# Maschinen, Formate, Tastaturen

# Ein leichterer TeX-Alltag dank moderner Entwicklungen.

DANTE-Herbsttagung 2010 Trier

Arno Trautmann

# Inhalt

# Worum geht es?

- Einfacherers Arbeiten mit TEX durch:
- Maschinen: Neuentwicklungen der letzten Jahre (XƎTEX, luaTEX), Nutzen für den normalen Anwender
- Formate: LATEX3-Kernel und Paket, die darauf basieren
- Tastaturen: Neo-Tastaturlayout

# Warum neue Maschinen?

- TEX wird seit langer Zeit nicht weiterentwickelt
- Änderungen nur noch für Bugfixes

$\Longrightarrow$ extrem stabil

# Warum neue Maschinen?

- TEX wird seit langer Zeit nicht weiterentwickelt
- Änderungen nur noch für Bugfixes

$\Longrightarrow$ extrem stabil

- im Laufe der Zeit gibt es aber neue Anforderungen:
- mehr Speicher für Variablen
- neue Codierungen (Eingabe, Schriften)
- pdf als Ausgabe-Format (pdf-Features nur durch direktes Erzeugen verfügbar)
- neue Schrifttechnologien

# Maschinen

- kurzer Überblick über die Entwicklung von TEX78 bis luaTEX (aktuelle Version unter http://github.com/alt/overview)

# Vorteile von XeTeX und luaTeX– im Alltag

## Vollständige utf8-Unterstützung

$\Longrightarrow$ nie mehr Probleme beim Austausch von Dokumenten, nie wieder kaputte Umlaute, keine umständliche Eingabe spezieller Zeichen anderer Sprachen, beliebig viele Sonderzeichen – ganz zu schweigen von asiatischem Satz ...

# Vorteile von XeTeX und luaTeX– im Alltag

## Vollständige utf8-Unterstützung

⟹ nie mehr Probleme beim Austausch von Dokumenten, nie wieder kaputte Umlaute, keine umständliche Eingabe spezieller Zeichen anderer Sprachen, beliebig viele Sonderzeichen – ganz zu schweigen von asiatischem Satz ...

## moderne Schrifttechnologien (ttf, otf, aat)

Keine speziellen Dateien zum Schrifteinbinden benötigt, Portabilität von Schriften anderer Anwendungen

⟹ modernste, neuste Schriften sind in vollem Funktionsumfang leicht verwendbar

# Nachteile neuer Maschinen

- etwas Neues muss gelernt werden
- weniger stabil (bugs/neue Features)
- nicht rückwärtskompatibel
- auf veralteten Systemen nicht verfügbar
- Wermutstropfen bei XeTeX: Mikrotypographie nicht wie bei pdfTeX möglich
- Verwirrung und Unklarheiten durch neue Vielfalt (Welche Maschine, welches Format, welche Distribution?)

# Nachteile neuer Maschinen

- etwas Neues muss gelernt werden
- weniger stabil (bugs/neue Features)
- nicht rückwärtskompatibel
- auf veralteten Systemen nicht verfügbar
- Wermutstropfen bei XeTeX: Mikrotypographie nicht wie bei pdfTeX möglich
- Verwirrung und Unklarheiten durch neue Vielfalt (Welche Maschine, welches Format, welche Distribution?)

$\Longrightarrow$ Was lehrt man den Anfänger?

