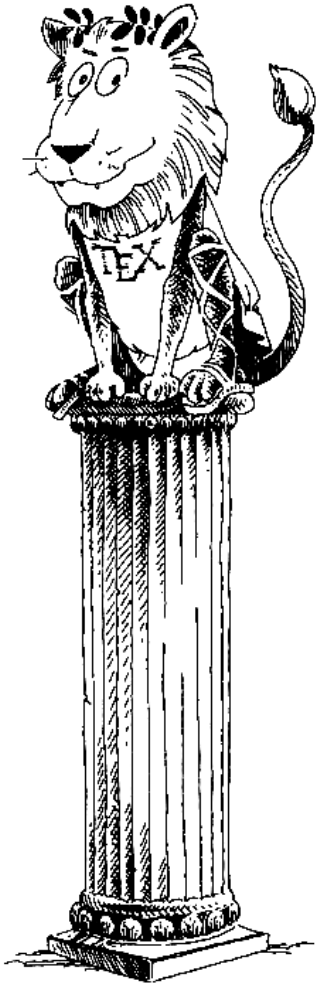
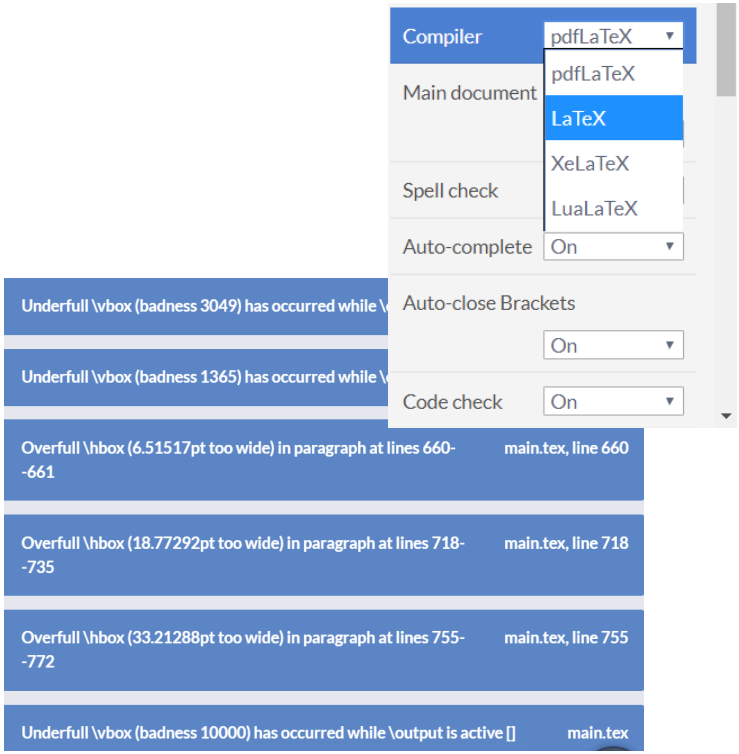


Illustrations by
DUANE BIBBY

Ein TeXnischer Überblick

Wo bin ich hier gelandet?



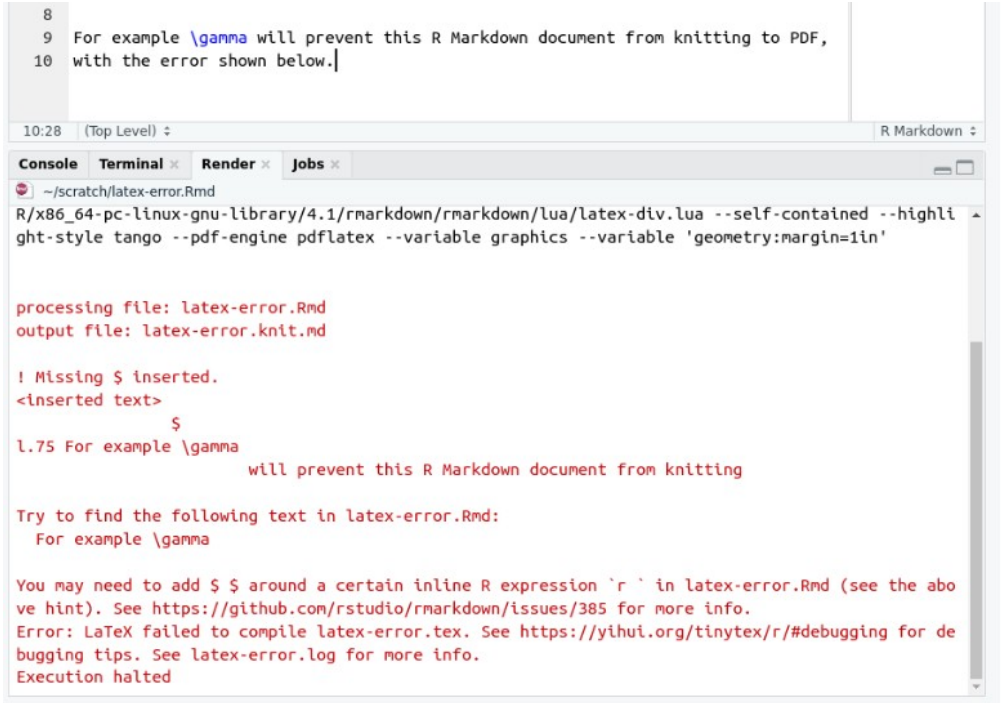
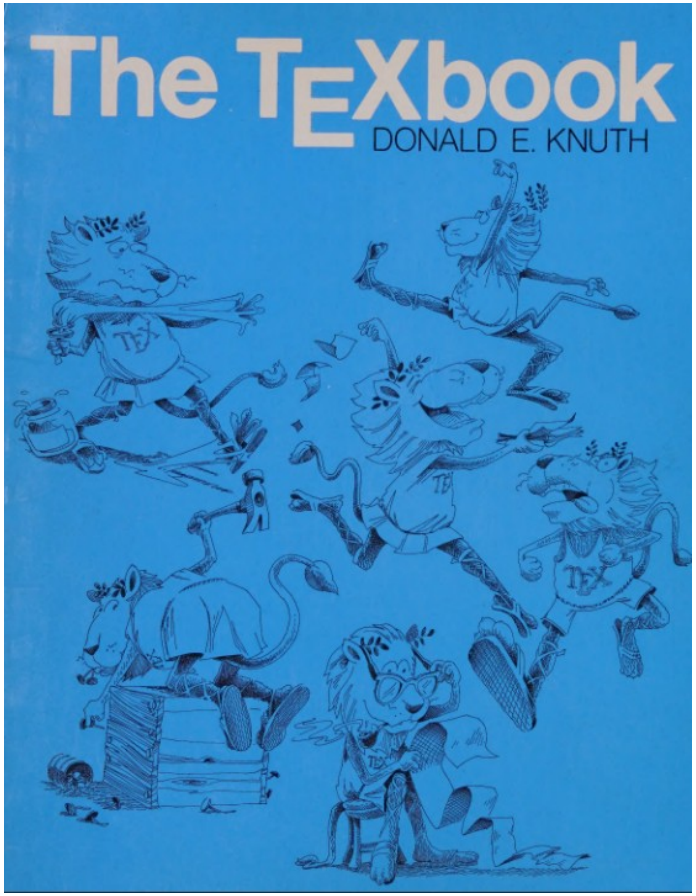


Fig. 2: LaTeX errors are often cryptic to new learners.

Wo bin ich hier gelandet?



✨ book report :3 ✨

Computers & Typesetting, Volume A:
The TeXbook

Donald E. Knuth

Addison-Wesely, 1984

Erwartungshaltungsmanagement

- Das ist KEIN Tutorial wie ihr eure Übungsblätter schreibt (Ihr werdet auch nicht sonderlich besser im TeXen)
- Schwerpunkt auf dem “inner working” von TeX (nicht LaTeX)
- kein Mathe
- “Modernes” LaTeX-Programmieren von Packages (expl3) wird hier nicht bearbeitet
- LaTeX Errors sind auch nach diesem Talk “cryptic” :(

Fahrplan

1. **Eine Familienchronik von TeX, LaTeX, und Verwandtschaft**
   (“Was unterscheidet TeX von LaTeX? Wie ist TeX programmiert?”)
2. **TeX als Makrosprache**
   (“Wie funktioniert die ‘Programmiersprache’ TeX?”)
3. **Arbeitsweise der TeX-Schriftsatzengine**
   (“Wie werden aus Buchstaben die Zeilen und Seiten?”)

Fahrplan

Fahrplan

1. Familienchronik

Overview of the TEX Historic Archive

Dr. Ulrik Vieth

EuroBachoTEX 2007

- History of TEX goes back to 30 years since 1977
- History of TEX well documented in the literature (but only early beginnings of TEX itself)
- History of everything else (macros, fonts, tools) very incomplete. Only anecdotal evidence left.
- Very few, if any, original files have survived
- Most details have been lost or are forgotten

Der Gründungsmythos

[Knuth: Kyoto Price Lecture 1997]

... in the olden days

Der Gründungsmythos [Knuth: Kyoto Price Lecture 1997]

... in the olden days

KI-generiert

Der Gründungsmythos

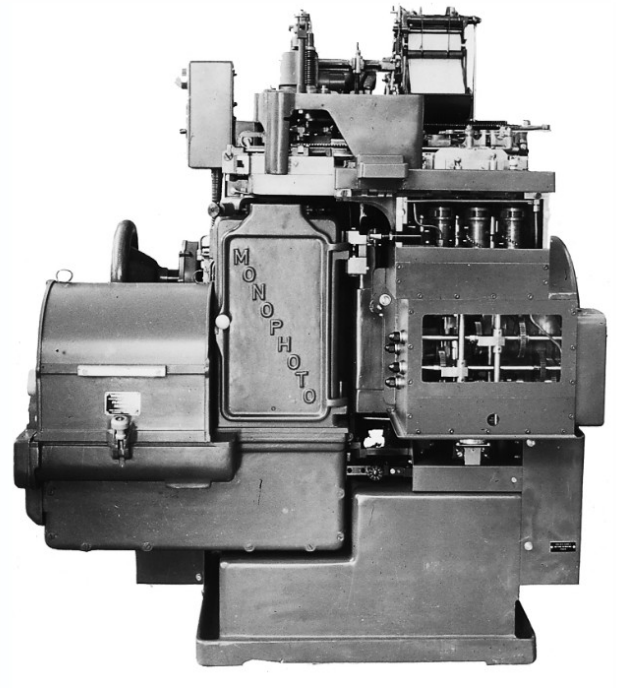
[Knuth: Kyoto Price Lecture 1997]

... doch dann erklärte uns PHOTOTYPESETTING den Krieg und alles änderte sich

Der Gründungsmythos

[Knuth: Kyoto Price Lecture 1997]

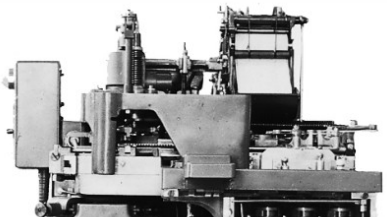
... doch dann erklärte uns PHOTOTYPESETTING den Krieg und alles änderte sich

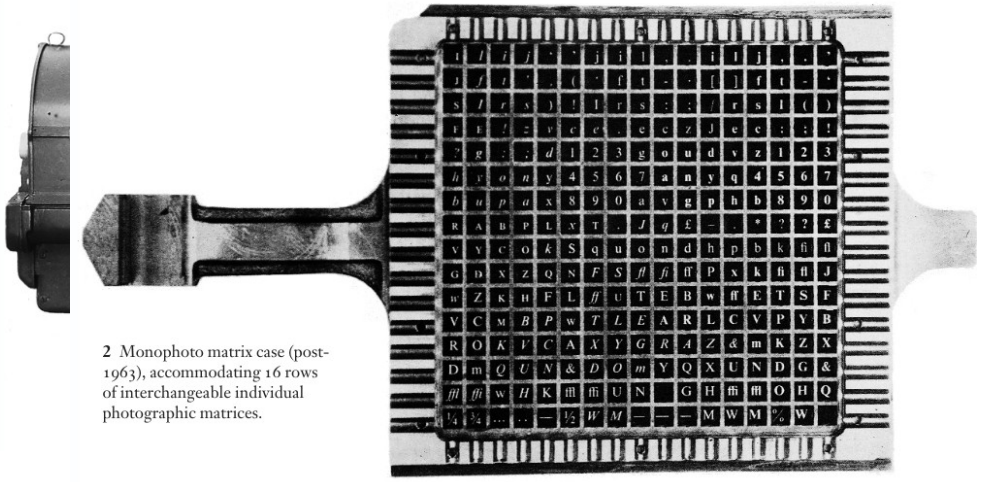


Der Gründungsmythos

[Knuth: Kyoto Price Lecture 1997]

