Der Stack der Stacks

Ulrich Hoffmann <uho@xlerb.de>

Überblick

- Stacks, Stacks, Stacks - was ist ein Stack-Element?
- der Stacks als
  - Stack
  - Queue
  - Liste
  - String
- Stack der Stacks
- Einsatzbeispiele
  - formatierte Zahlenausgabe
  - Eingabe vom Terminal
  - Listen
- Fazit

Stacks, Stacks, Stacks

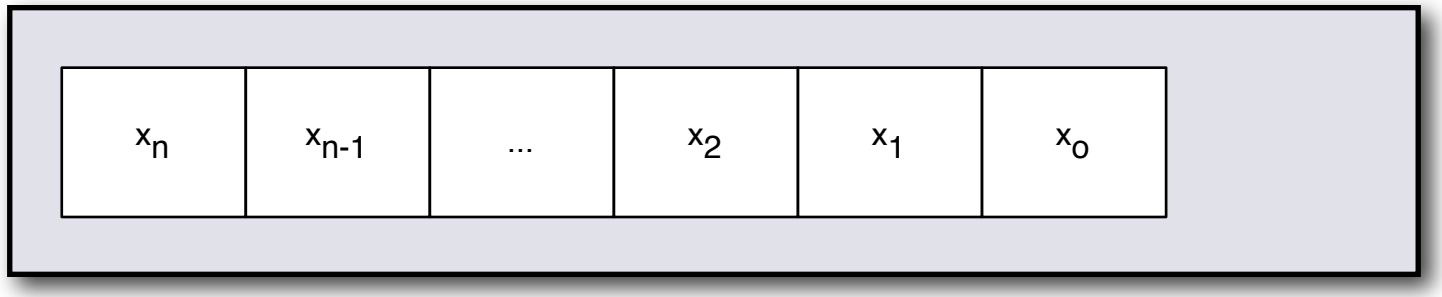
> Forth ist ***words - stacks - blocks***
>
> *Jeff Fox*

- Datenstack
- Returnstack
- Vokabularstack
  `only also` / `get-order set-order`
- Recognizer-Stack
- User-Stacks

Was ist ein Stack?

- **LiFo** - last in first out
- einfache koordinierte Speicherverwaltung
- `push, pop, top - LIT DROP`
- realisiert
  - im Hauptspeicher
  - als eigene Register
- Größe
  - potentiell beliebig
  - in vielen Umgebungen knapp

Was ist ein Stack?



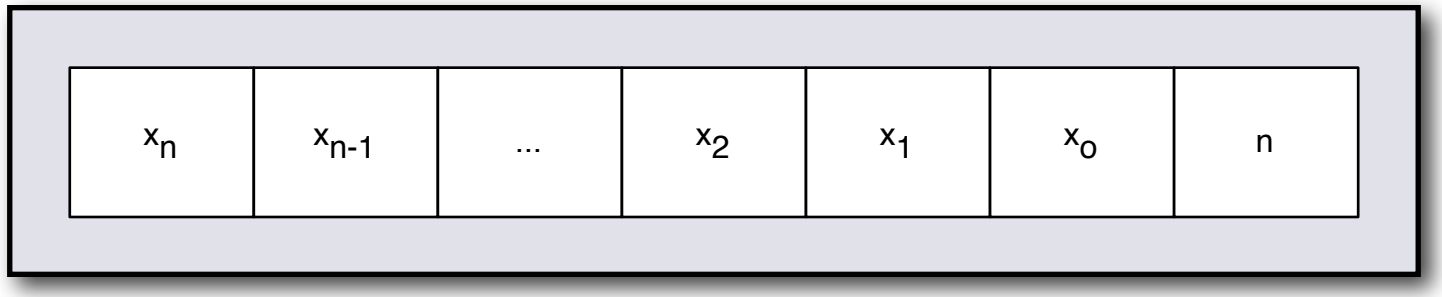
- Stack-Operatoren

  `dup swap drop over rot -rot pick roll` ...

  `>r r> r@` ...

  `! @` ...