# Formate

- fast niemand verwendet INITEX als Programm, fast immer wird ein Format verwendet, eine große Sammlung von Makros als Binärdatei
- plainTEX: Do It Yourself!
- LATEX: A Document Preparation System
- ConTEXt: monolithischer als LATEX, aber sehr flexibel einseztbar

# Formate

- fast niemand verwendet INITEX als Programm, fast immer wird ein Format verwendet, eine große Sammlung von Makros als Binärdatei
- plainTEX: Do It Yourself!
- LATEX: A Document Preparation System
- ConTEXt: monolithischer als LATEX, aber sehr flexibel einseztbar – „kann alles außer Dokumentation“

# Formate

- fast niemand verwendet INITEX als Programm, fast immer wird ein Format verwendet, eine große Sammlung von Makros als Binärdatei
- plainTEX: Do It Yourself!
- LATEX: A Document Preparation System
- ConTEXt: monolithischer als LATEX, aber sehr flexibel einseztbar – „kann alles außer Dokumentation“
- alle Formate können an Maschinen angepasst werden, um bestimmte Features leicht zugänglich zu machen: pdfLATEX, XƎLATEX, luaLATEX, ConTEXt Mk II bzw. IV, plain luaTEX etc.

# The neverending story: LaTeX3

- LaTeX $2_\varepsilon$: eigentlich als Zwischenversion zu LaTeX3 geplant, inzwischen eingefroren und meistverwendetes TeX-Format
- keine Weiterentwicklung im Kernel, keine Korrektur von Designfehlern
- LaTeX3 als lange währendes, unbekanntes Mysterium ...
- immer noch nicht als eigenständiges Format verfügbar

# The neverending story: LaTeX3

- LaTeX 2ε: eigentlich als Zwischenversion zu LaTeX3 geplant, inzwischen eingefroren und meistverwendetes TeX-Format
- keine Weiterentwicklung im Kernel, keine Korrektur von Designfehlern
- LaTeX3 als lange währendes, unbekanntes Mysterium ...
- immer noch nicht als eigenständiges Format verfügbar
- aber als Paket auf LaTeX 2ε
- einige Pakete bauen bereits auf LaTeX3-Code auf
- z. B. `siunitx`, `fontspec`, ...
- erfordert ε-TeX als gemeinsamen Nenner von pdfTeX und XeTeX
- inzwischen werden sogar luaTeX-features direkt genutzt (floating point modul)

# Designmerkmale von LaTeX3 (Auswahl)

- Versuch einer „gesunden“ (sane) Programmierebene, die von TeX abstrahiert
- gute, übersichtliche Namensgebung von internen Makros (Problem in LaTeX $2_\varepsilon$ durch evolutionäre Entwicklung)
- sinnvolle Strukturierung etc. etc. etc.

$\Longrightarrow$ Entwicklung durch Planung, nicht durch zu frühes Anwenden

# Designmerkmale von LaTeX3 (Auswahl)

- anderes Catcode-Regime: `\ExplSyntaxOn/Off` (als `expl3` in LaTeX $2_\varepsilon$) (vgl. `\makeatletter \makeatother`
- Leerzeichen werden ignoriert (~ wenn nötig)
- Namen: Das `@` als Trennstelle in Namen wird vom `_` übernommen
- Interne Namen enden mit einem `:`
- Nach dem `:` werden erwartete Argumente dargestellt

# Designmerkmale von LaTeX3 (Auswahl)

- anderes Catcode-Regime: `\ExplSyntaxOn/Off` (als `expl3` in LaTeX $2_\varepsilon$) (vgl. `\makeatletter \makeatother`
- Leerzeichen werden ignoriert (~ wenn nötig)
- Namen: Das `@` als Trennstelle in Namen wird vom `_` übernommen
- Interne Namen enden mit einem `:`
- Nach dem `:` werden erwartete Argumente dargestellt
- `\cs_new:Npn` ( ≈ `\def`)
  `Modulname_Zwischenname_Name : Argumente`

# Argumente (Auswahl)

- `N`: „no manipulation“es Token
- `n`: „no manipulation“, gruppiert
- `x`: „exhaustive expansion“, wie `\edef`
- `c`: baute eine Kontrollsequenz: `\cs_new:cpn{name} = \cs_new:Npn\name`
- `p`: TEX-Parametersequenz (`#1 xyz #2`)
- `T`: true-Zweig für `\if`-Abfragen
- `F`: false-Zweig für `\if`-Abfragen (T, F oder TF möglich!)
- `D`: „Don’t use“ – umbenannte TEX-Primitiva