… doch dann erklärte uns PHOTOTYPESETTING den Krieg und alles änderte sich



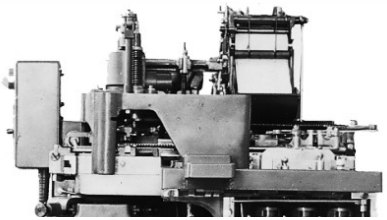


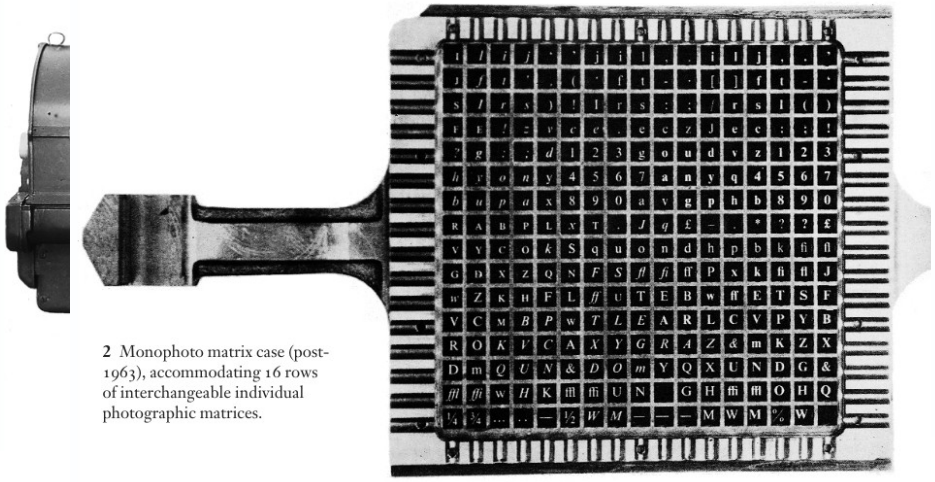
2 Monophoto matrix case (post-1963), accommodating 16 rows of interchangeable individual photographic matrices.

Der Gründungsmythos

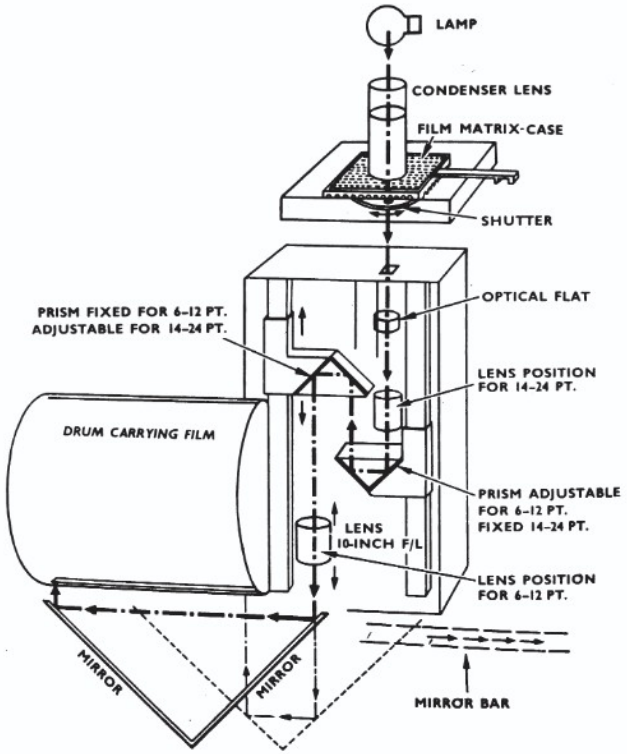
[Knuth: Kyoto Price Lecture 1997]

... doch dann erklärte uns PHOTOTYPESETTING den Krieg und alles änderte sich





2 Monophoto matrix case (post-1963), accommodating 16 rows of interchangeable individual photographic matrices.



Program A (*Addition, subtraction, and normalization*). The following program is a subroutine for Algorithm A, and it is also designed so that the normalization portion can be used by other subroutines which appear later in this section. In this program and in many other programs throughout this chapter, OFLO stands for a subroutine which prints out a message to the effect that MIX's overflow toggle was unexpectedly found to be "on." The byte size b is assumed to be a multiple of 4. The normalization routine NORM assumes that rI2 = e and rAX = f, where rA = 0 implies rX = 0 and rI2 < b.

```
01 EXP    EQU    1:1          Definition of exponent field.
02 FSUB   STA    TEMP         Floating-point subtraction subroutine
03        LDAN   TEMP         Change sign of operand.
```

Slide 10.

But the results were very disappointing. For example [SLIDE 10], here's some of the type from the second, "tuned up" version of their new fonts. These were much improved from the first attempt, but still unacceptable. The "N" in "NORM" was tipped; the "ff" in "effect" was much too dark; the letters "ip" in "multiple" were too close together; and so on.

Program A (*Addition, subtraction, and normalization*). The following program is a subroutine for Algorithm A, and it is also designed so that the normalization portion can be used by other subroutines which appear later in this section. In this program and in many other programs throughout this chapter, OFLO stands for a subroutine which prints out a message to the effect that MIX's overflow toggle was unexpectedly found to be "on." The byte size b is assumed to be a multiple of 4. The normalization routine NORM assumes that r12 = e and rAX = f, where rA = 0 implies rX = 0 and r12 < b.

```
01  EXP    EQU    1:1          Definition of exponent field.
02  FSUB   STA    TEMP         Floating-point subtraction subroutine
03         LDAN   TEMP         Change sign of operand.
```

Slide 10.

But the results were very disappointing. For example [SLIDE 10], here's some of the type from the second, "tuned up" version of their new fonts. These were much improved from the first attempt, but still unacceptable. The "N" in "NORM" was tipped; the "ff" in "effect" was much too dark; the letters "ip" in "multiple" were too close together; and so on.

(*Addition, subtraction, and normalization*). The fol
ne for Algorithm A, and it is also designed so that th
be used by other subroutines which appear later in
n and in many other programs throughout this
subroutine which prints out a message to the ef
gle was unexpectedly found to be "on." The byte si
tiple of 4. The normalization routine NORM assum
where rA = 0 implies rX = 0 and rI2 < *b*.

```
EQU     1:1          Definition of exponent field
STA     TEMP         Floating-point subtraction :
LDAN    TEMP         Change sign of operand.
```

photo 10

The following program
o that the normalization
later in this section. In
ut this chapter, OFLO
o the effect that MIX's
e byte size *b* is assumed
M assumes that rI2 = *e*
< *b*.

nent field.
ıtraction subroutine
erand.

g. For example [SLIDE 10],
"tuned up" version of their
the first attempt, but still
ed; the "ff" in "effect" was
le" were too close together;

and so on.

LaTeX Packages

TeX Format
≙ „Standard Library“

Engine
(Interpreter, Renderer)

LaTeX Packages

TeX Format
≙ „Standard Library“

Engine
(Interpreter, Renderer)

TeX
[1978]

LaTeX Packages

TeX Format
≙ „Standard Library“

plain TeX
[1978]

Engine
(Interpreter, Renderer)

TeX
[1978]

LaTeX Packages

TeX Format
≙ „Standard Library“

plain TeX
[1978]

LaTeX
[1985]

Engine
(Interpreter, Renderer)

TeX
[1978]

LaTeX Packages

TeX Format
≙ „Standard Library“

plain TeX
[1978]

LaTeX
[1985]

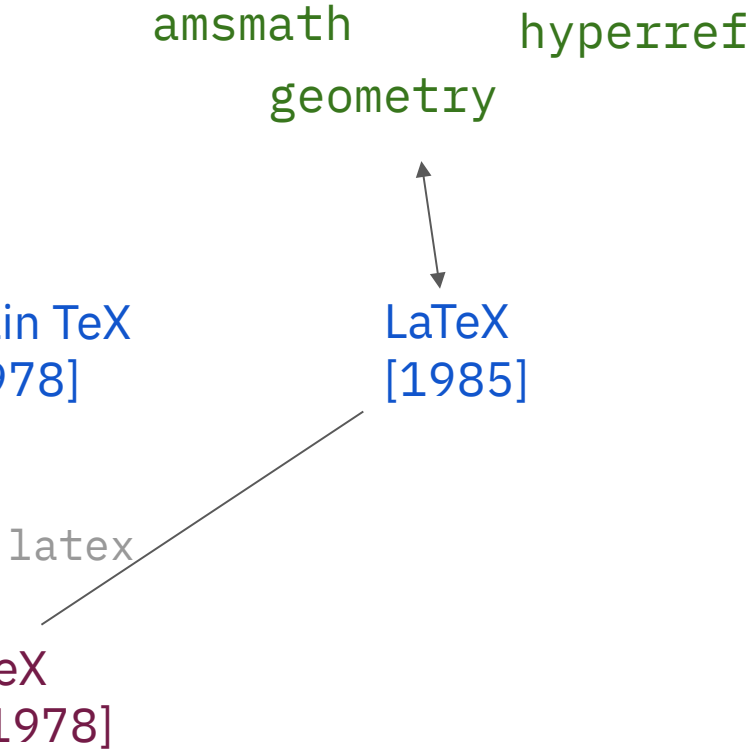
Engine
(Interpreter, Renderer)

TeX
[1978]

LaTeX Packages

TeX Format
≙ „Standard Library“

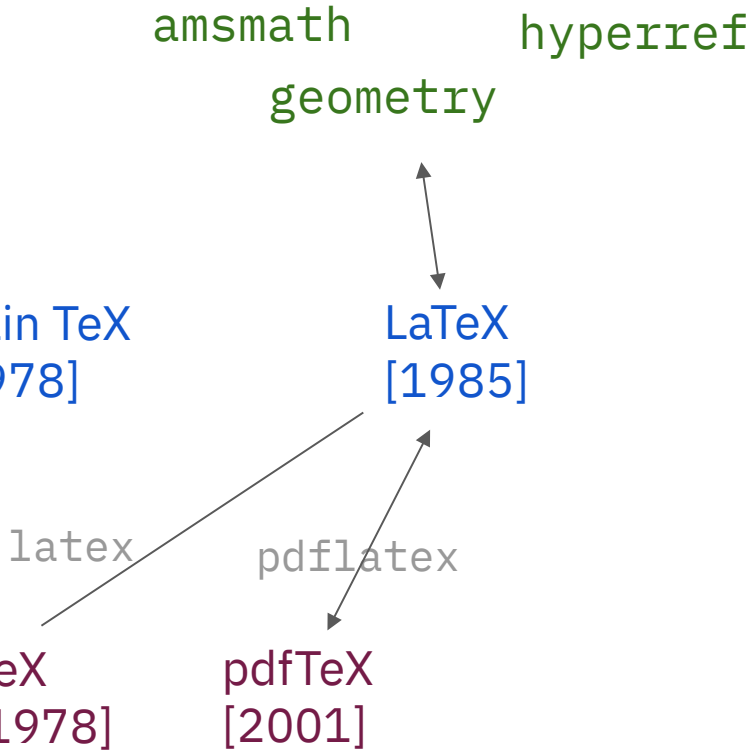
Engine
(Interpreter, Renderer)



LaTeX Packages

TeX Format
≙ „Standard Library“

Engine
(Interpreter, Renderer)

