Forth, die neue Synthese:

seedForth

ein minimaler Forth-Kern, den man wachsen lassen kann

Ulrich Hoffmann <uho@xlerb.de>

Überblick

seedForth

- Einleitung: Forth, die neue Synthese
- preForth (simpleForth, Forth)
- Aufteilung von Host und Target
- seedForth
- Zusammenfassung und Ausblick

Einleitung:
Forth, die neue Synthese

- Projekt mit Andrew Read
  - die Grundprinzipien von Forth verstehen
  - zerlegen und zusammensetzen
  - Basis für neues, modernes Forth

Forth, die neue Synthese

- EuroForth 2016

  Implementing the Forth Inner Interpreter in High Level Forth

- Forth 2017

  Stack der Stacks strings auf dem Datenstack

- EuroForth 2017

  handler based outer interpreter

- Forth 2018

  preForth, simpleForth, Forth

Forth, die neue Synthese

- Forth überall (so viel wie möglich)
- bootstrap-fähiges selbst generierendes System
- vollständig transparent
- einfach zu verstehen
- auf der Suche nach Einfachheit
- Analogie zur Biologie (und Physik und Technik)
- zerlegen und zusammensetzen

preForth

- vorgestellt auf der letztjährigen Tagung

- bootstrap-fähiges, selbst-generierendes System
- vollständig transparent
- einfach zu verstehen

preForth

Kann Forth aus weniger als Forth entstehen?

- Was kann man weglassen?
  - kein DOES>, BASE, STATE
  - kein <# # #>
  - kein CATCH/THROW
  - kein IMMEDIATE
  - keine Kontrollstrukturen
    IF ELSE THEN BEGIN WHILE REPEAT UNTIL
  - nur :
  - kein Speicher @ ! CMOVE ALLOT ,
  - kein input stream
  - kein Dictionary, kein EXECUTE EVALUATE
  - nicht interaktiv

preForth

- Was bleibt?
  - Datenstack, Returnstack
  - nur **`?EXIT`** und Rekursion als Kontrollstruktur
  - colon-Definitions
  - optional: Tail-Call-Optimierungs
  - Ein- und Ausgabe via **`KEY`**/**`EMIT`**
  - vorzeichenbehaftete Dezimalzahlen (eine Zelle)
  - Zeichen-Literalte in 'x'-Notation
  - dezimale Zahlenausgabe (eine Zelle)

simpleForth

- Bequemeres aber immer noch einfaches nicht-interaktives Forth
- als Werkzeug zum Erzeugen eines intertaktiven, vollständigen Forth Systems

simpleForth

simpleForth

- preForth ist Turing-vollständig

simpleForth

- preForth ist Turing-vollständig

  Ein vollständiges Forth in preForth zu schreiben ist möglich...
  ... aber sinnlos.

simpleForth

- preForth ist Turing-vollständig

  Ein vollständiges Forth in preForth zu schreiben ist möglich...
  ... aber sinnlos.

- preForth erweitern: simpleForth

simpleForth

- simpleForth is wie preForth
- preForth $\subset$ simpleForth

- zusätzlich:
  - Kontroll-Structuren: IF ELSE THEN BEGIN WHILE REPEAT UNTIL
  - Definitionen mit und ohne Header
  - Speicher: @ ! c@ c! allot c, ,
  - VARIABLE CONSTANT
  - ['] EXECUTE
  - immediate Definitionen

Bootstrapping Forth

- vollständiges, interaktives Forth ("*Forth*") in simpleForth

- die neue Synthese:
    - handler basierter Text-Interpretierer
    - dual words
    - dynamische Speicherverwaltung
    - ...

- geht - aber nicht wirklich zufriedenstellend

Beobachtungen / Unschönheiten

- "doppelte" Beschreibung von
  - Kontrollstrukturen
  - Struktur der Dictionary-Struktur (Header)

  1. für das generierte Forth
  2. zur Benutzung im interaktiven System

- die Suche geht weiter...

