Konzepte von Programmiersprachen

1. Einführung

Höhere Programmiersprachen

- Ziele beim Entwurf:
  Ausdrucksmächtigkeit, Schlichtheit, Eleganz
- Weitere Anforderungen:
  - Universalität
  - Implementierbarkeit
  - Effizienz -> ... bei Ausführung & ... bei Programmierung

Konzepte von Programmiersprachen

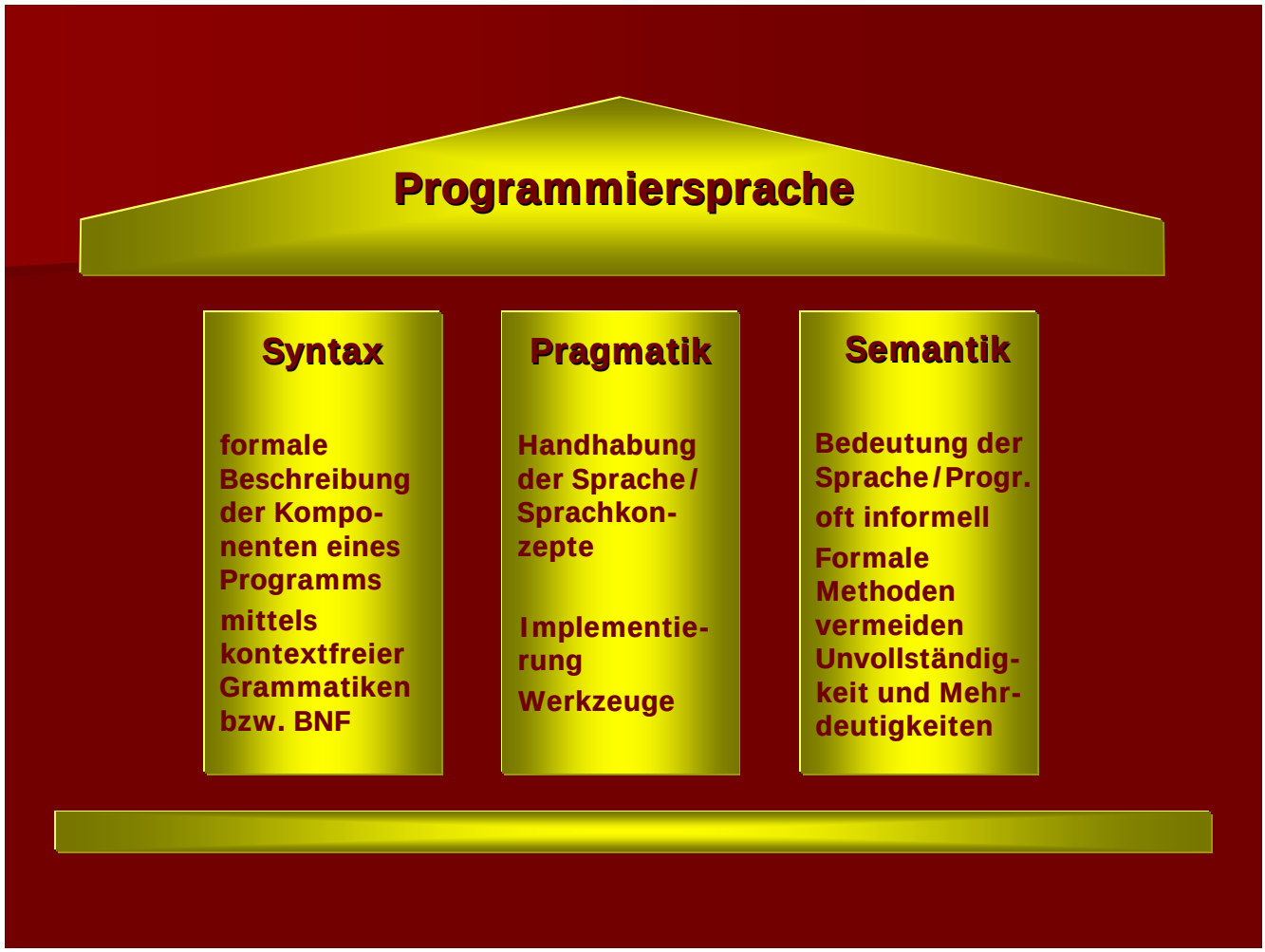
1. Einführung

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



Syntaxbeschreibung