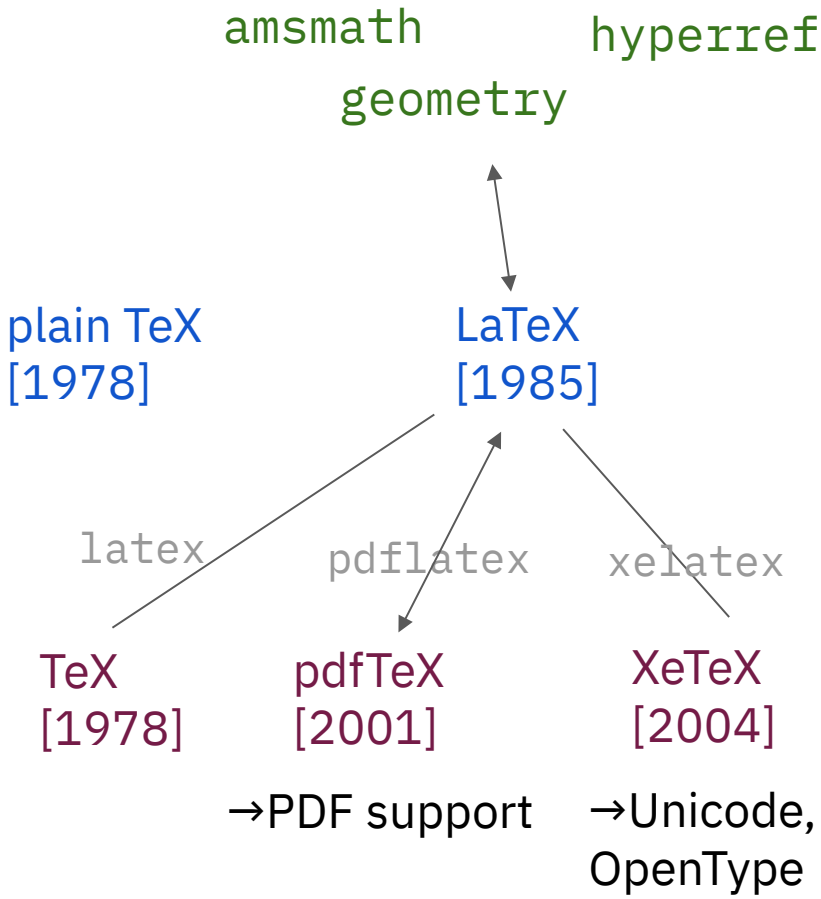


→PDF support

LaTeX Packages

TeX Format
≙ „Standard Library“

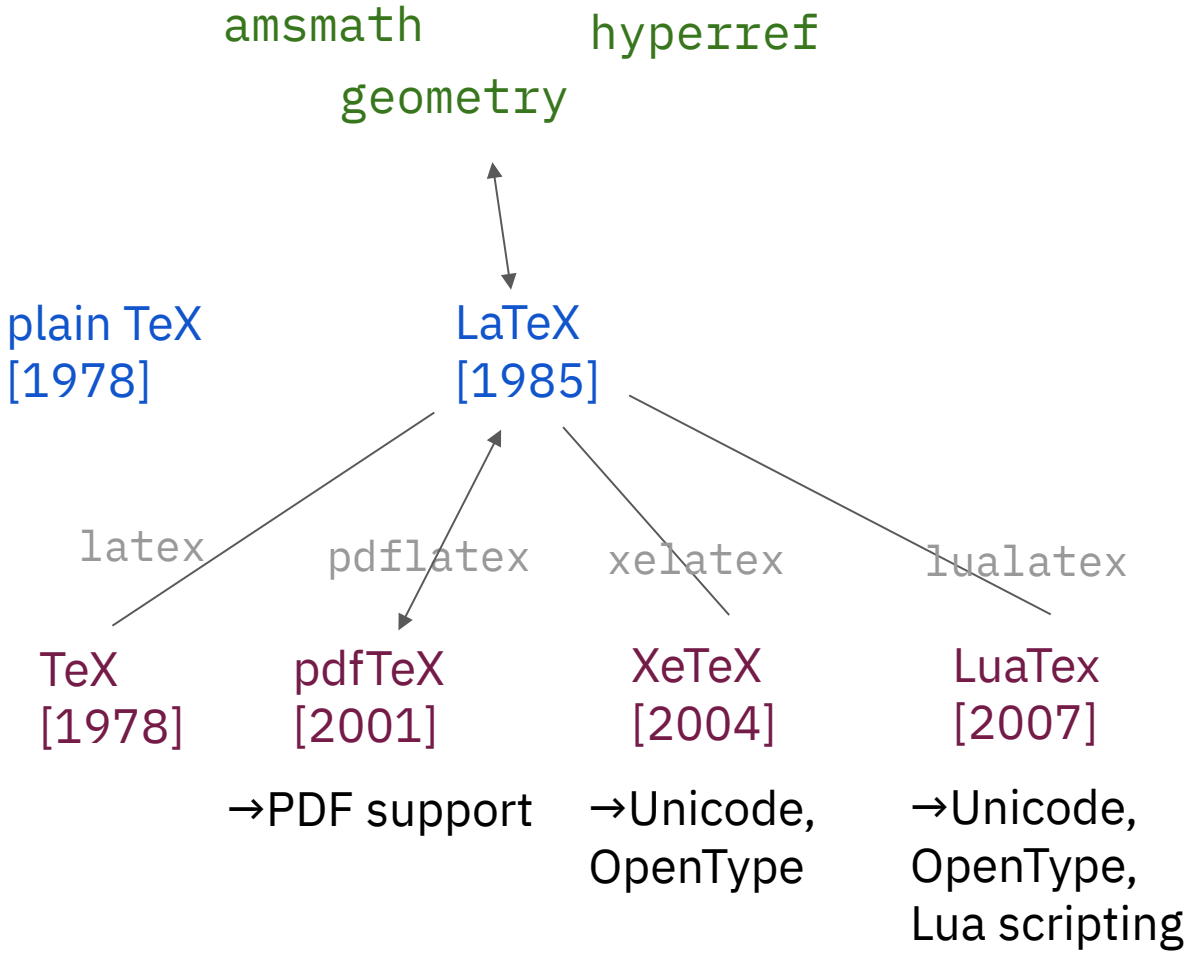
Engine
(Interpreter, Renderer)



LaTeX Packages

TeX Format
≙ „Standard Library“

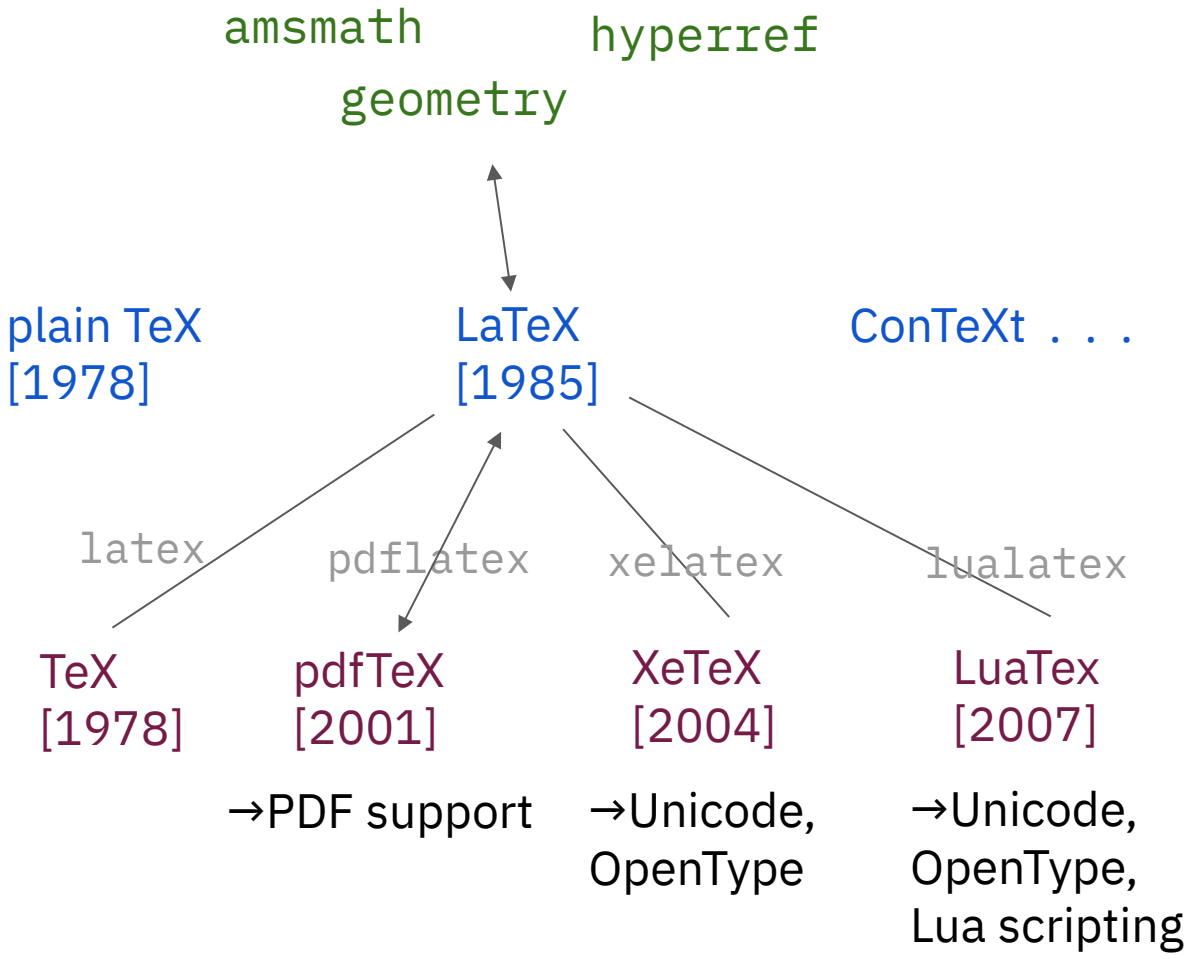
Engine
(Interpreter, Renderer)