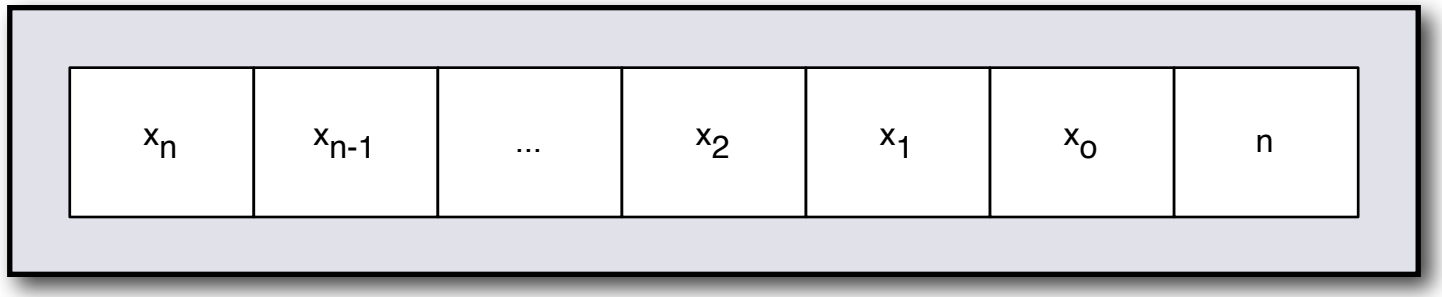
Der Stack als Stack



- für einen so dargestellten Stack:
  Prefix `*` für alle Operatoren.

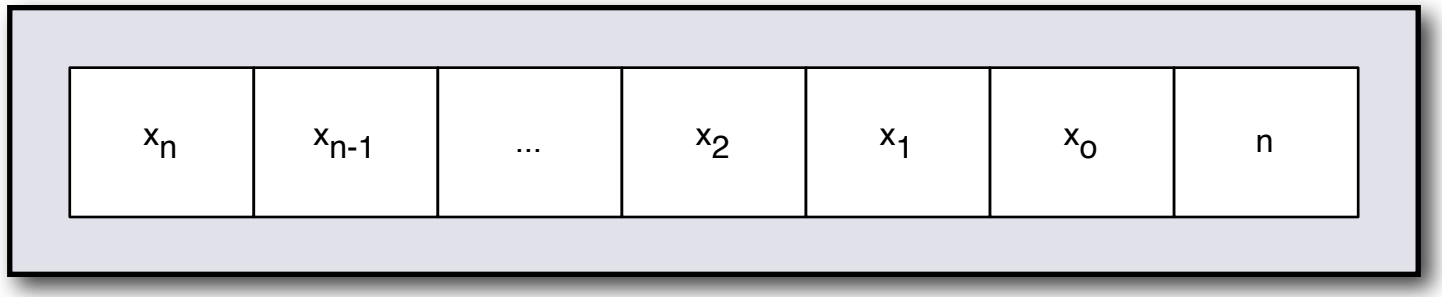
```
*dup *swap *drop *over *rot
*-rot *pick *roll ...

*>r *r> *r@ ...

*! *@ ...
```

Der Stack als Stack



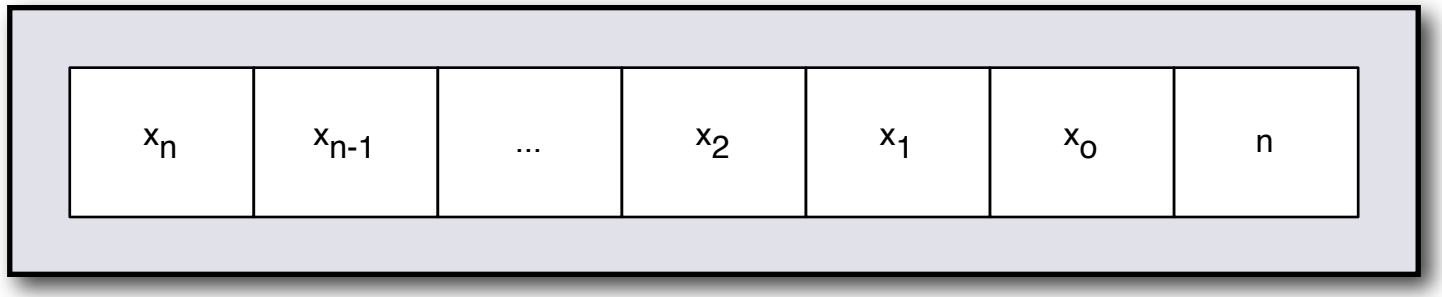

```
: *depth ( S -- S n )  dup ;

: *elt ( S n -- S x )  1 + pick ;

: *top ( S -- S x )  over ;

: *push ( S1 x -- S2 )  swap 1 + ;

: *pop ( S1 -- S2 x )  1 - swap ;
```