# Vorteile der Namensgebung

- Konsistente Einteilung in Module möglich (und strikter durchgesetzt)
- Quellcode wesentlich besser lesbar (Wird eine Variable gesetzt? Ein Zähler? Eine Konstante verwendet? ...)
- Anzahl der nötigen Argumente direkt sichtbar (`\use_none:nn`, `\use_none:nnnn`, `\use_ii:nnn`)
- Ändern von Expansion/Auswertung etc. eines Argumentes durch Ändern eines Buchstabens im Argument

# Beispiel für sinnvolle LaTeX3-Konstrukte

angenommen: Der Nutzer kann einen Namen `\name` vorgeben. Aus diesem Namen werden die Befehle `name1`, `name2` und `name-<name1>` gebaut. (`hallo` $\Longrightarrow$ `hallo1`, `hallo2`, `hallo-Inhalt 1`)

**TeX, meist in LaTeX verwendet:**

```
\expandafter\def\csname \name1\endcsname{Inhalt 1}
\expandafter\def\csname \name2\endcsname{Inhalt 2}
\expandafter\def\csname \name-\csname\name \endcsname\endcsname{Inhal
```

# Beispiel für sinnvolle LaTeX3-Konstrukte

angenommen: Der Nutzer kann einen Namen `\name` vorgeben. Aus diesem Namen werden die Befehle `name1`, `name2` und `name-<name1>` gebaut. (`hallo` ⟹ `hallo1`, `hallo2`, `hallo-Inhalt 1`)

**LaTeX3**

```
\ExplSyntaxOn
\cs_new:cpn{\name1}{Inhalt 1}
\cs_new:cpn{\name1}{Inhalt 2}
\cs_new:cpn{\name1-\cs:w \name \cs_end:}{Inhalt 1}
\ExplSyntaxOff
```

# LaTeX 2ε-Versionen

LaTeX 2ε bietet u. a. `\@nameuse` und `\@namedef` – aber recht selten genutzt (mir bis 4 Tage vor diesem Vortrag nicht bekannt, dass es Verwendung findet)

# xparse

## Definieren von Dokumentbefehlen

- xparse gehört nicht zum Kernel, sondern zu den xpackages
- bietet ein sehr effizientes Interface zum Definieren von Dokumentbefehlen (für den Endnutzer gedachte Befehle)

# xparse

## Definieren von Dokumentbefehlen

- `xparse` gehört nicht zum Kernel, sondern zu den `xpackages`
- bietet ein sehr effizientes Interface zum Definieren von Dokumentbefehlen (für den Endnutzer gedachte Befehle)
- `\(Declare/New/Renew/Provide)` – `Document` – `(Command/Environment)`
- erfordert keine LaTeX3-Syntax (nur intern damit programmiert)
- Argumente werden einzeln angegeben (vgl. Tabellenspalten `rcl`)
- verschiedene Arten von Argumenten beliebig mischbar

# xparse

## Anwendung

```
L2 \newcommand\mycommand[2]{<Definition>}
L3 \NewDocumentCommand\mycommand{mm}{<Definition>}
```

# xparse

## Anwendung

L2 `\newcommand\mycommand[2]{<Definition>}`

L3 `\NewDocumentCommand\mycommand{mm}{<Definition>}`

L2 `\newcommand\mycommand[opt][2]{<Definition>}`

L3 `\NewDocumentCommand\mycommand{O{opt}m}{<Definition>}`

# xparse
## Anwendung

L2 `\newcommand\mycommand[2]{<Definition>}`

L3 `\NewDocumentCommand\mycommand{mm}{<Definition>}`

L2 `\newcommand\mycommand[opt][2]{<Definition>}`

L3 `\NewDocumentCommand\mycommand{O{opt}m}{<Definition>}`

L2 ???