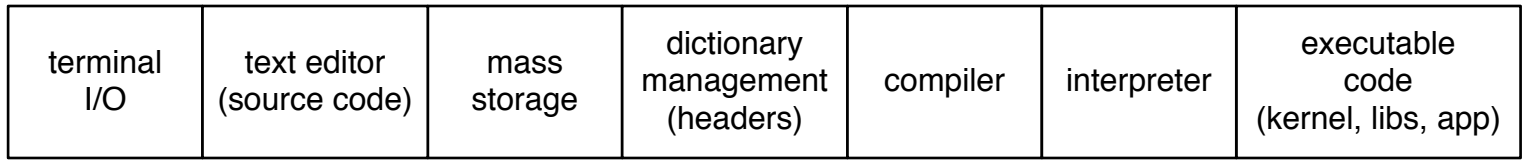
Einteilung von Host und Target

- Trennung von Entwicklungs- und Zielsystem
- Host / Target
- Welche Spielarten gibt es dabei?

interaktiver Host

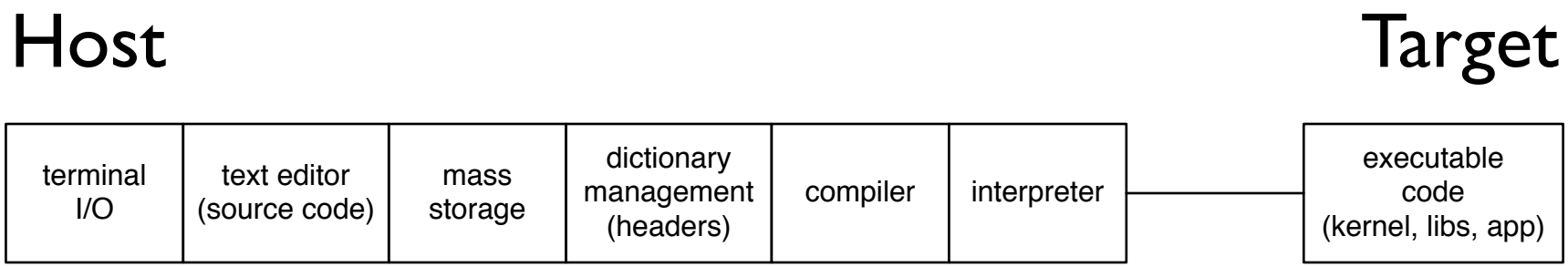
Host

Target



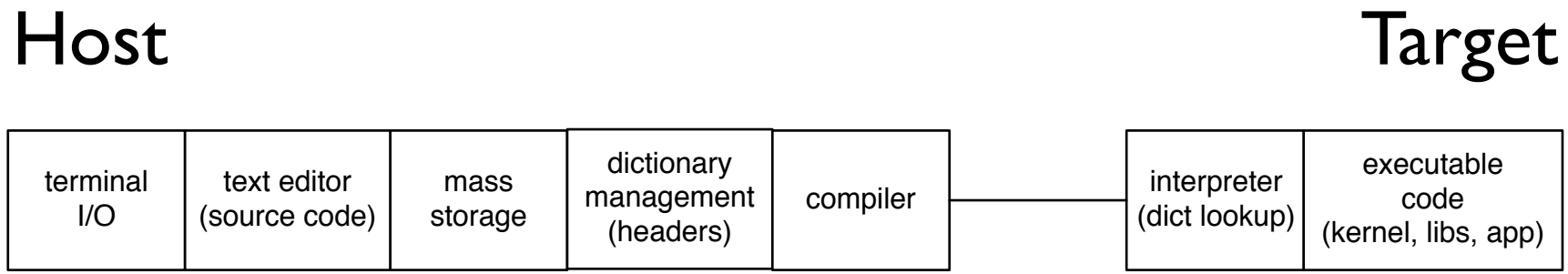
- Gesamtes System auf dem Host
- Typisches PC/Workstation Forth-System
- Desktop: SwiftForth, VFX, GForth, ...

