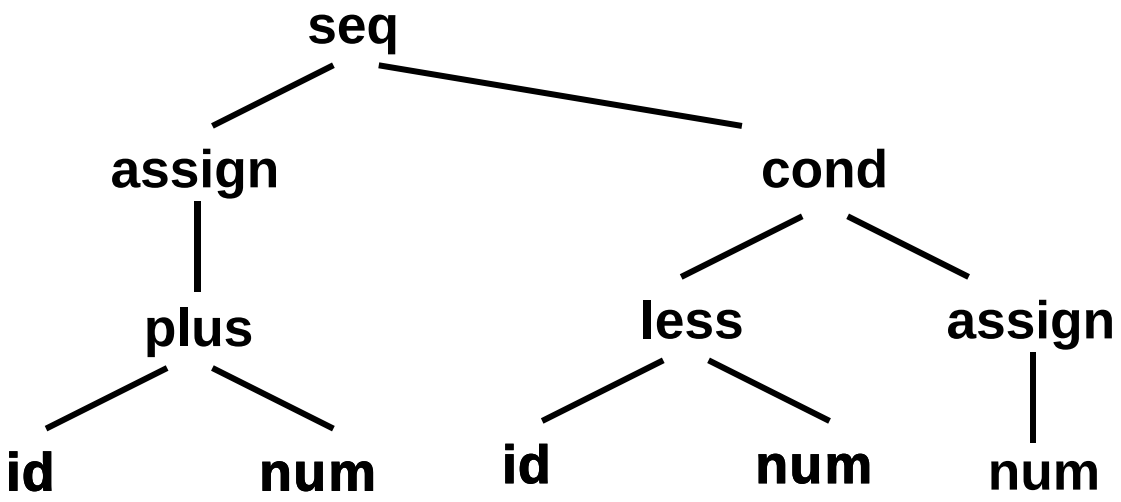
Regeln => Operationssymbole

S ->	id := E	=>	assign : E –> S
|	begin S ; S end	=>	seq : S x S –> S
|	if B then S	=>	cond : B x S –> S
E ->	E + E	=>	plus : E x E –> E
|	id	=>	id : –> E
|	num	=>	num : –> E
B ->	E < E	=>	less : E x E –> B

Konkreter Syntaxbaum



Abstrakter Syntaxbaum



Beispiel: Konzept Zählschleife

Abstrakte Syntax: for(index,anfang,ende,rumpf)

- **FORTRAN**

```
      do 1 i=1,10
1     a(i) = 0
```

- **Pascal**

```
for i:=1 to 10 do
  a(i) := 0;
```

- **C**

```
for (i=1; i<=10; i++)
   a(i)=0;
```

Beispiel: Konzept Zählschleife

Abstrakte Syntax: for(index,anfang,ende,rumpf)

- **FORTRAN**

```
      do 1 i=1,10
1     a(i) = 0
```

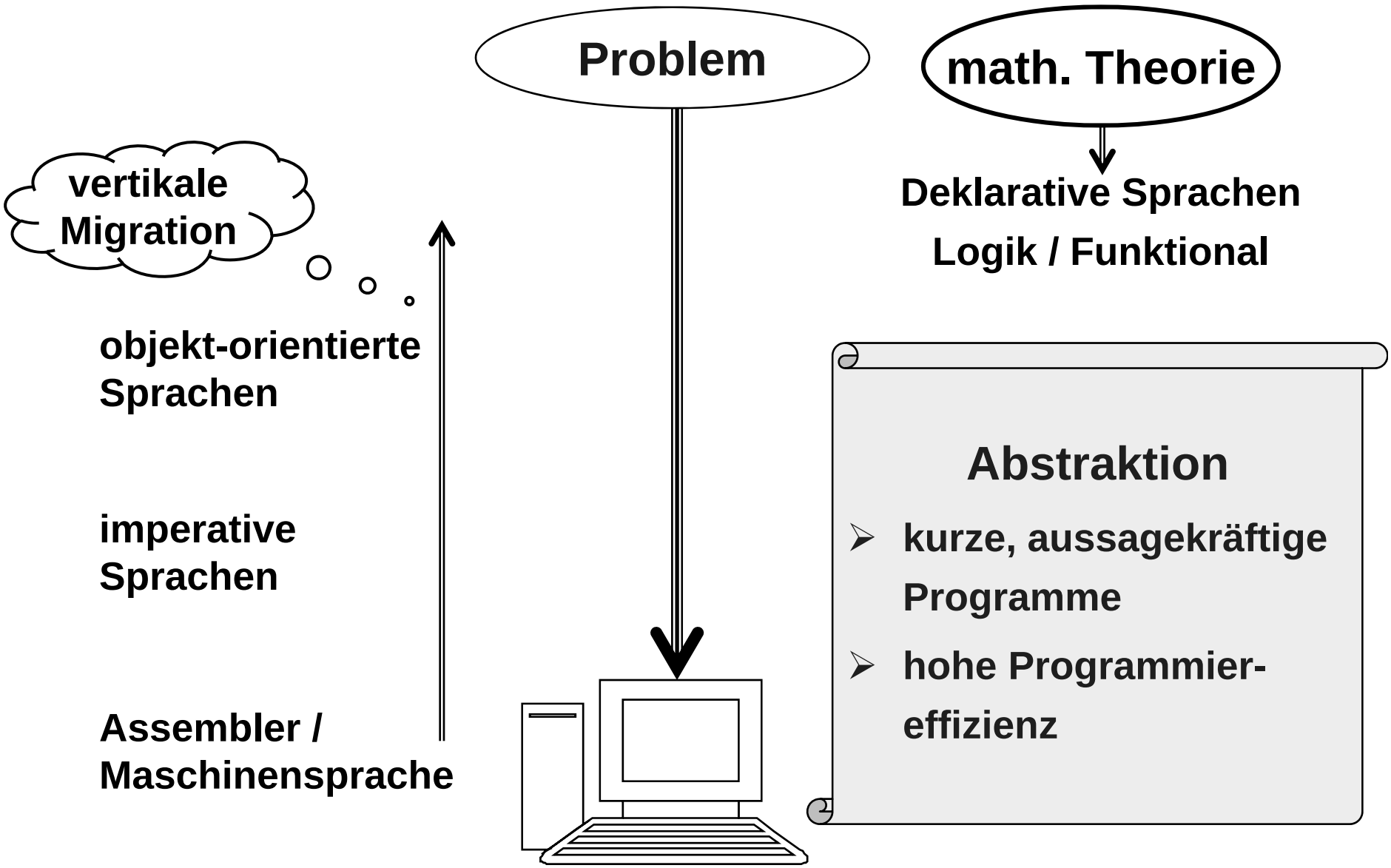
- **Pascal**

```
for i:=1 to 10 do
  a(i) := 0;
```