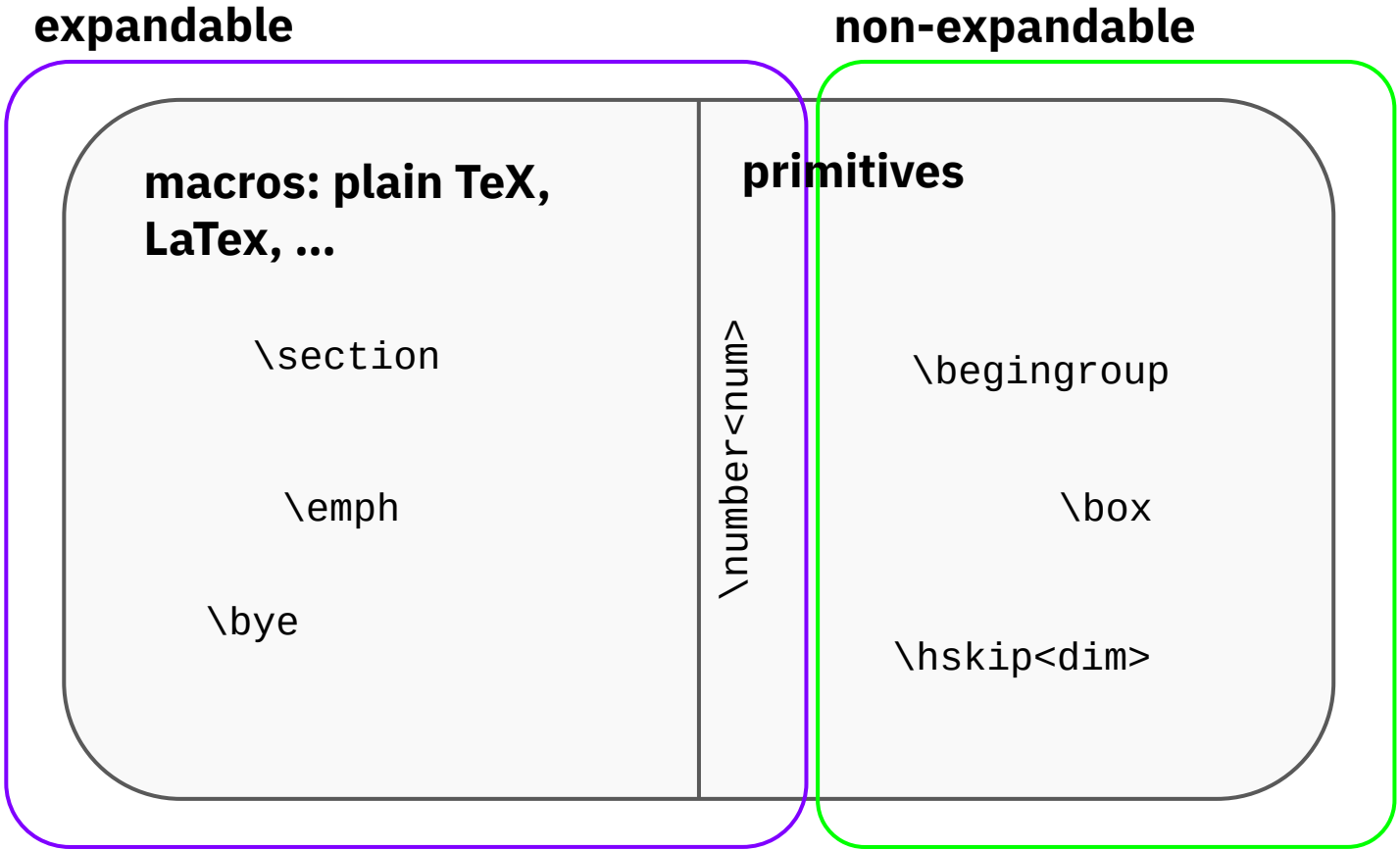


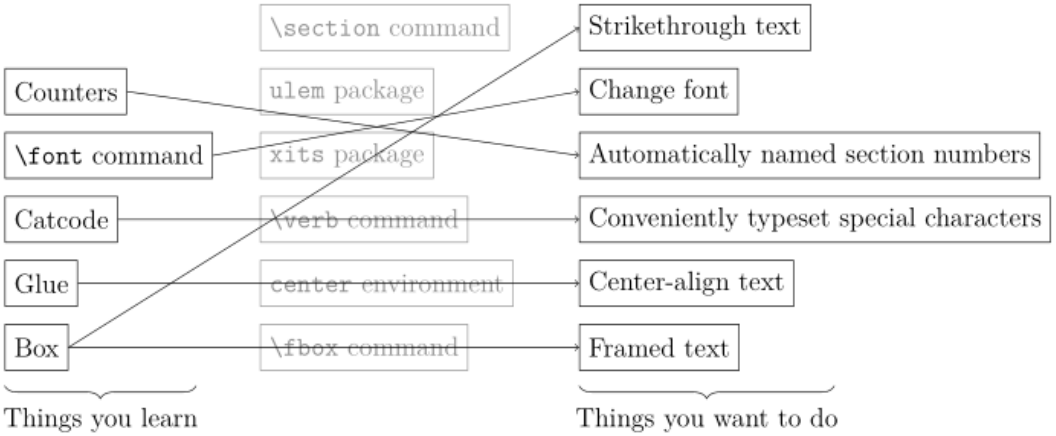
LaTeX Packages

TeX Format
≙ „Standard Library“

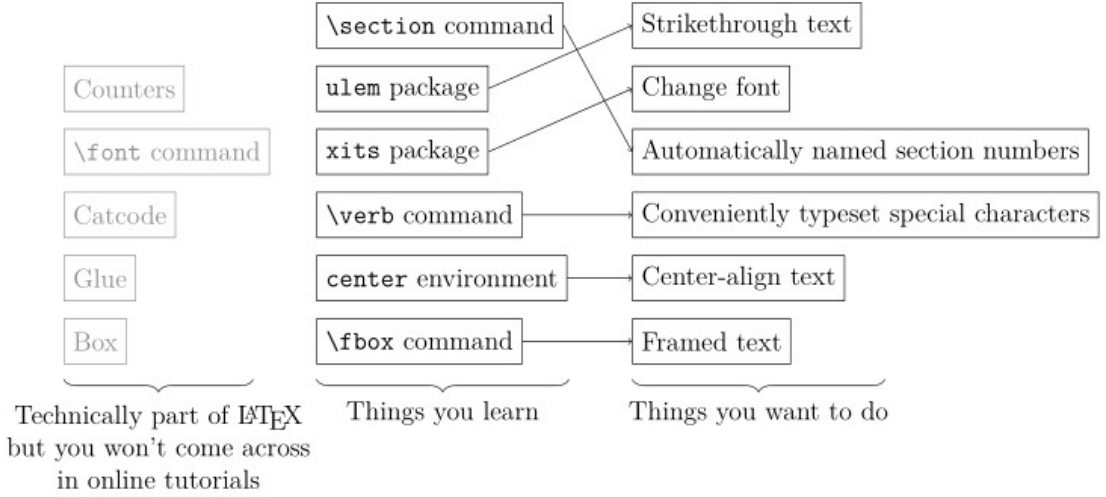
Engine
(Interpreter, Renderer)




expandable
non-expandable
macros: plain TeX, LaTex, ...
\section
\emph
\bye
\number<num>
primitives
\begingroup
\box
\hskip<dim>


\section command
ulem package
xits package
\verb command
center environment
\fbox command
Counters
\font command
Catcode
Glue
Box
Strikethrough text
Change font
Automatically named section numbers
Conveniently typeset special characters
Center-align text
Framed text
Things you learn
Things you want to do

TeX


\section command
ulem package
xits package
\verb command
center environment
\fbox command
Counters
\font command
Catcode
Glue
Box
Strikethrough text
Change font
Automatically named section numbers
Conveniently typeset special characters
Center-align text
Framed text
Technically part of LaTeX but you won't come across in online tutorials
Things you learn
Things you want to do

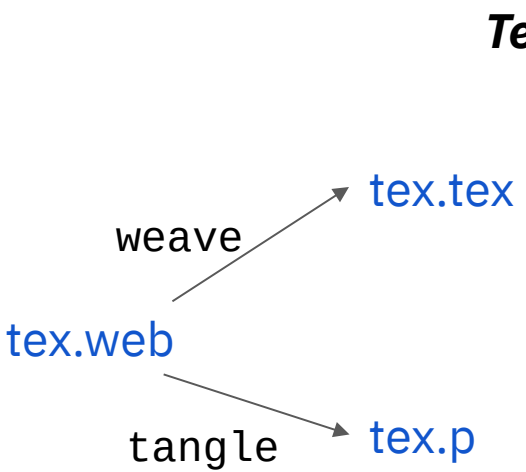
LaTeX

Trivia: In welcher Programmiersprache ist TeX programmiert?

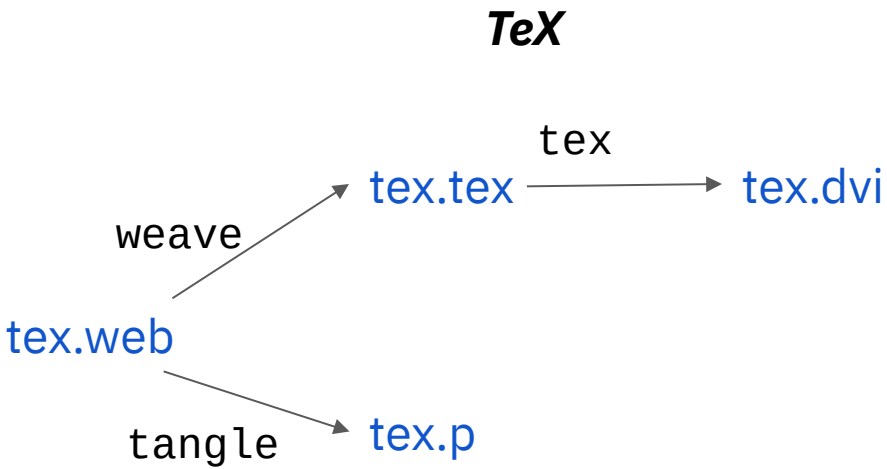
TeX

tex.web

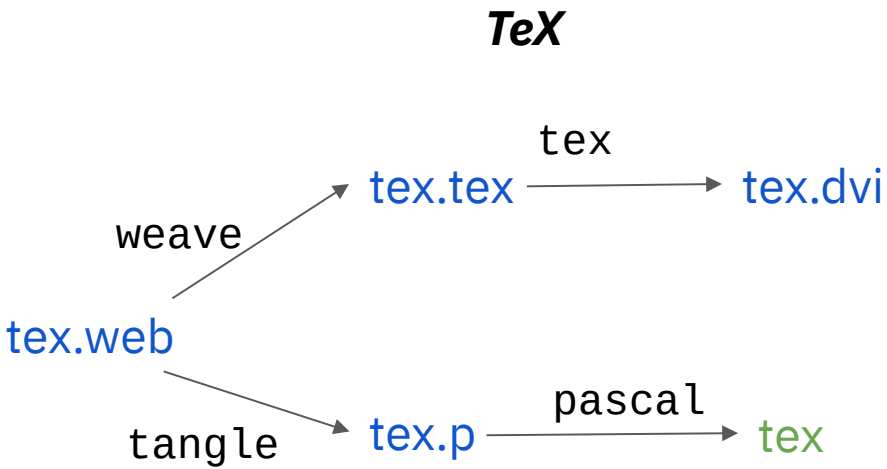
Trivia: In welcher Programmiersprache ist TeX programmiert?



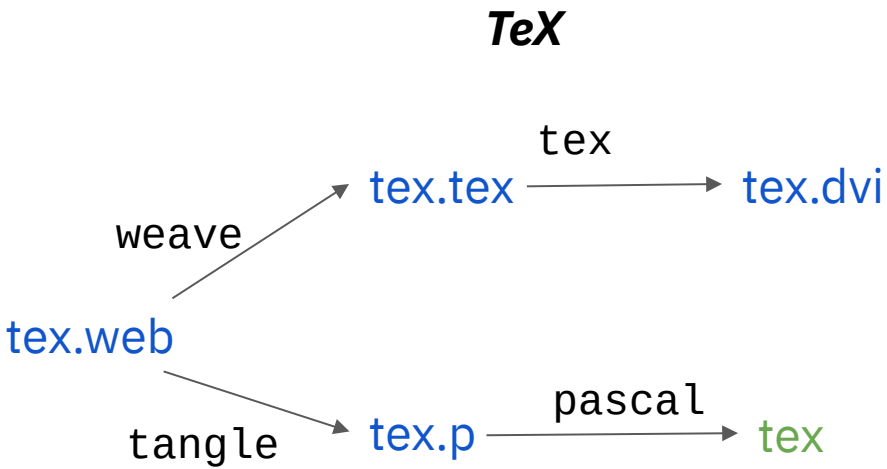
Trivia: In welcher Programmiersprache ist TeX programmiert?



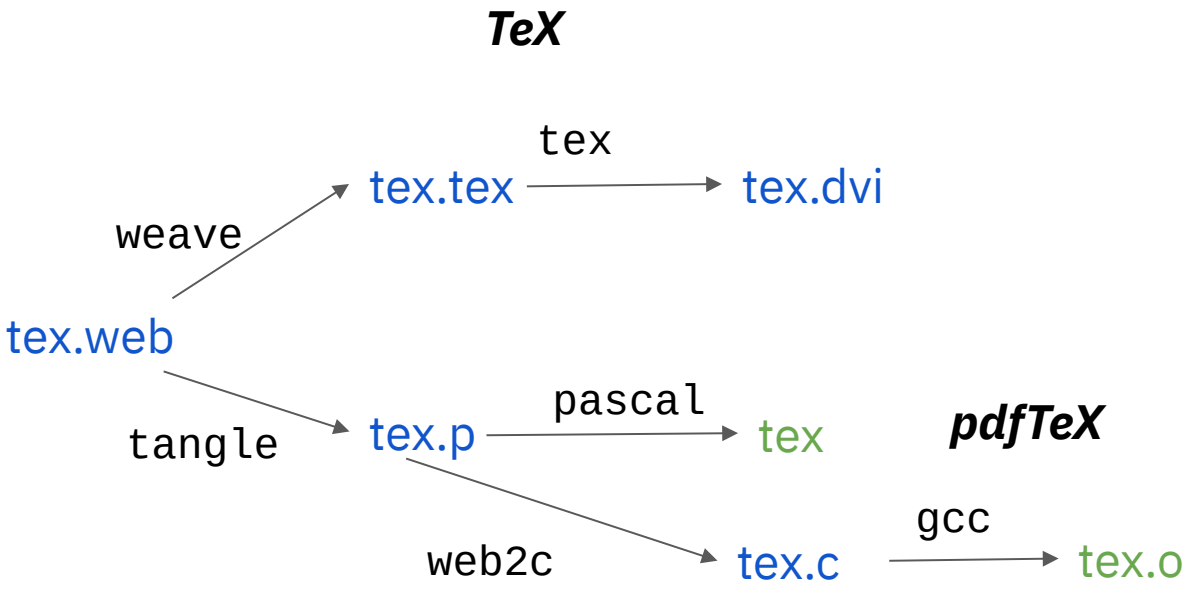
Trivia: In welcher Programmiersprache ist TeX programmiert?



