von Scannern und Parsern

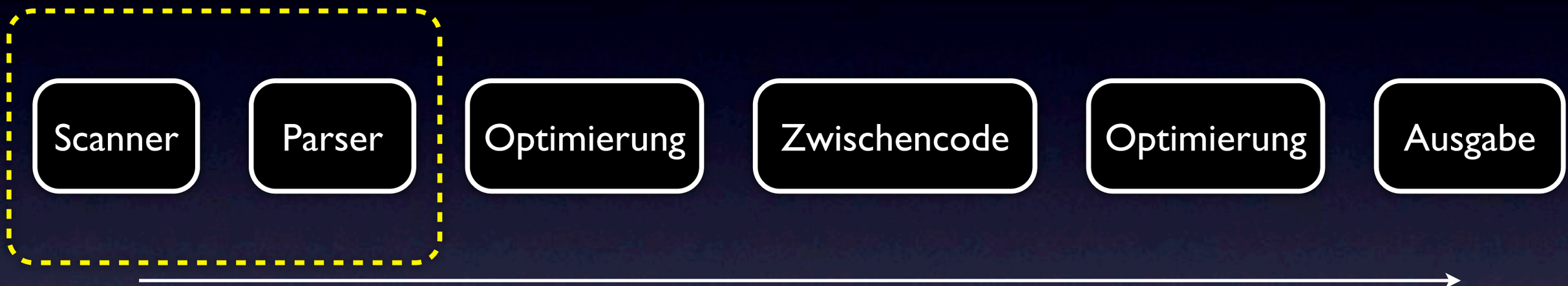
C4 - OpenChaos - 28. Januar 2010

Ben Fuhrmannek

Über mich

- Informatiker
- C4
- Freund des Parsergenerierens

Überblick Compilerbau



[Beispiel: Vereinfachter Aufbau eines modernen Compilers]

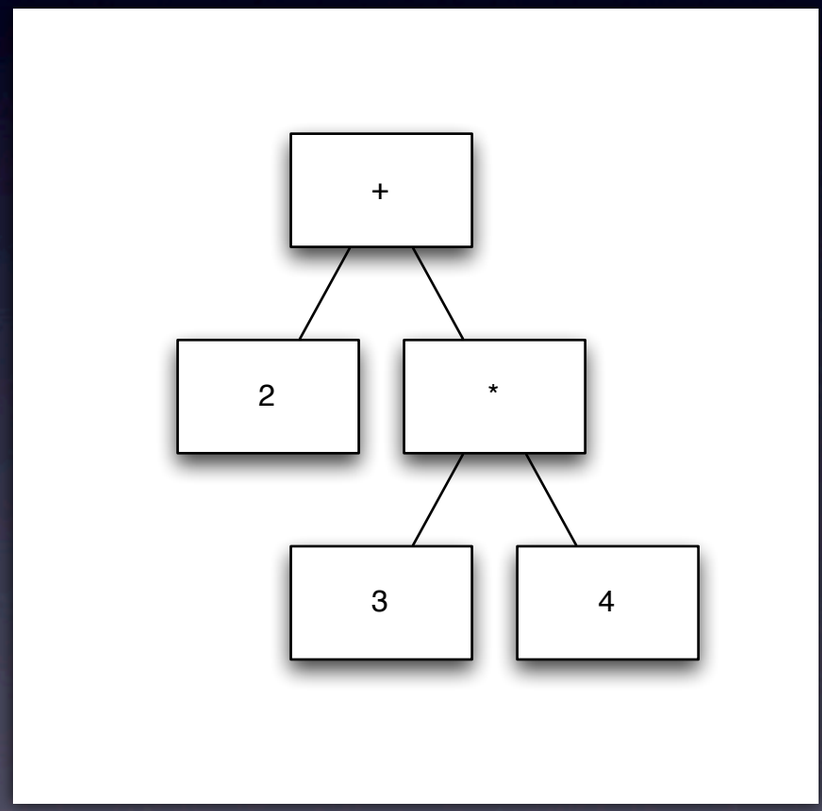
Vorher / Nachher

(triviale Arithmetik)