Der Stack als Stack




```
: *pick ( S1 n -- S2 )  *elt *push ;

: *roll ( S1 n -- S2 )  1 + roll swap ;
```

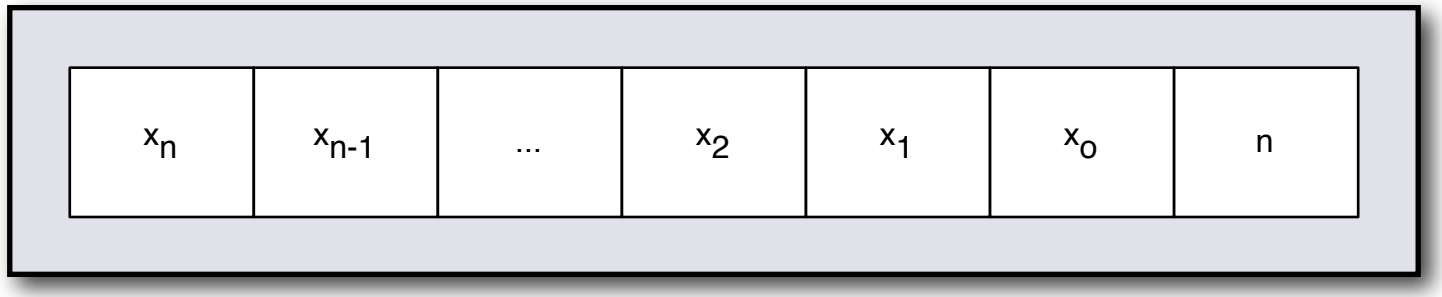
Der Stack als Stack




```
: *dup ( S1 -- S2 )  *top *push ;

: *over ( S1 -- S2 )  2 pick *push  ;

: *swap ( S1 -- S2 ) >r swap r> ;

: *rot ( S1 -- S2 ) >r rot r> ;

: *drop ( S1 -- S2 )  nip 1 - ;
```

```
: ()drop ( S -- ) 0 ?DO drop LOOP ;
```

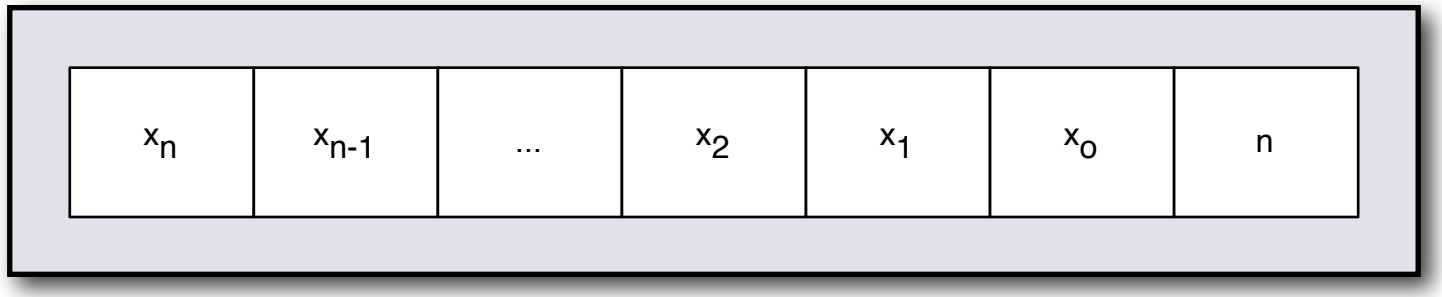
Der Stack als Stack




```
: *! ( S addr -- ) 2dup ! cell+
    BEGIN ( sn-1 ... s0 n addr )
      over
    WHILE ( sn-1 ... s0 n addr )
      rot over !  cell+ swap 1- swap
    REPEAT 2drop ;
```