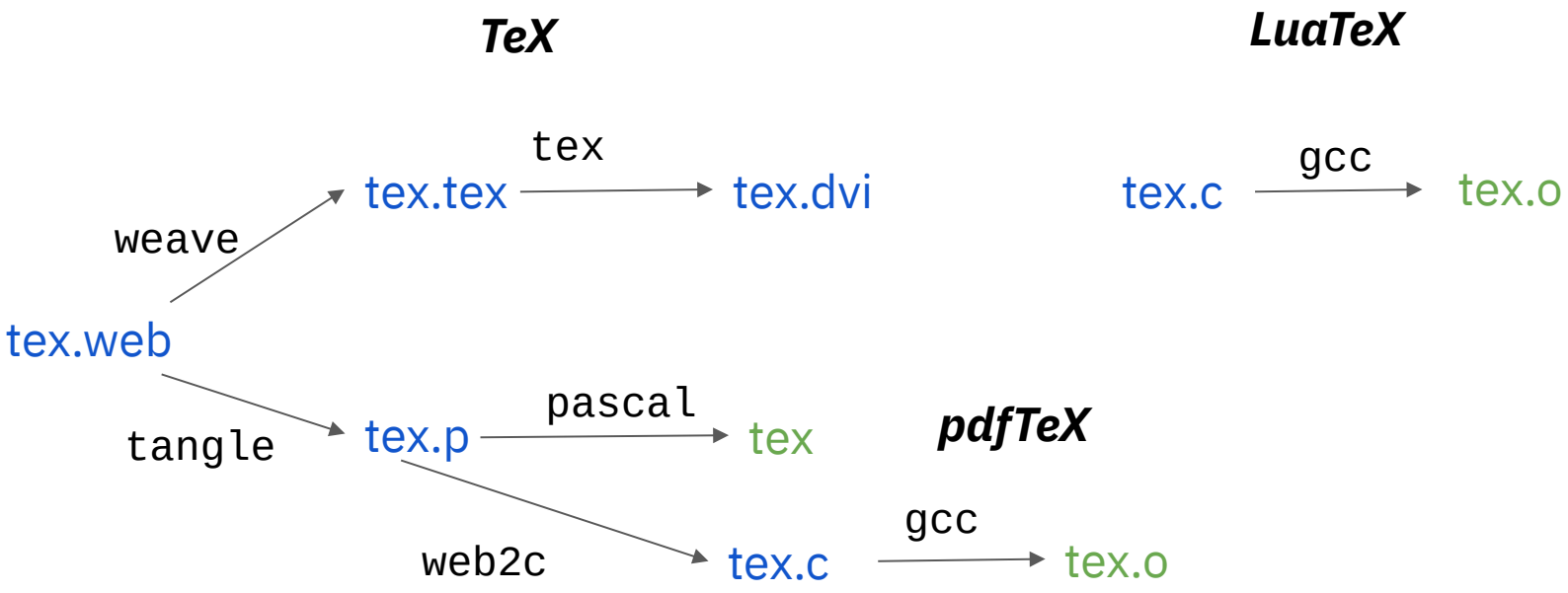
Trivia: In welcher Programmiersprache ist TeX programmiert?



Trivia: In welcher Programmiersprache ist TeX programmiert?



Trivia: In welcher Programmiersprache ist TeX programmiert?



Computers & Typesetting, Volume B: TeX: The Program

von Donald Knuth | 1. Januar 1986

5,0 ★★★★★ (10)

Gebundenes Buch

69^99 $

Lieferung **Do., 4. Dez.**

Wird versendet nach Deutschland

Nur noch 3 auf Lager (mehr ist unterwegs).

In den Einkaufswagen

Andere Angebote

17,05 $ (7+ gebrauchte und neue Artikel)

Kindle

65^99 $ Preis der Print-Ausgabe: ~~69,99$~~

Sofort lieferbar

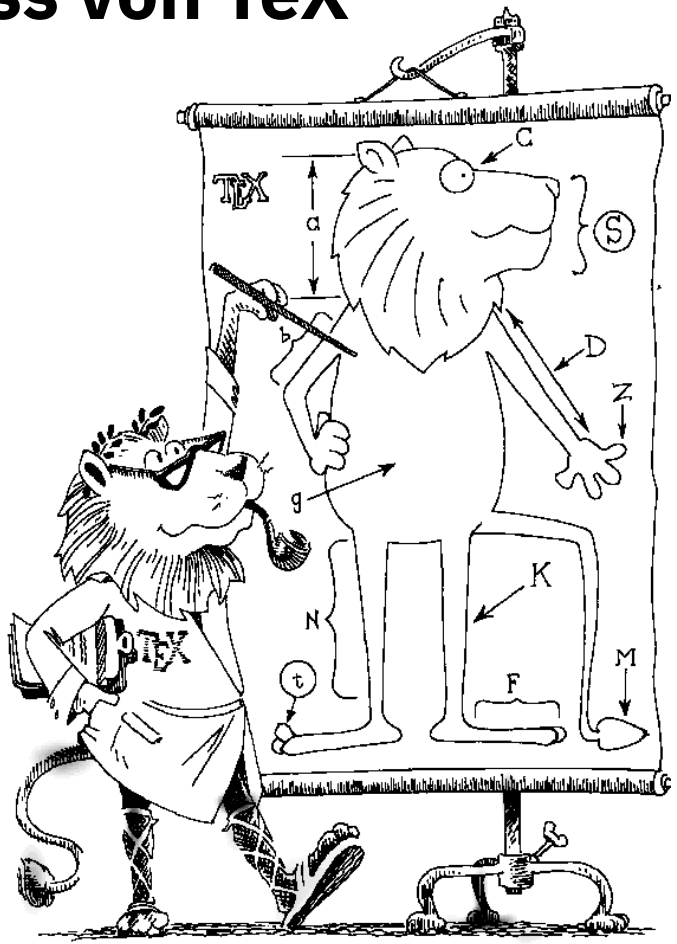
Intermission: Verdauungsprozess von TeX

We shall study TEX's digestive processes, i.e., what TEX does with the lists of tokens that arrive in its "stomach." Chapter 7 has described the process by which input files are converted to lists of tokens in TEX's "mouth," and Chapter 20 explained how expandable tokens are converted to unexpandable ones in TEX's "gullet" by a process similar to regurgitation. When unexpandable tokens finally reach TEX's gastro-intestinal tract, the real activity of typesetting begins, and that is what we are going to survey in these summary chapters.

Part 2

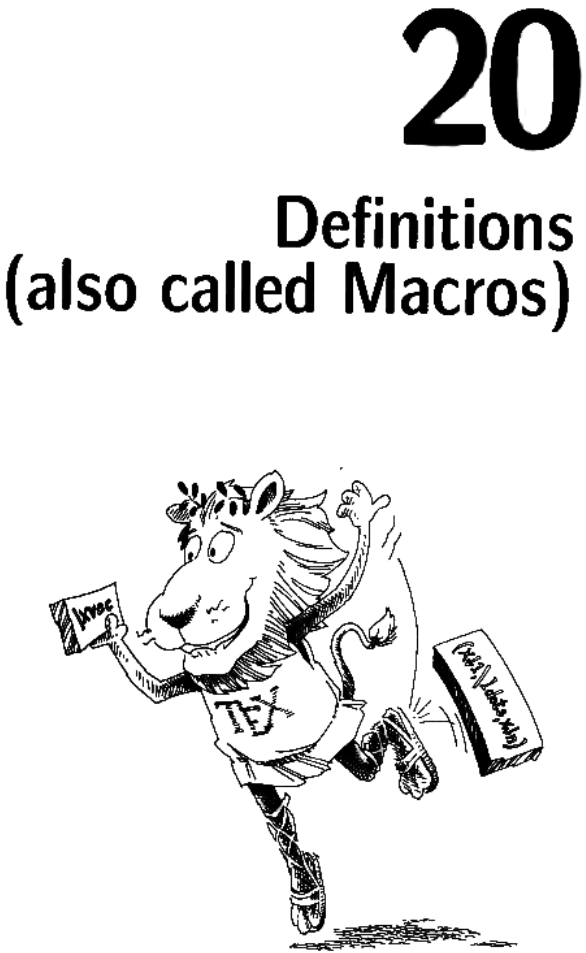
1. Tokenisierung des Sourcecode ("mouth")
2. Expansion der Tokens ("gullet")
3. Textsatz in die Form von Boxes ("stomach")
4. Finales Seitenlayout und Output ("...")

Part 3



2. TeX als Makrosprache

- Interpretierte Sprache
- Keine Funktionsaufrufe, sondern *Expansion* von Makros
- Kein Parsing/AST, sondern Stream Processing ohne positives Lookahead
- “Interpreter” expandiert so lange Tokens, bis diese nicht weiter expandierbar sind

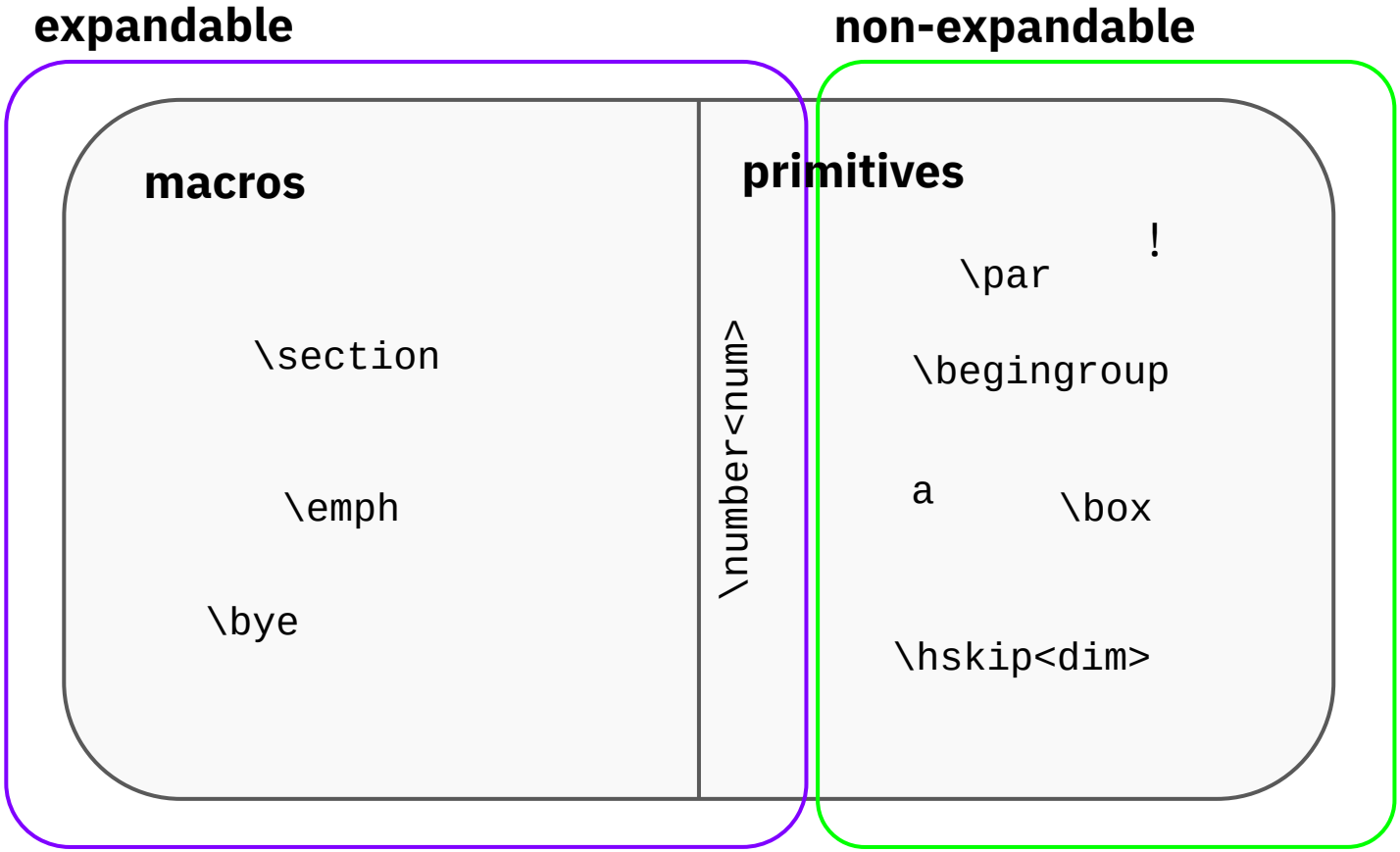


Tokenisierung (“mouth”)