interaktives umbilical target



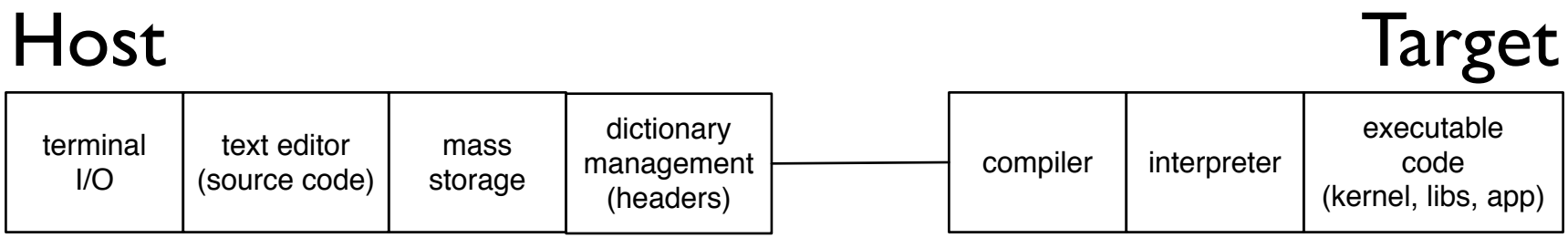
- Cross-Entwicklungsumgebung
- Host steuert Target
- Swift-X, VFX-Cross-Compiler, downCompiler
- 3-instruction-Forth

umbilical target mit interaktiver Kommandosprache



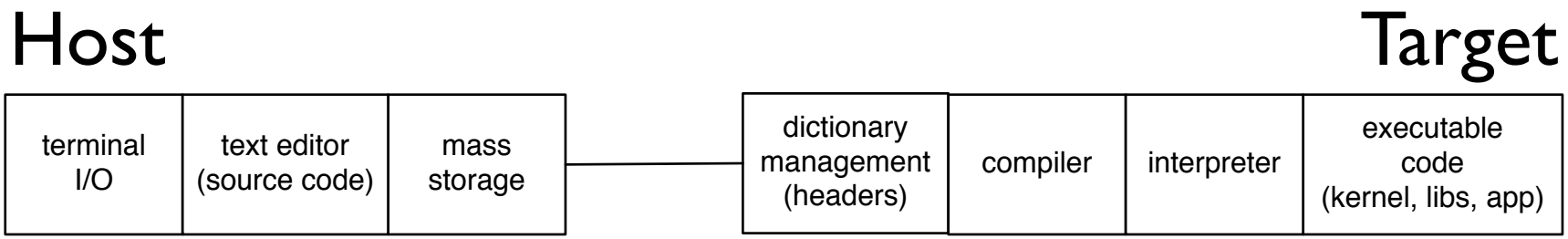
- Mit Cross-Compiler entwickelte Anwendung mit Text-Kommando-Schnittstelle
- Messsysteme mit serieller Schnittstelle zu Parametriesierung (z.B. "BitScope")

T4 artiges, interaktives Target



- Compiler und Interpreter auf dem Target
- Target steuert Host
- Beauftragt Host z.B. fürs Lesen des Eingabestroms, Dictionary-Suchen und Zahlenumwandlungen