L3 `\NewDocumentCommand\mycommand{sO{opt}mt{+}O{2nd opt}mm}{<...>}`

Aufruf:

`\mycommand*[Option1]{Argument} + [Option2] {Argument2}{Argument3}`

# xparse

## Argumenttypen – Auswahl

- `m` mandatorisch, Gruppe oder einzelnes Token
- `o` optional, wird zu `-NoValue-` wenn nicht gegeben
- `O{}` optional, wird zum Argument, wenn nicht gegeben
- `t{}` es wird geprüft, ob das in Klammern gegebene Token vorkommt (boolscher Test)
- `s` testet, ob ein Stern vorkommt (= `t{*}`)
- `d12` optionales Argument, das durch die Token `1` und `2` begrenzt ist, z. B. `t<>`
- ...

# Anwendungsbeispiel

## levelscheme

Ein Paket, das das einfache Zeichnen von atomaren Levelschemata ermöglicht. Intern in `expl3` und `TikZ` geschrieben, Nutzerbefehle mittels `xparse` erstellt.

# fontspec v2

- `fontspec` hat XeTeX für LaTeX-Nutzer erst richtig interessant gemacht
- sehr einfaches und mächtiges Interface für sämtliche Schriftfeatures

# fontspec v2

- `fontspec` hat XeTeX für LaTeX-Nutzer erst richtig interessant gemacht
- sehr einfaches und mächtiges Interface für sämtliche Schriftfeatures
- in Version 2: auch für luaTeX
- gleiche Syntax für den Nutzer, aber (fast) alle features von luaTeX verfügbar
- Code zum Schriftladen von ConTeXt kopiert und verwendet
- größtenteils in LaTeX3-Syntax geschrieben

# fontspec v2

- `\setmain/mono/sansfont{Schriftname}`
- unter XeTeX: Schriftname muss der sein, unter dem das Betriebssystem die Schrift kennt!

⇒ abhängig vom System!

- unter luaTeX: Schriftdatenbank, alle Namen einer Schrift verfügbar (Schriftname, Systemname, Name für Menschen, Dateiname etc.)
- falls vorhanden, werden zugehörige Schnitte bei Bedarf eingebunden

⇒ `\fontspec{Linux Libertine} \textit{test}` lädt die Libertine in kursivem Schnitt

# Was haben TEX und Tastaturbelegungen gemeinsam?

- Sie sind alt!
- die am weitesten verbreitete Belegung `qwerty/z` kam 1868 auf
- TEX kam 1978 in der ersten Version raus
- seitdem hat sich bei TEX einiges getan – bei Tastaturen nicht
- Betrachten wir die Entwicklung der Tastaturbelegungen!

# Tastaturbelegungen – ganz kurz

- Tastaturbelegung – es geht nicht um ergonomische Hardware!
- Anordnung der Tasten, Zeichenauswahl etc. noch von uralten Schreibmaschinen beibehalten
- Belegung durch mechanische Bedingungen gegeben (Verhaken der Stempel ...)
- nur wenige, kleine Änderungen (y und z im Deutschen getauscht, Umlaute)
- kaum Rücksichtnahme auf Erogonomie

# Neo – ein Tastaturlayout

- Hauptziel: ergonomische Tastaturbelegung
- schnelles Schreiben (auch mit unergonomischen Belegungen möglich)
- angenehmes Schreiben (geringe Laufwege für Finger entlasten Gelenke)