- `\n␣*\n` → [par]
- `(␣ | \n)+` → ␣
- “`\`” . `([a-zA-Z]+)`$_{\text{macroname}}$ `␣*` → [macroname]
- “`\`” . `([^a-zA-Z])`$_{\text{macroname}}$ → [macroname]
- Zeichen *x* → *x*

```
\TeX␣nical␣{\hskip 3pt}␣$\frac{a}{b}$↩
␣␣↩
```

```
TeX n i c a l ␣ { hskip 3 p t } ␣ $ frac { a } { b } $
par
```


expandable
non-expandable
macros
primitives
\section
\emph
\bye
\number<num>
!
\par
\begingroup
a
\box
\hskip<dim>

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
  while len(token_stream) > 0:
    tok = token_stream.popleft()
    if not tok.is_expandable():
      yield tok
    else:
      expanded = expand_once(tok)
      token_stream.expandleft(expanded)
```

```
def expand_once(tok) -> List[Token]:
  # only poplefts from stream
  assert tok.is_expandable()

  if tok.is_macro():
    macro = tok.get_macro()
    parameters = []
    while not macro.valid_params(parameters):
      parameters.append(token_stream.popleft())

    replacement = macro.fill_replacement(parameters)
    return replacement

  if tok.is_expandafter():
    # \expandafter <tok_a> <tok_b> expands <tok_b>
    tok_a = token_stream.popleft()
    tok_b = token_stream.popleft()
    if tok_b.is_expandable():
      expanded = expand_once(tok_b)
      return [tok_a] + expanded
    else:

      return [tok_a, tok_b]

  if tok.is_condition():

    ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
  while len(token_stream) > 0:
    tok = token_stream.popleft()
    if not tok.is_expandable():
      yield tok
    else:
      expanded = expand_once(tok)
      token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
     expandiere \centerline:
```

```
def expand_once(tok) -> List[Token]:
  # only poplefts from stream
  assert tok.is_expandable()

  if tok.is_macro():
    macro = tok.get_macro()
    parameters = []
    while not macro.valid_params(parameters):
      parameters.append(token_stream.popleft())

    replacement = macro.fill_replacement(parameters)
    return replacement

  if tok.is_expandafter():
    # \expandafter <tok_a> <tok_b> expands <tok_b>
    tok_a = token_stream.popleft()
    tok_b = token_stream.popleft()
    if tok_b.is_expandable():
      expanded = expand_once(tok_b)
      return [tok_a] + expanded
    else:
      return [tok_a, tok_b]

  if tok.is_condition():

    ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
\hskip0pt plus 1fil minus 1fil Hello \TeX \hss}
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
\hskip0pt plus 1fil minus 1fil Hello \TeX \hss}
    expandiere \TeX
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:
   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
\hskip0pt plus 1fil minus 1fil Hello \TeX \hss}
    expandiere \TeX
T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX \hss}
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
\hskip0pt plus 1fil minus 1fil Hello \TeX \hss}
    expandiere \TeX
T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX \hss}
```

```
token_stream: deque[Token]

def expand_token_stream() -> Iterable[Token]:
 while len(token_stream) > 0:
  tok = token_stream.popleft()
  if not tok.is_expandable():
   yield tok
  else:
   expanded = expand_once(tok)
   token_stream.expandleft(expanded)
```

```
def expand_once(tok) -> List[Token]:
 # only poplefts from stream
 assert tok.is_expandable()

 if tok.is_macro():
  macro = tok.get_macro()
  parameters = []
  while not macro.valid_params(parameters):
   parameters.append(token_stream.popleft())

  replacement = macro.fill_replacement(parameters)
  return replacement

 if tok.is_expandafter():
  # \expandafter <tok_a> <tok_b> expands <tok_b>
  tok_a = token_stream.popleft()
  tok_b = token_stream.popleft()
  if tok_b.is_expandable():
   expanded = expand_once(tok_b)
   return [tok_a] + expanded
  else:

   return [tok_a, tok_b]

 if tok.is_condition():

  ...
```

```
\centerline{Hello \TeX}
    expandiere \centerline:
    \def\centerline#1{\line{\hss#1\hss}}
\line{\hss Hello \Tex \hss}
    expandiere \line
\hbox to\hsize{\hss Hello \Tex \hss}
    expandiere \hsize
6.5in{\hss Hello \TeX \hss}
    expandiere \hss
\hskip0pt plus 1fil minus 1fil Hello \TeX \hss}
    expandiere \TeX
T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX \hss}
```

\def<*control sequence*><*parameter text*>{<*replacement text*>}

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

```
\def<control sequence><parameter text>{<replacement text>}

      \def\NP{\mathrm{NP}}
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
% $\row {\alpha^*} -> $(\alpha^*_1, … )$
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
% $\row {\alpha^*} -> $(\alpha^*_1, … )$
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
% $\row {\alpha^*} -> $(\alpha^*_1, … )$

\def\formatname#1#2{Name: #2, #1}
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
% $\row {\alpha^*} -> $(\alpha^*_1, … )$

\def\formatname#1#2{Name: #2, #1}
% \formatname{Max}{Mustermann} -> Name: Mustermann,
Max
```

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\NP{\mathrm{NP}}
% $\NP$ -> $\mathrm{NP}$ -> ...

\def\row#1{(#1_1, #1_2, \ldots, #1_n)}
% $\row{x}$ -> $(x_1, x_2, \ldots, x_n)$
% $\row {\alpha^*} -> $(\alpha^*_1, … )$

\def\formatname#1#2{Name: #2, #1}
% \formatname{Max}{Mustermann} -> Name: Mustermann,
Max
```

$(x_1, \ldots, x_n)$

“undelimited” Parameter (folgt # oder am Ende des parameter text)
→ Argument ist das nächste Token, oder {<tokens>}, wobei umschließende Klammern entfernt werden

\def<*control sequence*><*parameter text*>{<*replacement text*>}

`\def<control sequence><parameter text>{<replacement text>}`

```
\def\proclaim #1. #2\par%
%              ^^^^^^^^^^ parameter text
{\medbreak\noindent{\bf#1.\enspace}
{\sl#2}\par\medbreak}
```

“delimited” Parameter *#x* (folgt nicht #, sondern <non-parameter-tokens>)
→ Argument ist die (ggf. leere) Sequenz bis zum ersten Vorkommen der <non-parameter-tokens>

\def<*control sequence*><*parameter text*>{<*replacement text*>}

```
\def\proclaim #1. #2\par%
%              ^^^^^^^^^^ parameter text
{\medbreak\noindent{\bf#1.\enspace}
{\sl#2}\par\medbreak}
```

“delimited” Parameter #*x* (folgt nicht #, sondern <non-parameter-tokens>)
→ Argument ist die (ggf. leere) Sequenz bis zum ersten Vorkommen der <non-parameter-tokens>

```
\proclaim Thesis 1. Jede intuitiv berechenbare
Funktion ist Turing-berechenbar und umgekehrt.\par
* #1<-Thesis 1
* #2<-Jede intuitiv berechenbare Funktion ...
```

Thesis 1. *Jede intuitiv berechenbare Funktion ist Turing-berechenbar und umgekehrt.*

14 How TEX Breaks Paragraphs into Lines

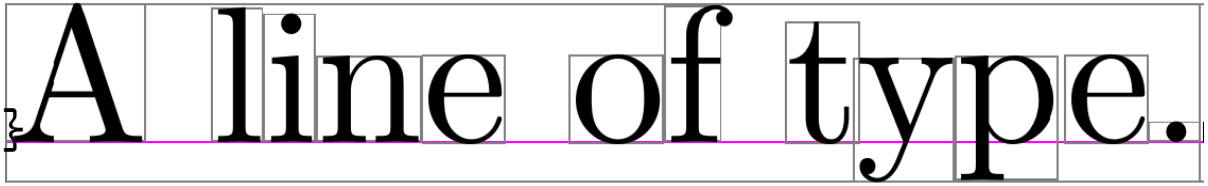
15 How TEX Makes Lines into Pages

Schriftsatz

Boxes und Glue

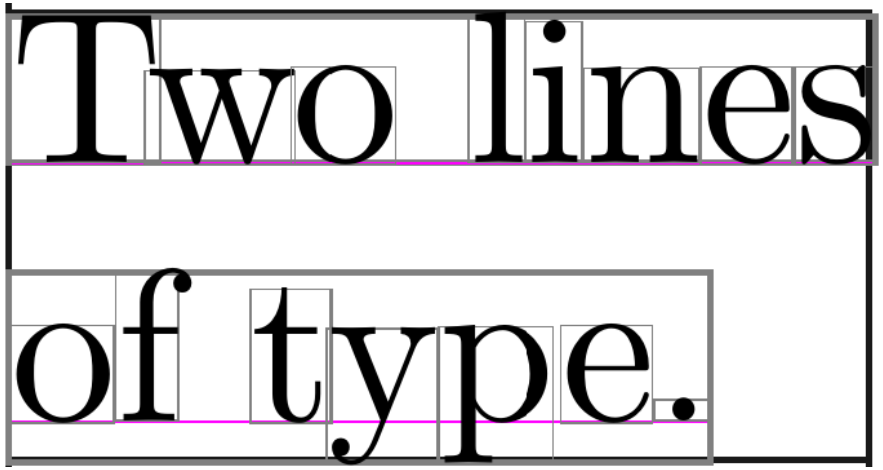
Horizontale Liste an Boxes

```
\hbox{A line of type.}
```



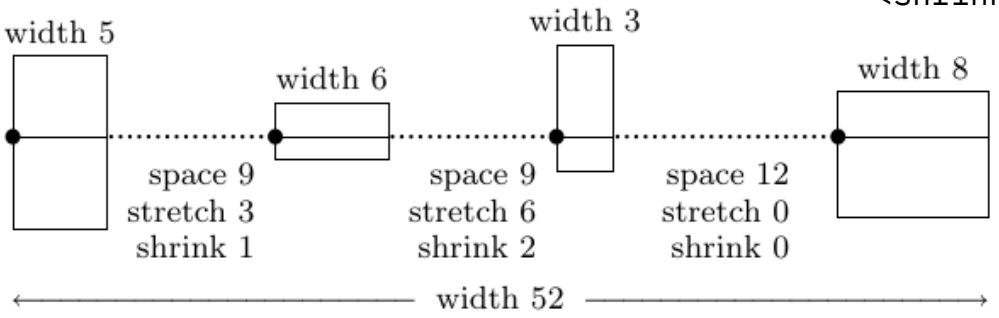
Vertikale Liste an Boxes

```
\vbox{\hbox{Two lines}%
  \hbox{of type.}}
```



Boxes und **Glue**

```
\hskip <width> plus<strech> minus <shrink>
\vskip <width> plus<strech> minus <shrink>
\hspace{\vskip <width> plus<strech> minus
<shrink>}
\vspace{\vskip <width> plus<strech> minus
<shrink>}
```