Eingabe:

“2 + 3 * 4”

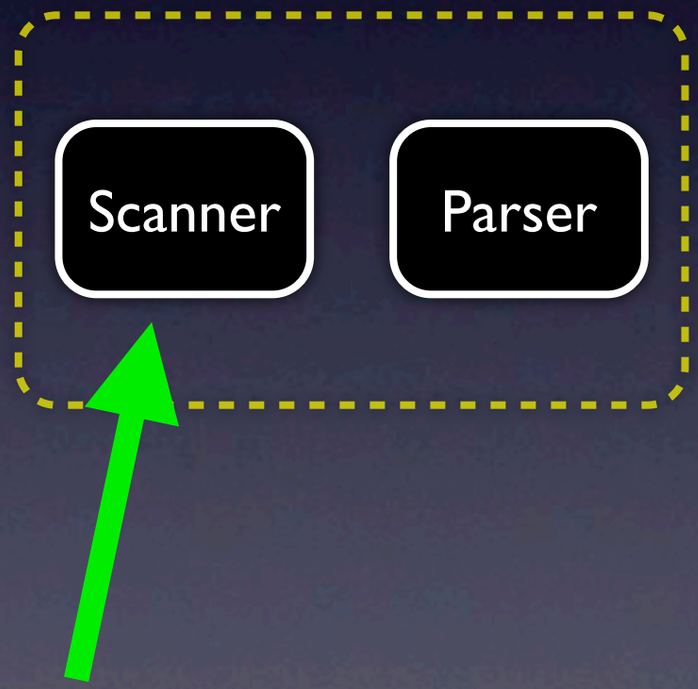
Ausgabe:



Wozu Parsen?

- Eingabe validieren.
  falsch wäre z.B. “23 + A” (Syntaxfehler)
  oder “2 3” (Logikfehler)
- Eingabe verarbeiten.
  z.B. Ausdruck ausrechnen

Scanner



Scanner

Google

scanner

Search Advanced Search

SafeSearch: Moderate ▼

Web › **Images** Show options…

Results **1** - **20** of about **10,700,000** (**0.22** seconds)

Related searches: **digital camera** **printer** **keyboard** **graphics tablet**

scanner
747 x 501 - 31k - jpg
Find similar images

Scanner
401 x 374 - 12k - jpg
Find similar images

acquire **Scanner**
438 x 308 - 19k - jpg

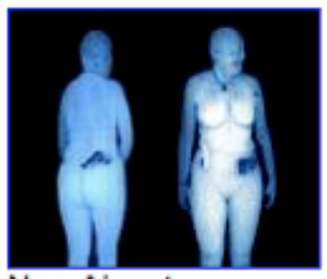
scanner |
400 x 352 - 28k - jpg
Find similar images

Also, better
278 x 248 - 16k - jpg
Find similar images

The **scanner**
375 x 375 - 15k - jpg
Find similar images

1.3.3 **Scanner**
300 x 300 - 14k - jpg
Find similar images

Canon CanoScan LiDE
400 x 400 - 61k - jpg
Find similar images



New Airport
400 x 331 - 35k - jpg
Find similar images

Slim Flatbed USB
400 x 308 - 29k - jpg
Find similar images

lexikalischer Scanner (Lexer)

Eingabe ==> logische Einheiten (Tokens)

Beispiel:

“2 + 3 * 4”