# Neo – ein Tastaturlayout

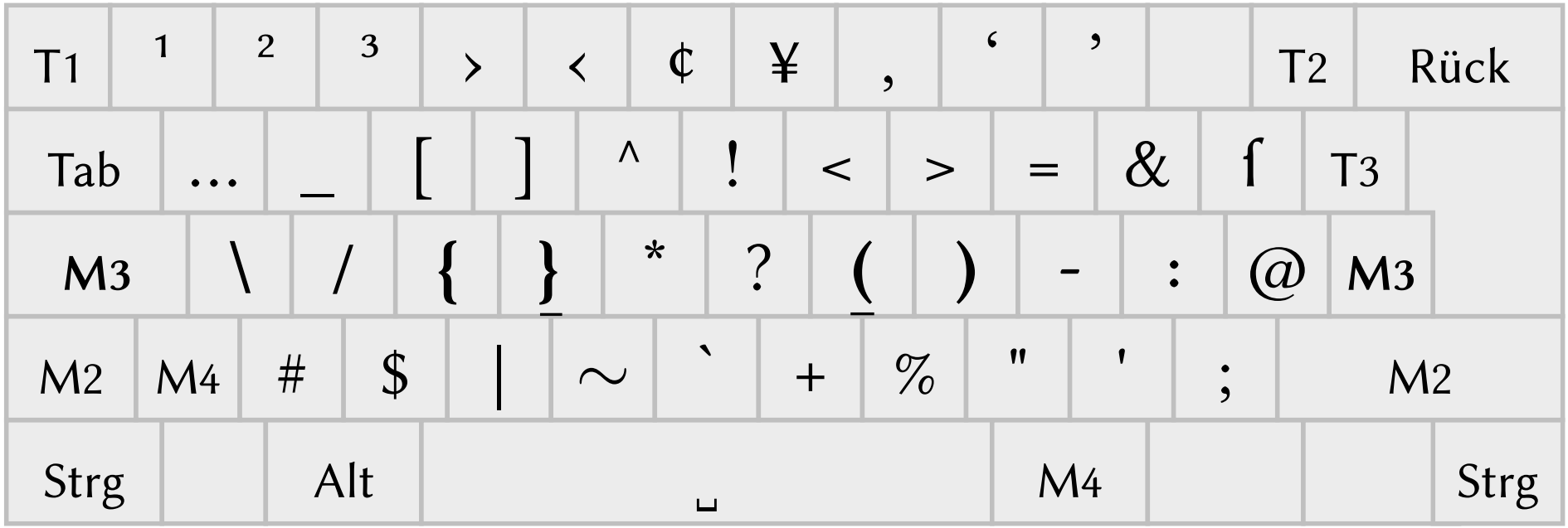
- Hauptziel: ergonomische Tastaturbelegung
- schnelles Schreiben (auch mit unergonomischen Belegungen möglich)
- angenehmes Schreiben (geringe Laufwege für Finger entlasten Gelenke)
- Nebenziel/Nebeneffekt: Große Vielzahl an Sonderzeichen leicht erreichbar
- Ergonomie: statt Rumklicken oder Zahleneingabe: Tastenkombination Ξ√Λℂ⊂∫∀∃∪∩Π□ΨΓΦℚ⇔↦⇒ΣΔ …