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

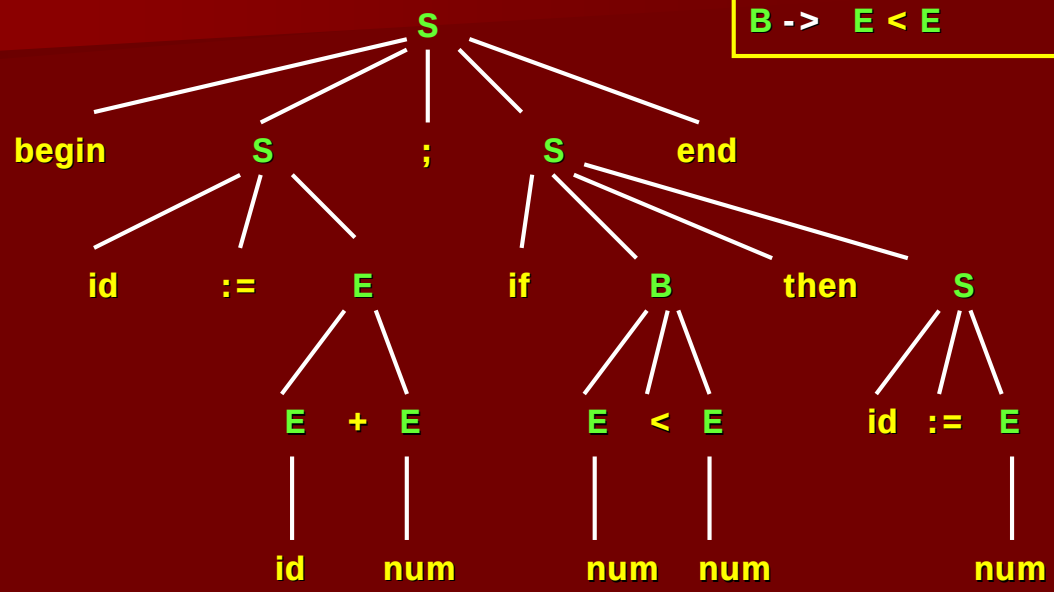
```
% Anweisungen (Statements)
S  ->  id := E                  % Wertzuweisung
   |   begin  S ; S end         % Hintereinanderausführung
   |   if  B then  S            % bedingte Anweisung

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

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

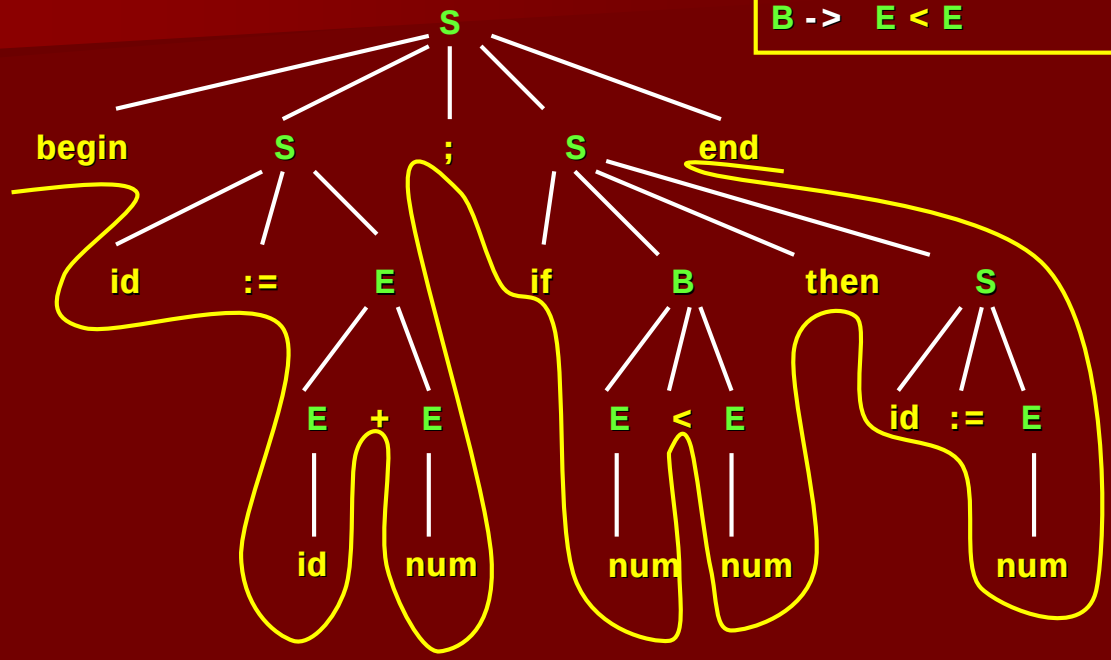
Syntaxbaum