==> Zahl:2 plus Zahl:3 mal Zahl:4

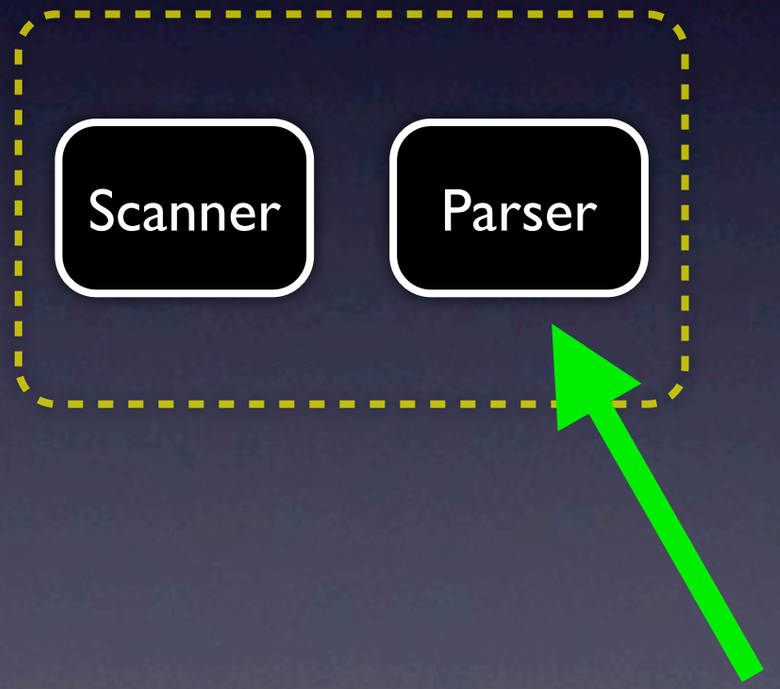
Scanner gefrickelt

```
#!/usr/bin/env python

import re
input = "2 + 3 * 4"

tokens = []
for t in re.findall(r'(\d+|\+|\*)', input):
    if re.match(r'\d+', t):
        tokens.append(('zahl', int(t)))
    elif t == '+':
        tokens.append('plus')
    elif t == '*':
        tokens.append('mal')

## ausgabe
for t in tokens:
    print t
```

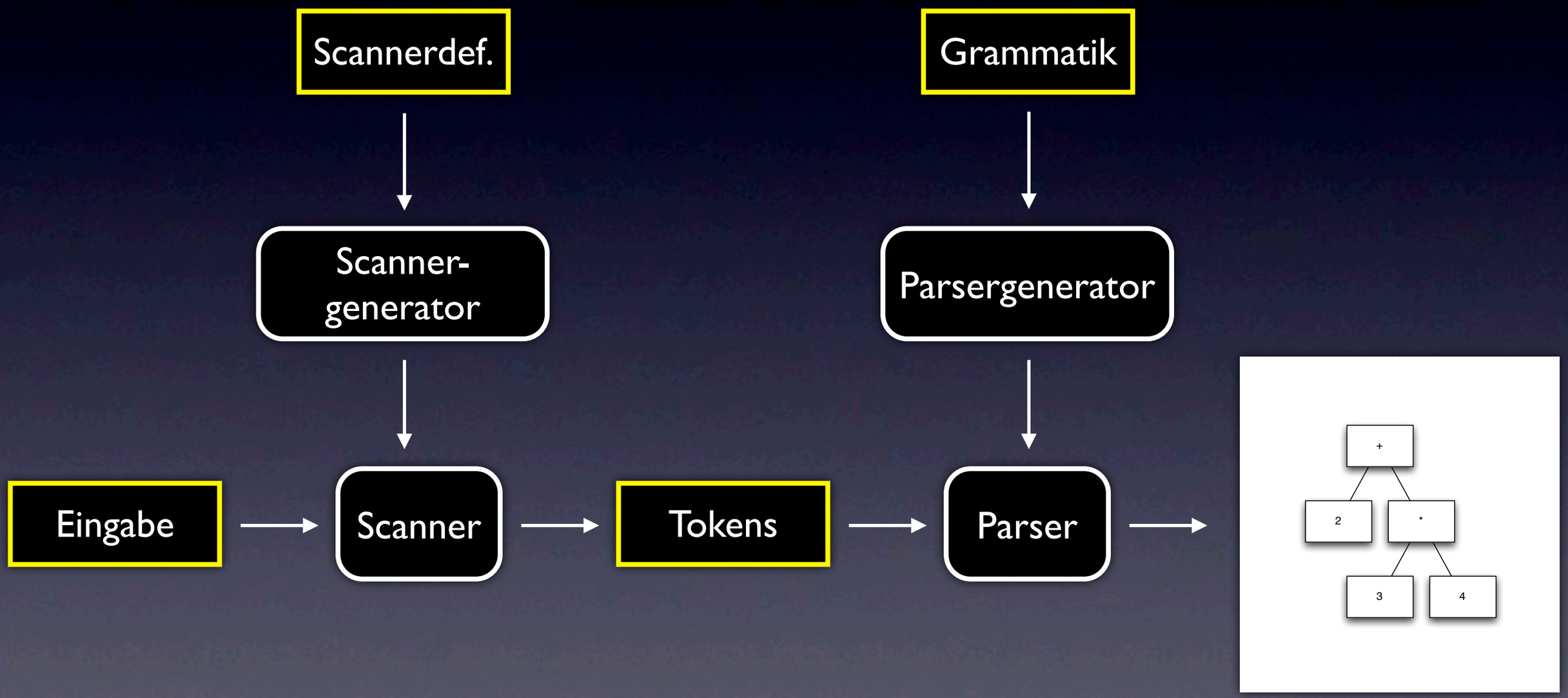
Scannerdefinition für leex

```
Definitions.

WS  = [\000-\s]
D = [0-9]

Rules.

{D}+ :     {token, {zahl, TokenLine, list_to_integer(TokenChars)}}.

\+ :       {token, {plus, TokenLine}}.
\* :       {token, {mal, TokenLine}}.

{WS}+ :    skip_token. %% whitespace
```