`*@ *>r *r>` entsprechend

Der Stack als Queue

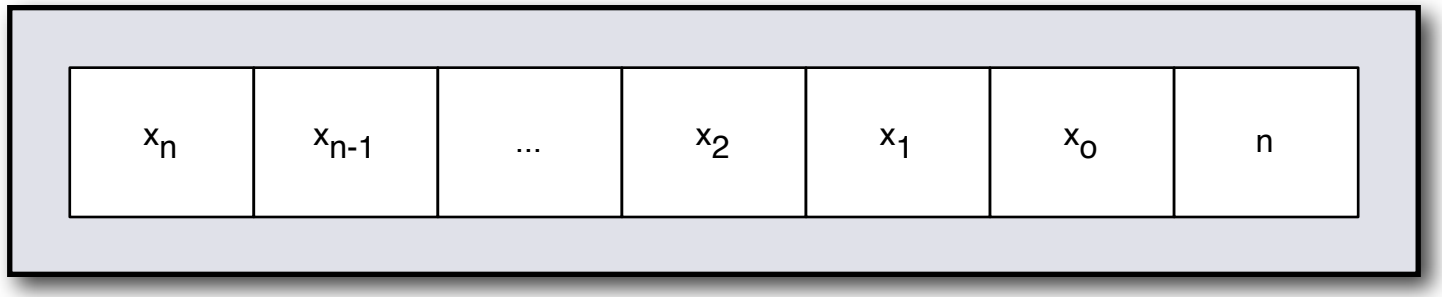


- **FiFo -** first in first out
- Daten "am Boden" des Stacks einfügen

```
: *pop ( S1 -- S2 x )  1 - swap ;

: *prepend ( S1 x -- S2 )  swap dup >r -roll r> 1 + ;

: *detract ( S1 -- S2 x )  *depth dup IF 1 - *roll *pop THEN ;
```

```
t{ 20 30 40 3 10 *prepend -> 10 20 30 40 4 }t
t{ 0 *detract -> 0 0 }t
t{ 10 20 30 3 *detract -> 20 30 2  10 }t
```

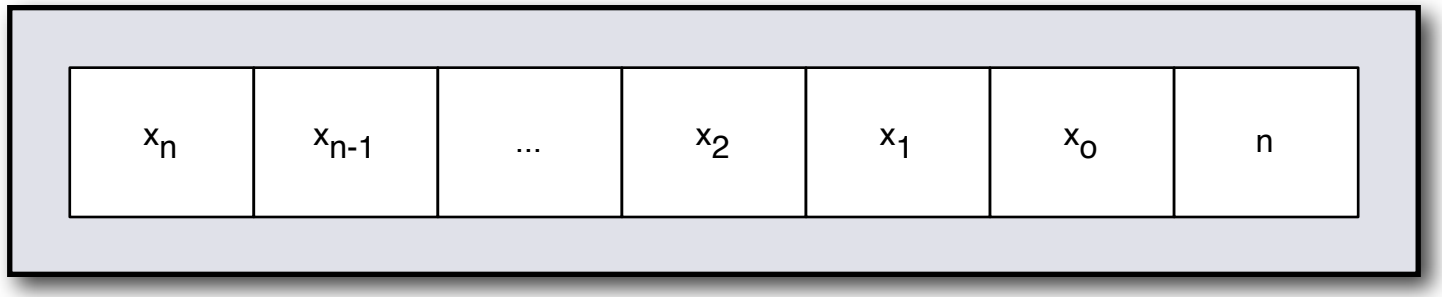
Der Stack als Liste




```
: *map ( S1 xt -- S2 )   \ xt has effect  ( x1 -- x2 )
   over 0 ?DO  ( s ... s n xt )
      >r dup roll r@ execute swap r>
   LOOP drop ;
```

```
: dup. ( x -- x ) dup . ;

: *.s ( S -- )  ['] dup. *map ;
```