```
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
```

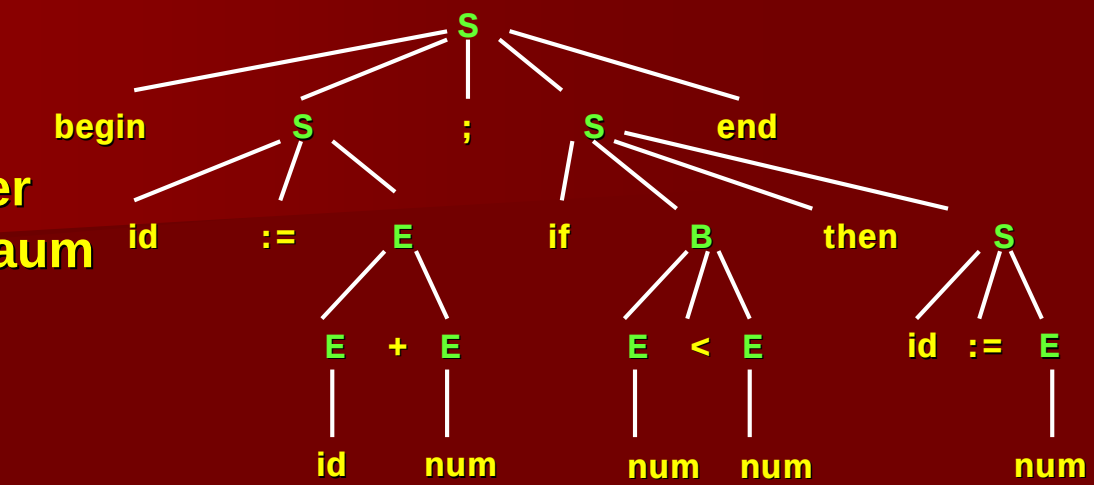


begin id := id + num ; if num < num then id := num end

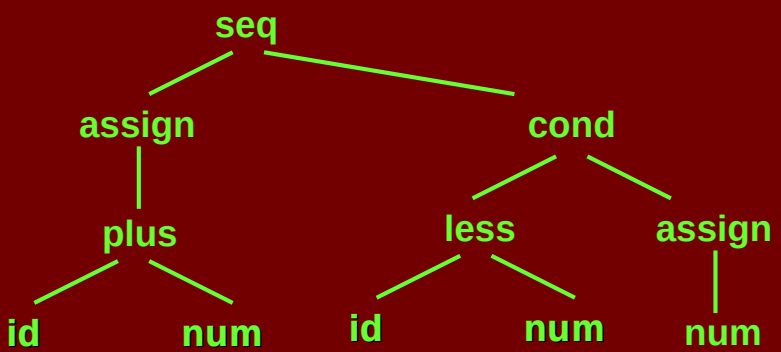
Abstrakte Syntax

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
```