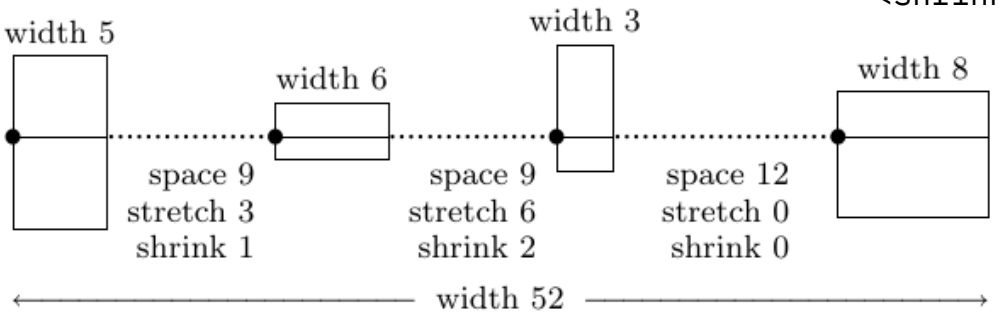
„Spacing“ hat eine

- *natürliche Weite,*
- maximale *Stretchability*
- maximale *Shrinkability*

etwa wie eine Feder
→ erlaubt Auffüllen von Boxen fester Größe

Boxes und **Glue**

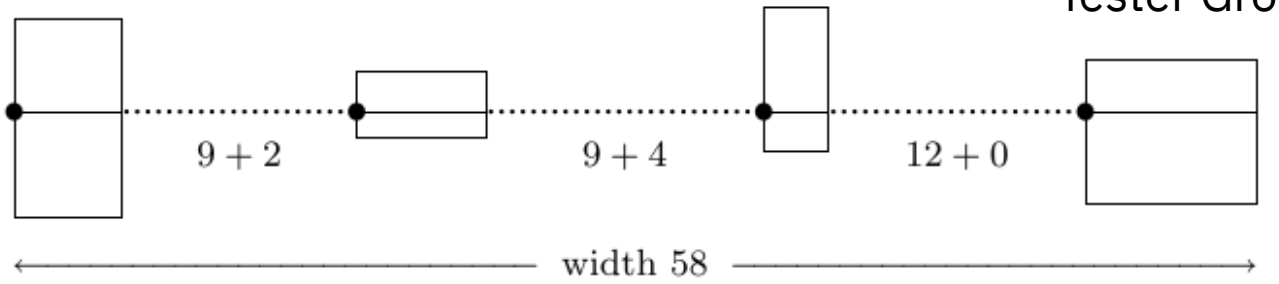
```
\hskip <width> plus<strech> minus <shrink>
\vskip <width> plus<strech> minus <shrink>
\hspace{\vskip <width> plus<strech> minus
<shrink>}
\vspace{\vskip <width> plus<strech> minus
<shrink>}
```



„Spacing“ hat eine

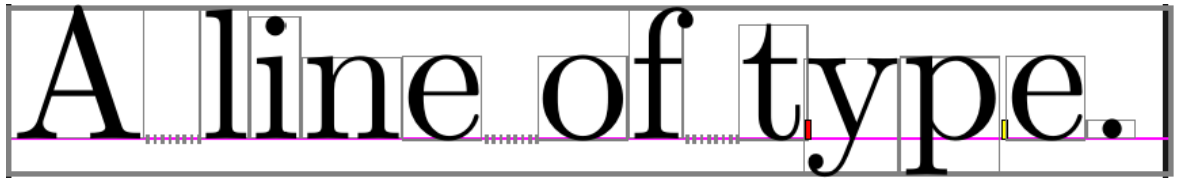
- *natürliche Weite,*
- maximale *Stretchability*
- maximale *Shrinkability*

etwa wie eine Feder
→ erlaubt Auffüllen von Boxen fester Größe



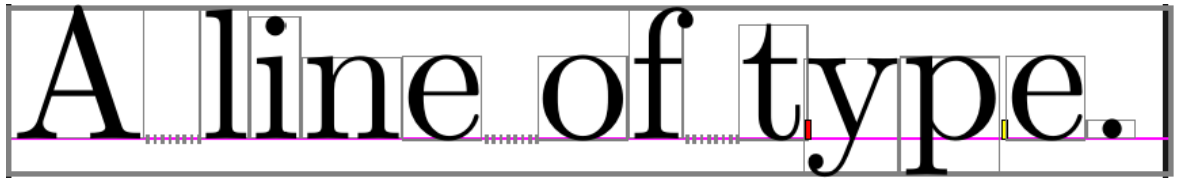
Boxes und **Glue**

```
\hbox to 64pt{A{\hskip 3pt}line{\hskip 3pt}of{\hskip 3pt}type.}
```



Boxes und **Glue**

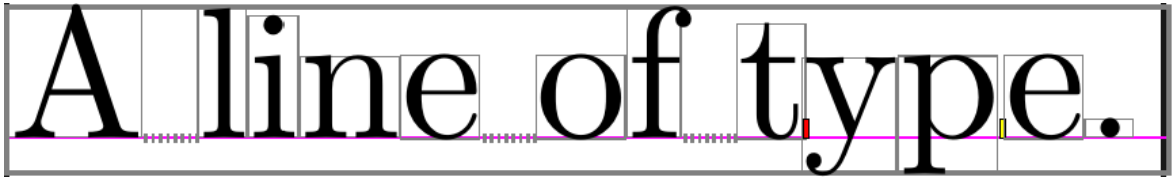
```
\hbox to 64pt{A{\hskip 3pt}line{\hskip 3pt}of{\hskip 3pt}type.}
```



Underfull hbox

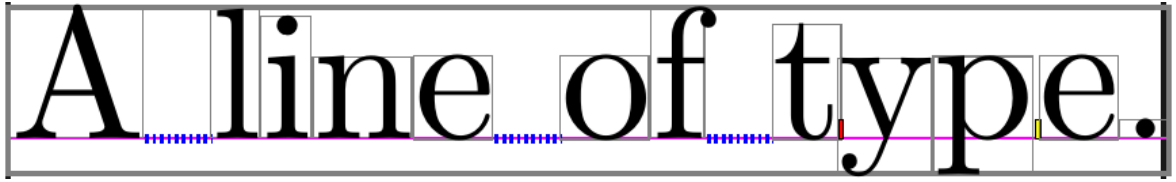
Boxes und **Glue**

```
\hbox to 64pt{A{\hskip 3pt}line{\hskip 3pt}of{\hskip 3pt}type.}
```



Underfull hbox

```
\hbox to 64pt{A{\hskip 3pt plus 1pt}line{\hskip 3pt plus 1pt}
of{\hskip 3pt plus 1pt}type.}
```



3a. Horizontales Typesetting

SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 1119–1184 (1981)

Breaking Paragraphs into Lines*

DONALD E. KNUTH AND MICHAEL F. PLASS
Computer Science Department, Stanford University, Stanford, California 94305, U.S.A.

SUMMARY

This paper discusses a new approach to the problem of dividing the text of a paragraph into lines of approximately equal length. Instead of simply making decisions one line at a time, the method considers the paragraph as a whole, so that the final appearance of a given line might be influenced by the text on succeeding lines. A system based on three simple primitive concepts called 'boxes', 'glue', and 'penalties' provides the ability to deal satisfactorily with a wide variety of typesetting problems in a unified framework, using a single algorithm that determines optimum breakpoints. The algorithm avoids backtracking by a judicious use of the techniques of dynamic programming. Extensive computational experience confirms that the approach is both efficient and effective in producing high-quality output. The paper concludes with a brief history of line-breaking methods, and an appendix presents a simplified algorithm that requires comparatively few resources.

3a. Horizontales Typesetting

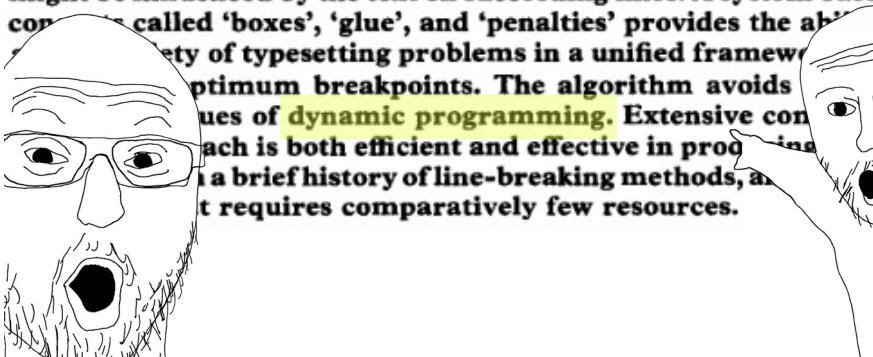
SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 1119–1184 (1981)

Breaking Paragraphs into Lines*

DONALD E. KNUTH AND MICHAEL F. PLASS
Computer Science Department, Stanford University, Stanford, California 94305, U.S.A.

SUMMARY

This paper discusses a new approach to the problem of dividing the text of a paragraph into lines of approximately equal length. Instead of simply making decisions one line at a time, the method considers the paragraph as a whole, so that the final appearance of a given line might be influenced by the text on succeeding lines. A system based on three simple primitive con called 'boxes', 'glue', and 'penalties' provides the ab deal satisfactorily with ety of typesetting problems in a unified framew single algorithm that ptimum breakpoints. The algorithm avoids ng by a judicious use ues of dynamic programming. Extensive con l experience confirms ach is both efficient and effective in prod lity output. The paper a brief history of line-breaking methods, a resents a simplified t requires comparatively few resources.



Simple example of minimum raggedness metric [edit]

For the input text

```
AAA BB CC DDDDD
```

with line width 6, a greedy algorithm that puts as many words on a line as possible while preserving order before moving to the next line, would produce:

```
------    Line width: 6
AAA BB    Remaining space: 0
CC        Remaining space: 4
DDDDD     Remaining space: 1
```

The sum of squared space left over by this method is $0^2 + 4^2 + 1^2 = 17$. However, the optimal solution achieves the smaller sum $3^2 + 1^2 + 1^2 = 11$:

```
------    Line width: 6
AAA       Remaining space: 3
BB CC     Remaining space: 1
DDDDD     Remaining space: 1
```

The difference here is that the first line is broken before `BB` instead of after it, yielding a better right margin and a lower cost 11.

Simple example of minimum raggedness metric [edit]

For the input text

```
AAA BB CC DDDDD
```

with line width 6, a greedy algorithm that puts as many words on a line as possible while preserving order before moving to the next line, would produce:

```
------    Line width: 6
AAA BB    Remaining space: 0
CC        Remaining space: 4
DDDDD     Remaining space: 1
```

The sum of squared space left over by this method is $0^2 + 4^2 + 1^2 = 17$. However, the optimal solution achieves the smaller sum $3^2 + 1^2 + 1^2 = 11$:

```
------    Line width: 6
AAA       Remaining space: 3
BB CC     Remaining space: 1
DDDDD     Remaining space: 1
```

The difference here is that the first line is broken before `BB` instead of after it, yielding a better right margin and a lower cost 11.

3a. Horizontales Typesetting

3a. Horizontales Typesetting

Gegeben: Absatzbreite *w*, Sequenz an Boxen (Zeichen), erlaubten Breakpoints und Glue. Beginnt mit `\parindent` und hört mit `\parfillskip` auf.

3a. Horizontales Typesetting

Gegeben: Absatzbreite *w*, Sequenz an Boxen (Zeichen), erlaubten Breakpoints und Glue. Beginnt mit `\parindent` und hört mit `\parfillskip` auf.

Gesucht: Auswahl der Breakpoints in der Sequenz (Zeilen der Länge *w* = Intervall zwischen konsekutiven Breakpoints) sodass „Hässlichkeit“ des Absatzes minimiert wird.

3a. Horizontales Typesetting

Gegeben: Absatzbreite *w*, Sequenz an Boxen (Zeichen), erlaubten Breakpoints und Glue. Beginnt mit `\parindent` und hört mit `\parfillskip` auf.

Gesucht: Auswahl der Breakpoints in der Sequenz (Zeilen der Länge *w* = Intervall zwischen konsekutiven Breakpoints) sodass „Hässlichkeit“ des Absatzes minimiert wird.