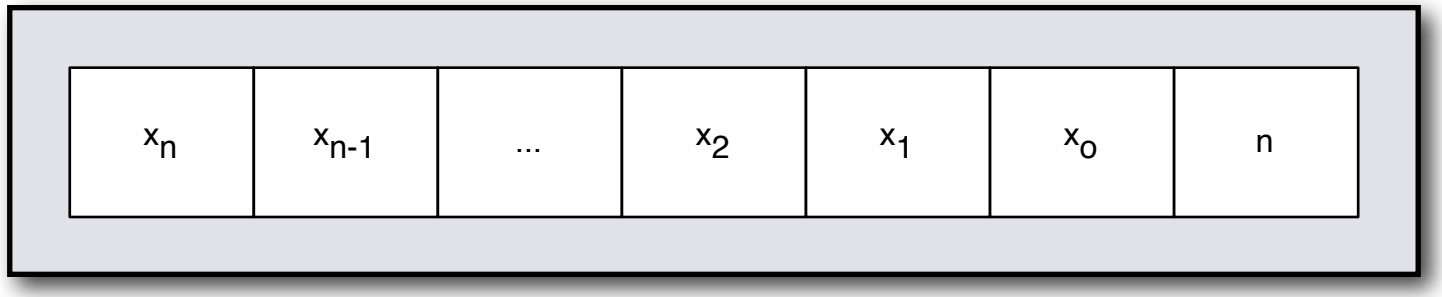
Der Stack als Liste




```
: *reduce ( S x1 xt -- x2 )
    \ xt has stack effect ( x1 x2 -- x3 )
    2 pick  0 ?DO  ( s ... s n x xt )
       >r >r  *detract  r> r@ execute r>
    LOOP drop nip ;
```

```
t{ 10 20 30 3  0 ' + *reduce -> 60 }t
```

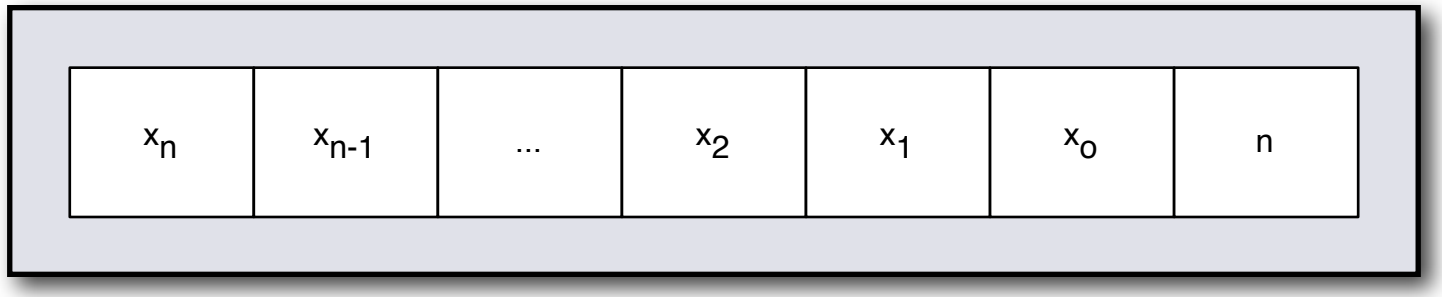
Der Stack als Liste




```
: *filter ( S1 xt -- S2 )
    \ xt hast stack effect ( x -- f )
    over 0 ?DO >r
      *detract  dup r@ execute
      IF  *push ELSE drop THEN
    r> LOOP drop ;
```

```
t{ 10 -20 30 -40 4  ' 0< *filter ->  -20 -40 2 }t
```