# Neo – ein Tastaturlayout mit vielen Sonderzeichen

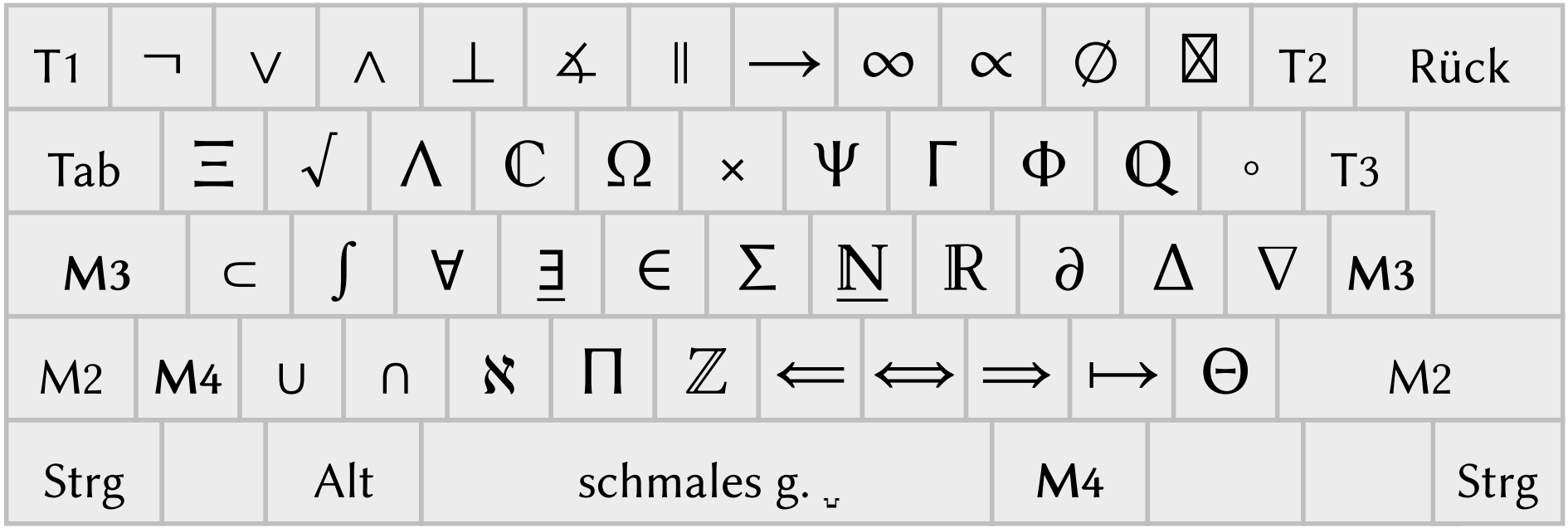
- Sonderzeichen erreichbar über zusätzliche Modifier
- maximal zwei Modifier plus Taste
- für noch mehr Sonderzeichen: Compose-Taste
  (unter Linux und Solaris bekannt, jetzt auch für Windows, und stark erweitert)

# Neo – Ebenen 1 und 2:

# Neo – Ebene 3:



# Neo – Ebene 6:


T1 ¬ ∨ ∧ ⊥ ∡ ∥ ⟶ ∞ ∝ ∅ ⊠ T2 Rück
Tab Ξ √ Λ ℂ Ω × Ψ Γ Φ ℚ ∘ T3
M3 ⊂ ∫ ∀ ∃ ∈ Σ ℕ ℝ ∂ Δ ∇ M3
M2 M4 ∪ ∩ ℵ ∏ ℤ ⟸ ⟺ ⟹ ⟼ Θ M2
Strg Alt schmales g. ␣ M4 Strg

# altTEX?

- experimentelles Paket zum „alternativen TEXen“
- Versuch, häufige Arbeiten einfacher zu gestalten
- Ansätze zur besseren Lesbarkeit und Übersicht von Quellcode
- stark auf Neo aufbauend (Verwendung vieler unicode-Zeichen)

# Itemize ohne altTEX

- Aufzählung im normalen Text:

```
Text
\begin{itemize}
\item erster Punkt
\item zweiter Punkt
\end{itemize}
mehr Text
```

- recht viel Schreibarbeit
- ein guter Editor kann sehr viel Arbeit abnehmen
- aber Code schlecht lesbar

# itemize mit altTEX

- Idee: Nehme ein freies Unicodezeichen (z. B. •)
- weise dem Zeichen eine Bedeutung zu, die seinem Aussehen entspricht (z. B. Aufzählungspunkt)
- implementiere so, dass kein weiterer Code nötig ist (Umgebungen etc.)