- **C**

```
for (i=1; i<=10; i++)
   a(i)=0;
```

Klassifikation von Programmiersprachen

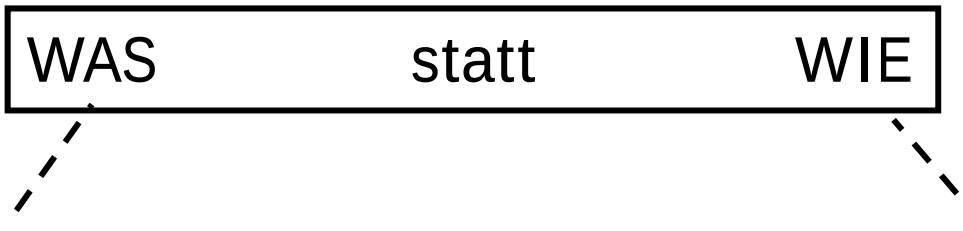


Programmierparadigmen

- **imperative Programmierspachen**
  - **lat. imperare = befehlen**
  - **Programme = Abfolge von Befehlen**
  - **Abstraktion von Maschinensprachen**
  - **prozedurale Sprachen wie FORTRAN, PASCAL, MODULA, ADA, C, OCCAM etc.**
  - **objektorientierte Sprachen wie SMALLTALK, Eiffel, C++, Java etc.**

- **deklarative Programmiersprachen**
  - **lat. declarare = erklären**
  - **Programme = Problemspezifikationen**
  - **Basis: Mathematische Theorie**
  - **funktionale Sprachen wie LISP, ML, Miranda, Haskell**
  - **Logik-Sprachen wie Prolog**

Grundidee deklarativer Sprachen



Logik
Problemspezifikation

Kontrolle
Auswertung

intellektuell
anspruchsvoll

automatisierbar

Referentielle Transparenz

Der Wert eines Ausdrucks hängt nur von seiner Umgebung und nicht vom Zeitpunkt seiner Auswertung ab. Deshalb kann ein Ausdruck immer durch einen anderen Ausdruck mit gleichem Wert ersetzt werden (Substitutionsprinzip).

Seiteneffektfreiheit

Gleichheitsprinzip, Substitutionsprinzip

Ein Ausdruck kann immer durch einen anderen Ausdruck mit gleichem Wert ersetzt werden.

-> equational reasoning

Beispiel: Seiteneffekte

```
program example;
var flag : boolean;

function  f (n : integer) : integer;
begin
  if  flag  then f:= n  else f:= n+1;
  flag := not flag;
end;

begin
  flag := true;
  if f(2) = f(2)  then writeln(“ok“)
                  else writeln(“nicht ok“);
end.
```

Imperatives vs funktionales Programm

Pascal

```
var a : array [1..n] of integer;
procedure quicksort (l,r: integer);
var x,i,j,tmp : integer;
begin
 if r>l then
   begin
     x:=a[l]; i:=l; j:=r+1;
     repeat
       repeat i:=i+1 until a[i]>=x;
       repeat j:=j-1 until a[j]<=x;
       tmp:=a[j];a[j]:=a[i];a[i]:=tmp;
     until j<=i;
     a[i]:=a[j]; a[j]:=a[l]; a[l]:=tmp;
     quicksort(l,j-1); quicksort(j+1,r);
   end
end
```

Haskell

```
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs)
   = quicksort (filter (< x) xs)
       ++ [x] ++
     quicksort (filter (>=x) xs)
```