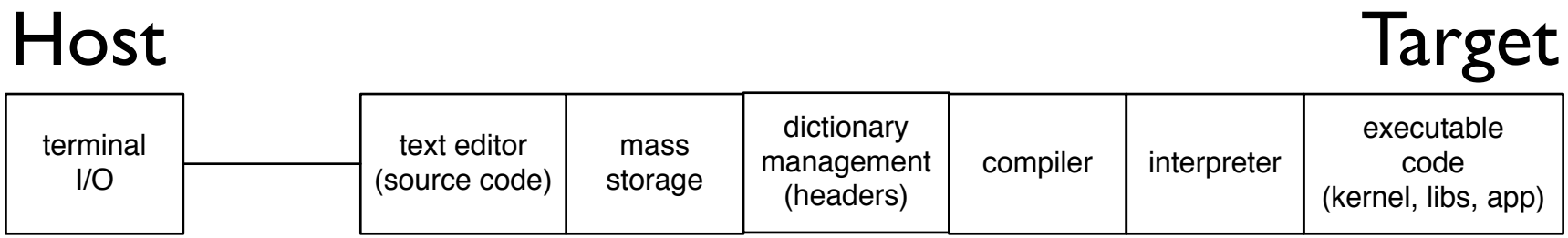
interaktive Target mit smartem Terminal



- Compiler, Interpreter, Dictionary auf dem Target
- eForth auf Controllern, 4eForth, noForth, mecrisp
- Source-code upload
- picocom, e4thcom, python-uploader, ...

- Block-Interface auf dem Target LOAD
  BLOCKs auf dem Host

eigenständiges interaktives Target mit normalem Terminal

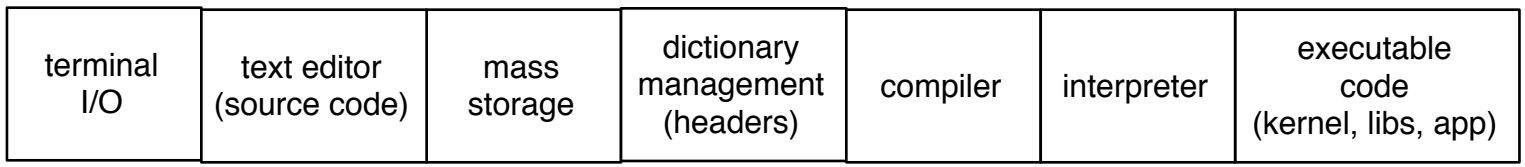


- Block-Interface auf dem Target LOAD und BLOCKs auf dem Target
- Source code auf dem Target, Block Editor
- mecrisp mit Blocks im Flash (tm4c1294)

eigenständiges interaktives Target mit eigenem Terminal

Host

Target



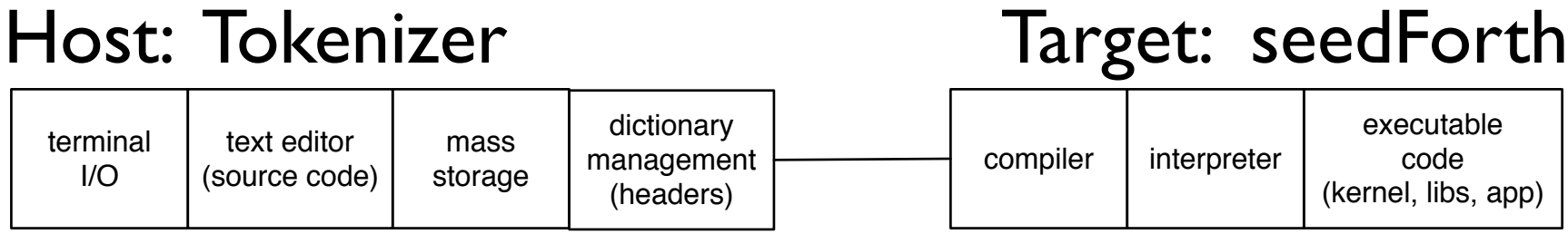
- Target hat eigenes Display und Eingabemöglichkeit
- BDET (Bring Deine Eigene Tastatur)
- Target ist Entwicklungssystem

Die Geburt von seedForth

seedForth

- eliminiert die Probleme mit "doppelt" definitierten Strukturen
- vereinfacht die Basis noch weiter
- sehr kleines (potentiell) interaktives Forth-System
- 460 LOC
- hat ein mit :-Definitionen erweiterbares Dictionary
- kann zu einem vollständigen, interaktivem Forth erweitert werden
- *akzeptiert Source-Code in Byte-Token-Form*
- seedForth für i386 und AMD64