# Beispiel: itemize

- Ziel: der Code soll genau das tun, wonach er aussieht:

```
normaler Text

  • erster Punkt
  • zweiter Punkt

weiter im Text
```

⇒ Äufzahlung ohne weitere TEX-Befehle

# Beispiel: itemize

- erster Ansatz: • aktiv machen (TEX sieht dann • als Befehl an)

```
\catcode\`• = \active
```

# Beispiel: itemize

- erster Ansatz: • aktiv machen (TEX sieht dann • als Befehl an)

  ```
  \catcode\`• = \active
  ```

- Bedeutung zuweisen:

  ```
  \let • \item
  ```

# Beispiel: itemize

- erster Ansatz: • aktiv machen (TEX sieht dann • als Befehl an)

```
\catcode\`• = \active
```

- Bedeutung zuweisen:

```
\let • \item
```

- Bereits Verbesserung im Lesefluss:

```
\begin{itemize}
• erster Punkt
• zweiter Punkt
\end{itemize}
```

# Beispiel: itemize

- nächster Schritt: Umgebung `\begin{verbatim}` einsparen!
- dazu: bei jedem Auftreten von • prüfen, ob man bereits in der Aufzählung ist:

# Beispiel: itemize

- nächster Schritt: Umgebung `\begin{verbatim}` einsparen!
- dazu: bei jedem Auftreten von • prüfen, ob man bereits in der Aufzählung ist:

• in Aufzählung?

ja ⟹ `\item`

nein ⟹ `\begin{itemize} \setbooltrue{insideitemize}\item`

# Beispiel: itemize

- schließlich: `\end{verbatim}` einsparen
- Problem: kein Zeichen zum Beenden verwendet!

$\Longrightarrow$ wann weiß TEX, dass die Umgebung beendet werden soll?

# Beispiel: itemize

- schließlich: `\end{verbatim}` einsparen
- Problem: kein Zeichen zum Beenden verwendet!

⇒ wann weiß TEX, dass die Umgebung beendet werden soll?

- Ansatz: Eindeutige Sequenz beendet die Umgebung: Doppeltes Zeilenende (= Leerzeile)

⇒ definiere Zeilenende als Makro, das überprüft, ob es ein- oder zweimal aufgetreten ist

- falls zweimal: beende Umgebung und setze Zeilenende auf normale Bedeutung zurück

# Beispiel: itemize

```
\cs_new:Npn\newitemi{%
  \if_meaning:w\insideitemizei\inside%
    \cs_set_eq:NN\altlastitem1%
    \exp_after:wN\item%
  \else:%
    \begin{itemize}%
    \cs_set_eq:NN\insideitemizei\inside%
    \cs_set_eq:NN\altlastitem1%
    \makeenteractive%
    \exp_after:wN\item%
  \fi:
}
```

# altTEX für den Mathesatz

## Unicodezeichen im Mathesatz

- Schreiben von Unicodezeichen als Mathesatz: Paket `unicode-math` von Will Robertson

# altTEX für den Mathesatz

## Unicodezeichen im Mathesatz

- Schreiben von Unicodezeichen als Mathesatz: Paket `unicode-math` von Will Robertson
- Ermöglicht direkte Eingabe aller mathematischen Zeichen unter Verwendung von OpenType-Matheschriften
- `∫₀⁸ x² = 170 ⅔` $\int_0^8 x^2 = 170\frac{2}{3}$

# altTEX für den Mathesatz

## Hoch- und Tiefstellen

- Hoch- und tiefgestellte Zahlen können sehr lästig werden (z. B. Tensorrechnung)

$\Longrightarrow$ Ziel: Einsparen von Gruppierungen!

- Ansatz: Definiere _ und ^ als Makros, die ihr Argument bis zum nächsten Leerzeichen suchen, falls keine Klammer geöffnet wird
- Alles zwischen _,^ und einem Leerzeichen wird als Argument hoch- bzw. tiefgestellt
- erspart immens viel lästige Schreibarbeit
- nur für kurze Argumente geeignet, da aber am nützlichsten
- bei langen Argumenten: wie üblich gruppieren

# altTEX für den Mathesatz

## Klammersetzen

- Klammern müssen fast immer mit `\left` und `\right` gesetzt werden:

```
1 \[(\ frac  12)  \ left (\ frac  12  \ right )\]
```

$$(\frac{1}{2})\left(\frac{1}{2}\right)$$

$\Longrightarrow$ Ziel: Klammern sollen immer angepasst werden

- Ansatz: Sichere Bedeutung von `()` in Makros `\openbrace` `\closebrace`
- mache `()` aktiv
- definiere `()` als `\left\openbrace` bzw. `\right\openbrace`