Def:

- *Badness* einer Zeile ≈ 100 (tatsächlicherStretch/natürlicherStretch)^3
- *Penalty* einer Zeile: Explizite Kosten, die für unschöne Breaks vergeben werden (etwa innerhalb eines Worts, bei explizitem `\penalty`)
- *Demerits („Hässlichkeit“)* = Summe Badness + Penalty aller Zeilen

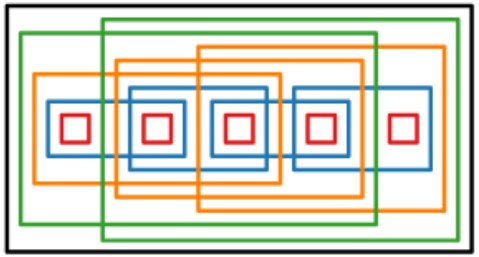
In olden times when wishing still helped one, there lived a king -.741 41
whose daughters were all beautiful; and the youngest was so .877 67
beautiful that the sun itself, which has seen so much, was aston- -.425 8
ished whenever it shone in her face. Close by the king's castle lay -.816 54
a great dark forest, and under an old lime-tree in the forest was -.191 1
a well, and when the day was very warm, the king's child went .107 0
out into the forest and sat down by the side of the cool fountain; -.588 16
and when she was bored she took a golden ball, and threw it up -.234 1
on high and caught it; and this ball was her favorite plaything. .000 0

Greedy Ergebnis: 188

In olden times when wishing still helped one, there lived a .780 47
king whose daughters were all beautiful; and the youngest was .346 1
so beautiful that the sun itself, which has seen so much, was .667 30
astonished whenever it shone in her face. Close by the king's .514 14
castle lay a great dark forest, and under an old lime-tree in the .027 0
forest was a well, and when the day was very warm, the king's .173 1
child went out into the forest and sat down by the side of the .346 4
cool fountain; and when she was bored she took a golden ball, .275 2
and threw it up on high and caught it; and this ball was her .593 21
favorite plaything. .002 0

Bestes Ergebnis: 120

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.



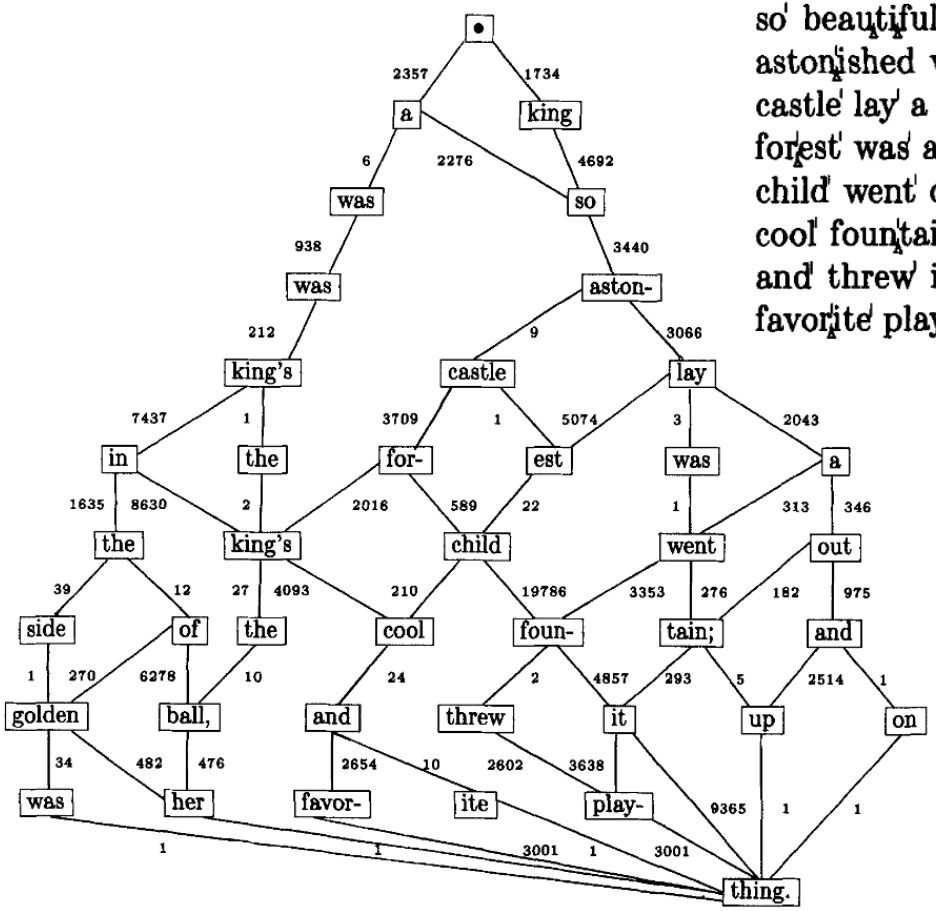
Dynamisches Programmieren

- zerlegt Instanz in **überlappende** Teilinstanzen,

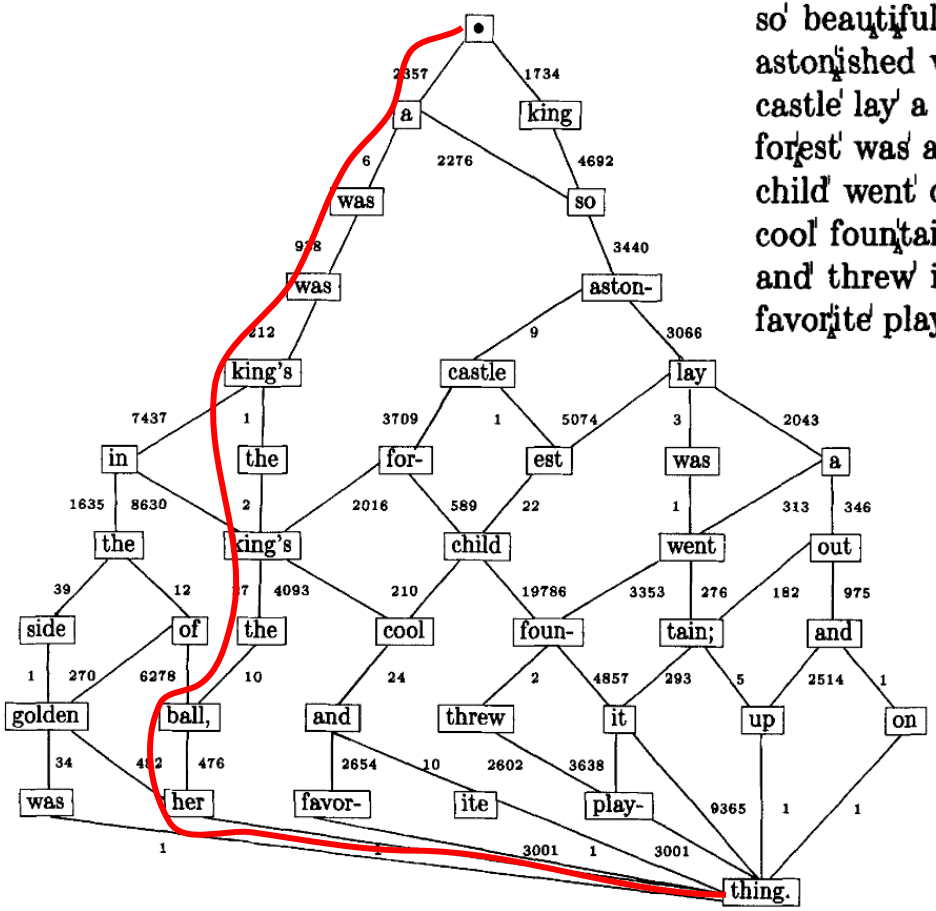
  d.h. Teilinstanzen haben oft dieselben Teilteilinstanzen.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

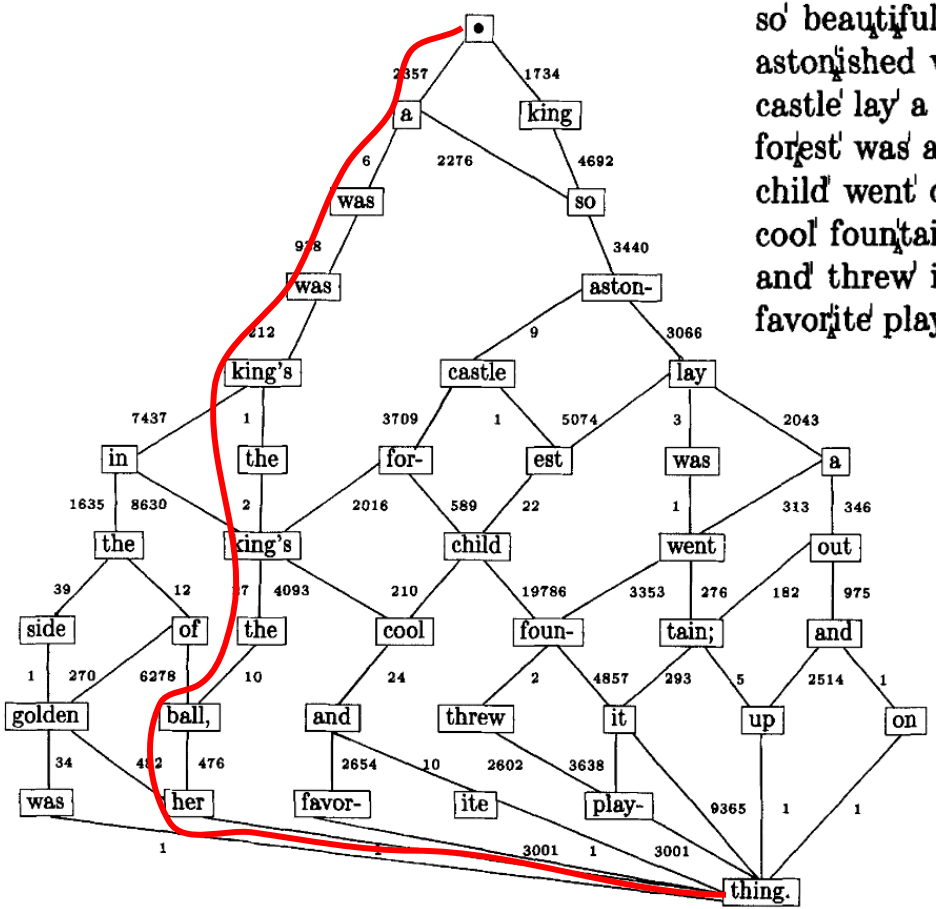
In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.



In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.



In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.



In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Q: Zeitkomplexität?

3b. Vertikales Typesetting

einfach:
symmetrisch zum
horizontalen Fall

3b. Vertikales Typesetting

einfach:
symmetrisch zum horizontalen Fall

3b. Vertikales Typesetting

einfach:
symmetrisch zum horizontalen Fall

TEX breaks lists of lines into pages by computing badness ratings and penalties, more or less as it does when breaking paragraphs into lines. But pages are made up one at a time and removed from TEX's memory; there is no looking ahead to see how one page break will affect the next one. In other words, TEX uses a special method to find the optimum breakpoints for the lines in an entire paragraph, but it doesn't attempt to find the optimum breakpoints for the pages in an entire document. The computer doesn't have enough high-speed memory capacity to remember the contents of several pages, so TEX simply chooses each page break as best it can, by a process of "local" rather than "global" optimization.

3b. Vertikales Typesetting

~~einfach: symmetrisch zum horizontalen Fall~~

TEX breaks lists of lines into pages by computing badness ratings and penalties, more or less as it does when breaking paragraphs into lines. But pages are made up one at a time and removed from TEX's memory; there is no looking ahead to see how one page break will affect the next one. In other words, TEX uses a special method to find the optimum breakpoints for the lines in an entire paragraph, but it doesn't attempt to find the optimum breakpoints for the pages in an entire document. The computer doesn't have enough high-speed memory capacity to remember the contents of several pages, so TEX simply chooses each page break as best it can, by a process of "local" rather than "global" optimization.

Korollar: Absatz-Bauen und Seiten-Bauen können sich nicht abstimmen

3b. Vertikales Typesetting

~~einfach: symmetrisch zum horizontalen Fall~~

TEX breaks lists of lines into pages by computing badness ratings and penalties, more or less as it does when breaking paragraphs into lines. But pages are made up one at a time and removed from TEX's memory; there is no looking ahead to see how one page break will affect the next one. In other words, TEX uses a special method to find the optimum breakpoints for the lines in an entire paragraph, but it doesn't attempt to find the optimum breakpoints for the pages in an entire document. The computer doesn't have enough high-speed memory capacity to remember the contents of several pages, so TEX simply chooses each page break as best it can, by a process of "local" rather than "global" optimization.

Korollar: Absatz-Bauen und Seiten-Bauen können sich nicht abstimmen

das gilt auch für den horizontalen Fall:

▸ **EXERCISE 14.15**

Since TEX reads an entire paragraph before it makes any decisions about line breaks, the computer's memory capacity might be exceeded if you are typesetting the works of some philosopher or modernistic novelist who writes 200-line paragraphs. Suggest a way to cope with such authors.

Ende

The whole TEX language has been presented in the previous chapters; we have finally reached the end of our journey into previously uncharted territory. Hurray! Victory!