Host/Target-Aufteilung bei seedForth



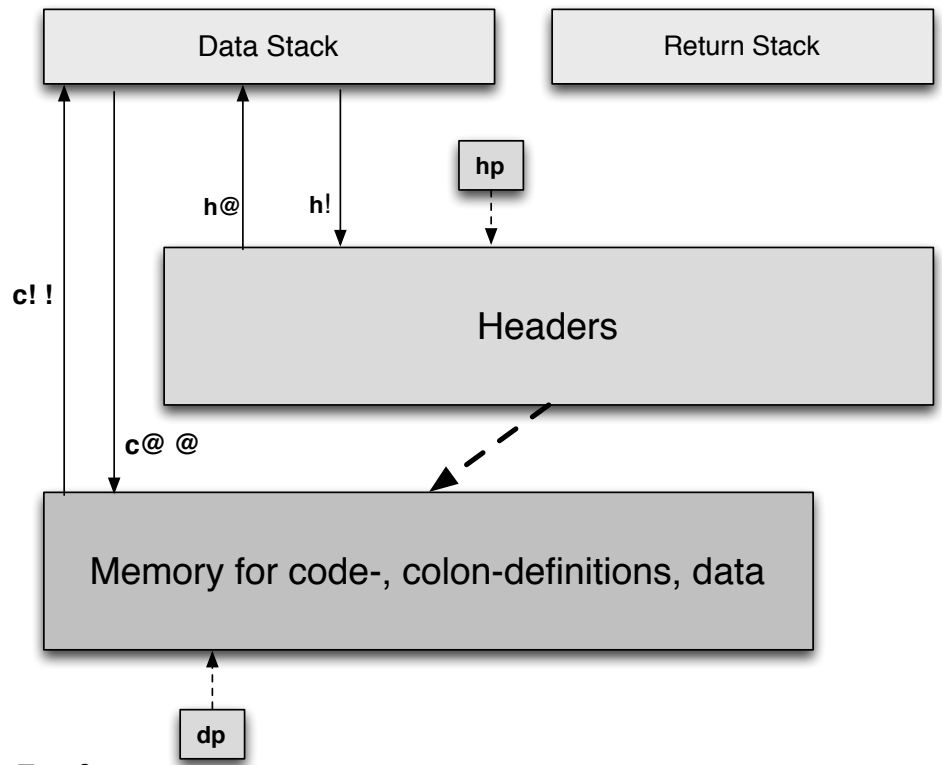
- Tokenizer (Schreibhilfe)
  - wandelt Source-Code in Byte-Token-Source-Code um
  - läuft zeitlich von seedForth getrennt

- seedForth
  - akzeptiert Byte-Token-Source-Code und verarbeitet ihn

Die Geburt von seedForth

einfachste Namen:

Namen sind nur Zahlen



seedForth virtuelle Maschine

- Datenstack, Returnstack
- Dictionary
  adressierbarer Speicher für Code, :-Definitionen, Daten
- Header
  Array, das Wort-Indizes auf Anfangsadressen abbildet

seedForth words

```
$00 #FUN: bye        $01 #FUN: emit         $02 #FUN: key        $03 #FUN: dup
$04 #FUN: swap       $05 #FUN: drop         $06 #FUN: 0<         $07 #FUN: ?exit
$08 #FUN: >r         $09 #FUN: r>           $0A #FUN: -          $0B #FUN: unnest
$0C #FUN: lit        $0D #FUN: @            $0E #FUN: c@         $0F #FUN: !
$10 #FUN: c!         $11 #FUN: execute      $12 #FUN: branch     $13 #FUN: ?branch
$14 #FUN: negate     $15 #FUN: +            $16 #FUN: 0=         $17 #FUN: ?dup
$18 #FUN: cells      $19 #FUN: +!           $1A #FUN: h@         $1B #FUN: h,
$1C #FUN: here       $1D #FUN: allot        $1E #FUN: ,          $1F #FUN: c,
$20 #FUN: fun        $21 #FUN: interpreter  $22 #FUN: compiler   $23 #FUN: create
$24 #FUN: does>      $25 #FUN: cold         $26 #FUN: depth      $27 #FUN: compile,
$28 #FUN: new        $29 #FUN: couple       $2A #FUN: and        $2B #FUN: or
$2C #FUN: catch      $2D #FUN: throw        $2E #FUN: sp@        $2F #FUN: sp!
$30 #FUN: rp@        $31 #FUN: rp!          $32 #FUN: $lit
```

```
: interpreter ( -- )
   key execute   tail interpreter ;
```

```
: compiler ( -- )
   key ?dup 0= ?exit compile, tail compiler ;
```