Der Stack als String

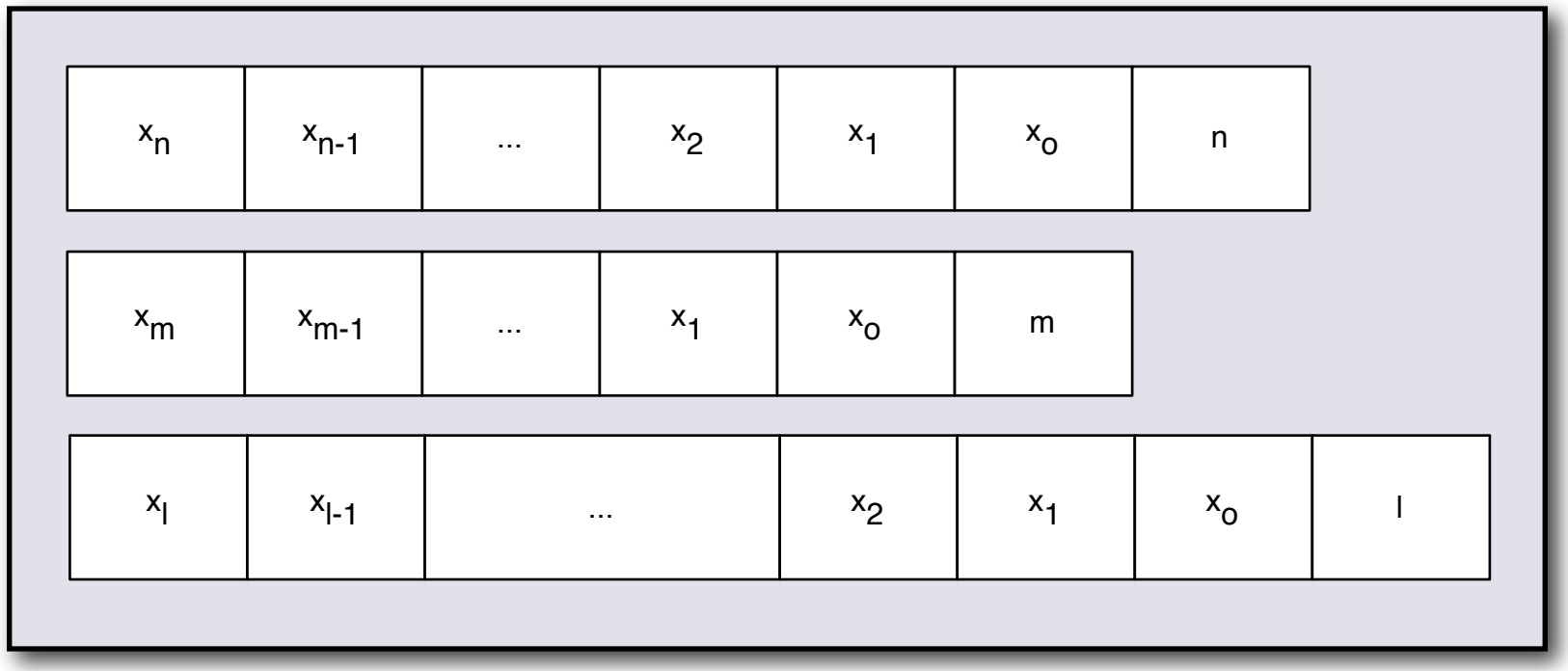



```
: spell ( addr len -- S )
   dup >r bounds ?DO  I c@  LOOP r> ;
```

```
t{ s" abc" spell ->  char a char b char c 3 }t
```

```
: *show ( S -- S )  ['] dupemit *+map ;
: *type ( S -- )  *show ()drop ;
```