# altTEX für den Mathesatz

## Klammersetzen

- Klammern müssen fast immer mit `\left` und `\right` gesetzt werden:

```
1 \[(\ frac  12)  \ left (\ frac  12  \ right )\]
```

$$(\frac{1}{2})\left(\frac{1}{2}\right)$$

$\Longrightarrow$ Ziel: Klammern sollen immer angepasst werden

- Ansatz: Sichere Bedeutung von `()` in Makros `\openbrace \closebrace`
- mache `()` aktiv
- definiere `()` als `\left\openbrace` bzw. `\right\openbrace`
- Problem: Jegliche andere Verwendung von Klammern (im Text, bei `picture`-Umgebung u. ä.) funktioniert nicht mehr!

$\Longrightarrow$ Abfangen durch geschicktere Definition, Überprüfen des Mathemodus etc.

# altTEX für den Mathesatz

## Matrizen

- Eingabe von Matrizen ist recht aufwändig und unübersichtlich:

```
1 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Unicode verfügbt über Klammern, die zum Einschließen von Matrizen geeignet sind
- mit Neo leicht zu erzeugen über Compose-Funktion:

# altTEX für den Mathesatz

## Matrizen

- Eingabe von Matrizen ist recht aufwändig und unübersichtlich:

```
1 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Unicode verfügbt über Klammern, die zum Einschließen von Matrizen geeignet sind
- mit Neo leicht zu erzeugen über Compose-Funktion:

```
♫ x <Klammertyp> <Anzahl Spalten>
♫ x ( 3
⎛ ⎞
⎜ ⎟
⎝ ⎠
```

# altTEX für den Mathesatz

## Matrizen

- Idee: wieder aktive Zeichen
- erstes Klammerzeichen (links oben) beginnt die Matrix, setzt andere Zeichen auf die entsprechenden Makros
- alle rechten Klammerzeichen sorgen für einen Zeilenumbruch in der Matrix
- zur Sicherheit: rechte Klammerzeichen ignorieren das nächste Zeichen
  $\Longrightarrow$ verhindert Doppelausführung, falls rechte und linke Begrenzungen identische Zeichen sind
- das letzte rechte Zeichen beendet die Matrix

# altTEX für den Mathesatz

## Matrizen

- weitere mögliche Verbesserung: Definiere den Tabulator als Zellentrennzeichen & $\Longrightarrow$ keine Steuerzeichen stören die Matrix, sie passt hervorragend ins Dokument, an Tabulatoren ausgerichtet

# altTEX für den Mathesatz

## Matrizen

- weitere mögliche Verbesserung: Definiere den Tabulator als Zellentrennzeichen & ⟹ keine Steuerzeichen stören die Matrix, sie passt hervorragend ins Dokument, an Tabulatoren ausgerichtet
- Nachteil: Schreiben mitten in der Zeile ist nicht möglich, da TEX zeilenweise einliest:

```
a   = ⎛ ⎞
      ⎜ ⎟
      ⎝ ⎠ + b
```

Außerdem keine Matrizen mit verschiedenen Klammern möglich ⟹ ließe sich aber implementieren, wenn die Eingabe möglich wäre.

# Quellen, Links, Verweise

Quellcode für altTEX, TEX-overview, `levelscheme` und diesem Vortrag:
http://github.com/alt
LATEX3 Quellen und Dokumentation: `texdoc source3` in einem aktuellen TEXlive und unter
http://www.latex-project.org/latex3.html
fontspec, unicode-math und alles andere von Will Robertson:
http://github.com/wspr
The official successor of TEX:
http://river-valley.tv/tug-2010/an-earthshaking-announcement