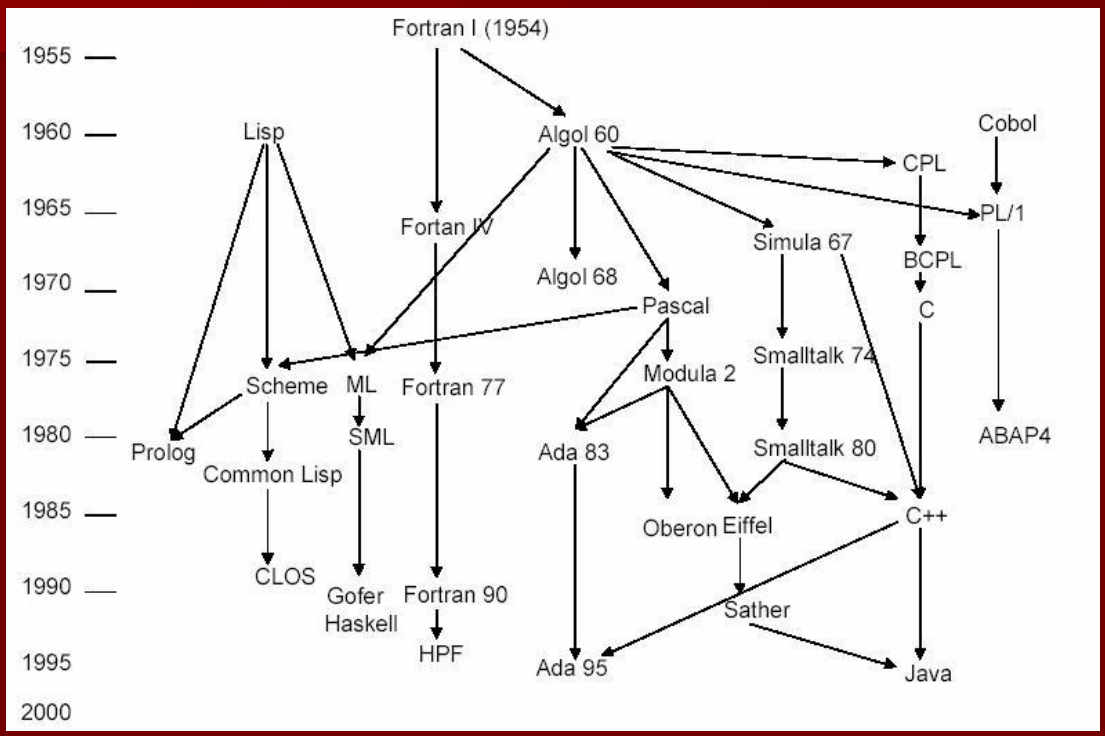
- **Pascal**

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

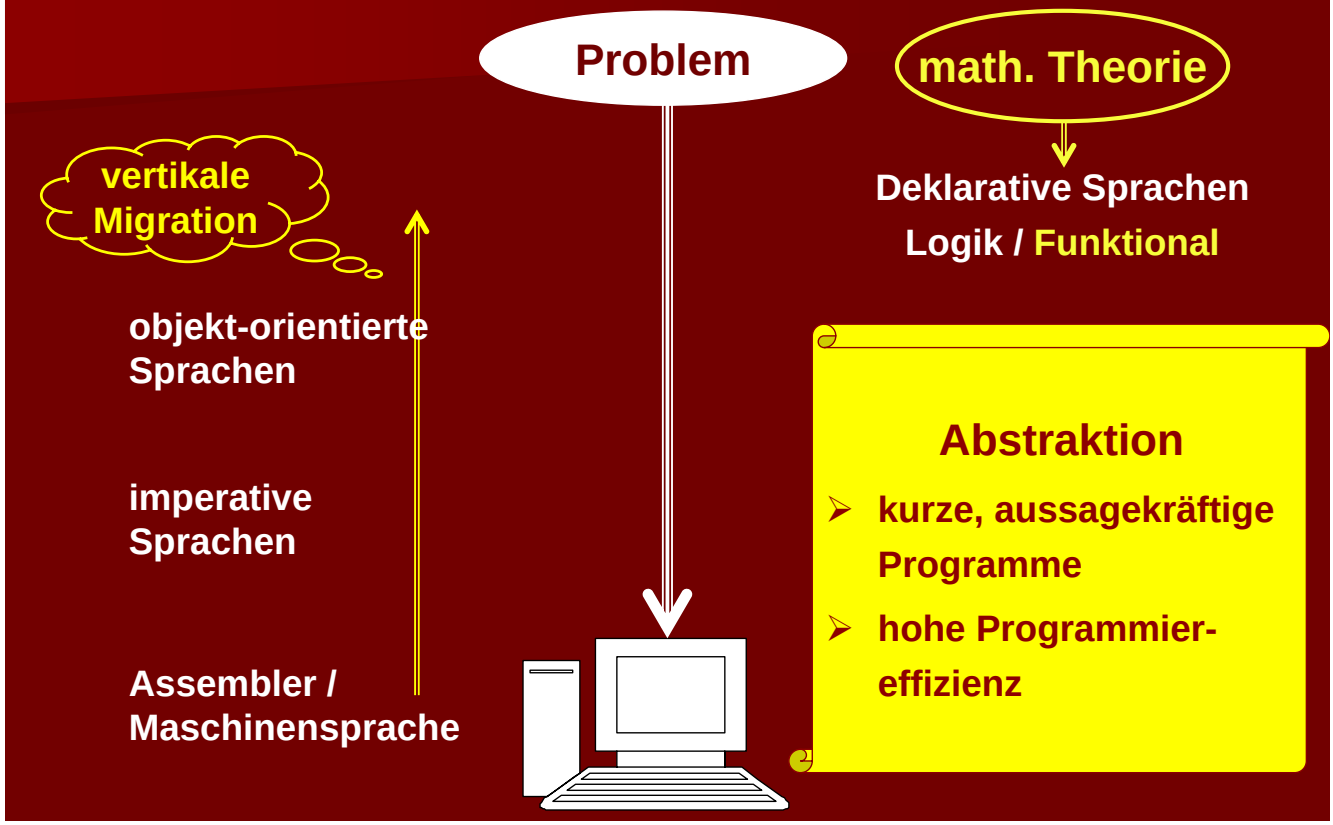
- **C**

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

Historie



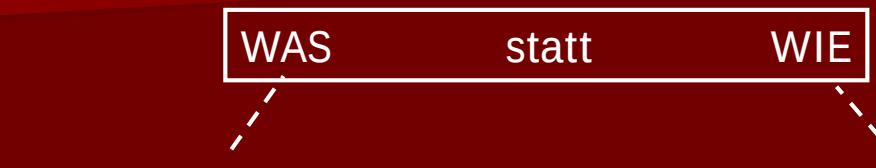
Klassifikation von Programmiersprachen



Programmierparadigmen

- **imperative Programmierspachen**
  - lat. imperare = befehlen
  - Programme = Abfolge von Befehlen
  - Abstraktion von Maschinensprachen
  - prozedurale Sprachen wie FORTRAN, PASCAL, MODULA, ADA, C, OCCAM etc.
  - Objekt-orientierte 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)
```