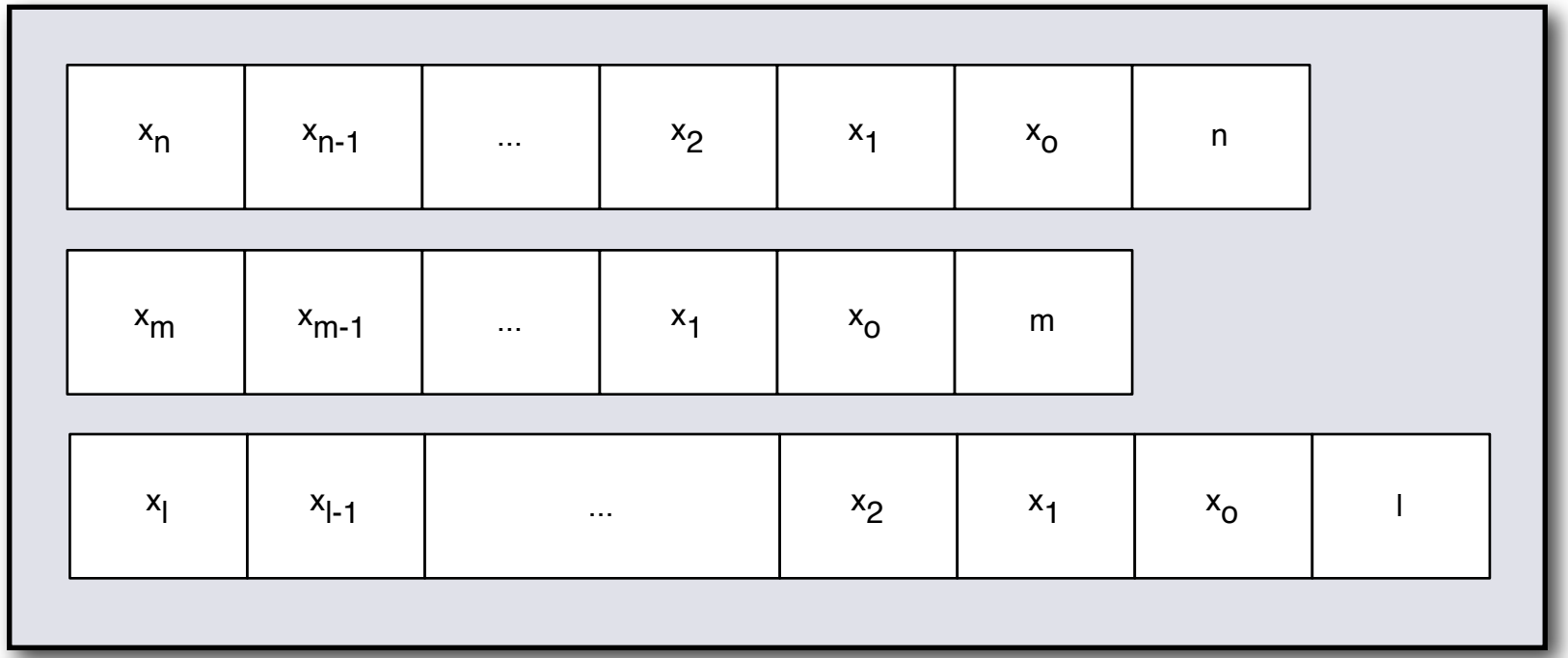
Stack der Stacks



- Stack-Operatoren für die komplexen Stack-Elemente
- Prefix `()` für alle Operatoren.

```
()dup ()swap ()drop ()over ()rot
()join ...
```

Stack der Stacks




```
: ()drop ( S -- ) 0 ?DO drop LOOP ;

: ()dup ( S -- S S ) *depth dup 0 ?DO  dup >r pick r> LOOP ;

: ()swap ( S1 S2 -- S2 S2 )
    **depth 1+ >r *depth r@ + r> 0 ?DO dup >r roll r> LOOP drop ;

: ()join ( S1 S2 -- S3 )  *depth 1 + roll + ;

: ()over ( S1 S2 -- S1 S2 S1 )  ()>r ()dup ()r> ()swap ;

: ()rot ( S1 S2 S3 -- S2 S3 S1 )  ()>r ()swap  ()r> ()swap ;
```

Einsatzbeispiele

- Wo kann der Stack der Stacks eingesetzt werden, wo sonst Puffer verwendet werden?

formatierte Zahlenausgabe

```
: hold ( S1 n c -- S2 ) swap >r *prepend r> ;
: sign ( S1 n n -- S2 n )  0< IF  [char] - hold THEN ;

: <#  ( n  -- S n )  0 swap ;
: #> ( S1 n1 -- S2 ) drop ;

: # ( S1 n1 -- S2 n2 )  s>d  base @ um/mod swap
    dup 9 > IF 10 - [char] A ELSE [char] 0 THEN + hold ;

: #s ( S1 n1 -- S2 n2 )  BEGIN # dup 0= UNTIL ;

: *. ( n -- )  <#  bl hold  dup >r  abs #s r> sign  #> *type ;
```

```
t{ 124 <# #S #>  ->  char 1 char 2 char 4  3 }t
```

Eingabe vom Terminal

```
: query ( -- S )
   0 BEGIN key
        dup 13 xor over 10 xor and
     WHILE ( c )
        dup 8 = over 127 = or IF drop 8 emit space 8 emit *drop ELSE
        dup emit *push THEN
     REPEAT
   drop ;
```

demo

Listen

```
: range ( to from -- S ) 0 -rot ?DO  I *push LOOP ;

: fac ( n -- )  1 + 1 range  1 ['] * *reduce ;
```

```
t{ 6 fac -> 720 }t
```

Fazit

- Der Stack der Stack ist ja schon längst da:
  - get-order set-order n>r nr>
  - der Stack ist nützlich, für mehr als gedacht
  - Effizienz im Auge behalten
  - prima Erweiterung des Werkzeugkastens

Forth ist ***words - stacks - blocks***

Jeff Fox