seedForth Tokenizer

- wandelt Menschen-lesbaren Source-Code in Byte-Token-Source-Code um (*"Texteditor-Aufgabe"*)
- about 100 LOC

demo.seedsource

```
program demo.seed

'H' # emit  'e' # emit  'l' # dup emit emit  'o' # emit  10 # emit

': 1+ ( x1 -- x2 ) 1 #, + ;'

'A' #  1+  emit   \ outputs B

end
```

demo.seed

```
00000000  02 48 01 02 65 01 02 6c  03 01 01 02 6f 01 02 0a  |.H..e..l....o...|
00000010  01 20 0c 00 02 01 1e 22  15 0b 00 02 41 33 01 00  |. ....."....A3..|
```

seedForth Tokenizer

- wandelt Menschen-lesbaren Source-Code in Byte-Token-Source-Code um (*"Texteditor-Aufgabe"*)
- about 100 LOC

demo.seedsource

```
program demo.seed

'H' # emit  'e' # emit  'l' # dup emit emit  'o' # emit  10 # emit

': 1+ ( x1 -- x2 ) 1 #, + ;'

'A' #  1+  emit   \ outputs B

end
```

demo.seed

```
00000000  02 48 01 02 65 01 02 6                    ...o...|
00000010  01 20 0c 00 02 01 1e 2                    ...A3..|
```

```
$ cat demo.seed | bin/seedForth
seed
Hello
B
```

seedForth wächst

Geplante Erweiterungen hin zu einem vollständigen, interaktivem Forth-System

- ✓ dynamische Speicherverwaltung mit allocate, resize und free
- ✓ definierende Worte incl. DOES>
- Tokenizer und preForth ausgedrückt in seedForth, sodass es eigenständig wird.
- Header mit Dictionary-Suche und DUAL-Unterstützung
- Text-Interpreter und Compiler für nicht-Token Source-Code Handler based ggf. mit string Descriptors und Regulären Ausdrücken oder Recognizer
- kompilierende Worte
- ein Forth-Assembler für die Ziel-Plattform und zusätzliche Primitives
- Multitasking
- OOP
- File und Betriebssystem-Schnittstelle
- Hardware-Zugriff

Zusammenfassung

Forth: die neue Synthese

preForth

- bootstrap-fähiges, selbst-generierendes System
- vollständig transparent
- einfach zu verstehen

seedForth

- byte tokenisierter Source-Code
- anfangs sind Wortnamen nur Zahlen-Indizes in das Header-Array
- erweiterbar zu einem vollständigen, interaktivem Forth
- einfach zu verstehen

Ausblick

-

Zusammenfassung

Forth: die neue Synthese

preForth

- bootstrap-fähiges, selbst-generierendes System
- vollständig transparent
- einfach zu verstehen

seedForth

- byte tokenisierter Source-Code
- anfangs sind Wortnamen nur Zahlen-Indizes in das Header-Array
- erweiterbar zu einem vollständigen, interaktivem Forth
- einfach zu verstehen

Ausblick ✔ ✔

-

Forth: ***words - stacks - blocks***

Jeff Fox

Zusammenfassung

Forth: die neue Synthese

preForth
- bootstrap-fähiges, selbst-generierendes System
- vollständig transparent
- einfach zu verstehen

seedForth
- byte tokenisierter Source-Code
- anfangs sind Wortnamen nur Zahlen-Indizes in das Header-Array
- erweiterbar zu einem vollständigen, interaktivem Forth
- einfach zu verstehen

Ausblick ✔ ✔
- Forth: ***words - stacks - blocks***
  Jeff Fox

projektionales Editieren

Block-Editor
Hex-Editor
Text-Editor
Formular-Editor
...