Parser



Big Picture


Scannerdef.
Grammatik
Scanner-
generator
Parsergenerator
Eingabe
Scanner
Tokens
Parser
+
2
*
3
4

Formale Grammatik (Definition)

(N, T, P, S)-Tupel

- N: Nichtterminale
- T: Terminale / Alphabet (äquivalent zu Tokens)
- P: Produktionen (Regeln)
- S: Startsymbol (aus der Menge N)

Grammatik (Produktion)

Eingabe: “2 + 3 * 4”

Regel 1: Exp -> Exp plus Exp
Regel 2: Exp -> Exp mal Exp
Regel 3: Exp -> zahl

Grammatik (N, T, P, S)

- N: Exp, Operation
- T: zahl, plus, mal
- P: (siehe vorher)
- S: Exp

Parsen

Regel 1: Exp -> Exp plus Exp
Regel 2: Exp -> Exp mal Exp
Regel 3: Exp -> zahl

Eingabe: “2 + 3 * 4”
Eingabe als Tokens: zahl plus zahl mal zahl

Regel 1: Exp -> Exp plus Exp
Regel 2: Exp -> Exp mal Exp plus Exp
Regel 3: Exp -> Exp mal Exp plus zahl
Regel 3: Exp -> Exp mal zahl plus zahl
Regel 3: Exp -> zahl mal zahl plus zahl

Parserdefinition in yecc

```
Nonterminals Exp.

Terminals zahl plus mal.

Left 100 plus.
Left 200 mal.

Rootsymbol Exp.

Exp -> Exp plus Exp.
Exp -> Exp mal Exp.
Exp -> zahl.
```

Beispiele und Erweiterungen

- ... siehe Beispiele mit yecc (erlang) und ply (python)

Links

- http://en.wikipedia.org/wiki/Comparison_of_parser_generators
- http://en.wikipedia.org/wiki/Context-free_grammar
- http://en.wikipedia.org/wiki/LALR_parser
- http://en.wikipedia.org/wiki/Context-free_grammar#Derivations_and_syntax_trees
- http://en.wikipedia.org/wiki/Backus–Naur_Form
- Python Lex/Yacc http://www.dabeaz.com/ply/
- Lemon Parser Generator http://www.hwaci.com/sw/lemon/