# Managed Forth

M. Anton Ertl, TU Wien

# Fehler durch falsche Speicherzugriffe

- Arrayzugriff über die Grenzen hinaus $\rightarrow$ bounds checking
- Falsche Adresse bei Feldzugriff $\rightarrow$ Objekttyp prüfen
- Zugriff auf deallokiertes Objekt $\rightarrow$
  keine manuelle Deallokation
  oder keine Adressen von deallokierten Objekten

## Existierende Implementierungsstrategien

- Objektdeskriptor im Objekt (fast alle)
- statische Typüberprüfung (Factor, Java, C#)
- Tags auf allen Daten (Oforth, Lisp, Python)
- Garbage Collection (die meisten) oder Reference Counting (Python) oder statische Lebenszeitanalyse (Rust)

## Managed Forth

- keine statische Typüberprüfung
- keine Tags

# Wie?

- keine Adressen oder xts auf Datenstack, kein `@`, `!`
- Daher keine `Variablen`
  Stattdessen `Values`
  entsprechend für Felder
- Objektstack enthält Adressen von Objekten
- `Ovalues` enthalten Objektadressen $\leftrightarrow$ Objektstack
  entsprechend für Felder
- Feldzugriff, Arrayzugriff überprüft Objekttyp
- für xts `defer`-artige Wörter, Methoden, oder einen xt-Stack
- Stacküberlauf/unterlauf abfangen
  Stack cache bei Garbage collection berücksichtigen

# Wie? Deallokation

- keine explizite Deallokation (GC oder reference counting)
- oder Regionen mit Reihenfolge der Deallokation
  Adressen von kürzerlebende Objekte nicht in längerlebenden Objekten
  Globale `ovalues` und Objektstack bei Deallokation überprüfen

# Beispiel: einfache Struktur

```
begin-structure intlist
  ovalue: intlist-next
  value:  intlist-val
end-structure

: list-sum ( list -- n )
  0 begin
    odup null o<> while
    odup intlist-val +
    intlist-next
  repeat
  odrop ;
```

# Beispiel: objektorientiert

```
class intlist
  ovalue: next
  value:  val
  m: sum1 ( n1 intlist -- n2 )
    val + next sum1 ;m
end-class

class emptylist
  m: sum1 ( n1 emptylist -- n1 )
  ;m
end-class

: sum ( collection -- n )
  0 sum1 ;
```

# Beispiel: Array

```
: array-sum ( array -- n )
  0 odup array-len 0 ?do ( array n1 )
    odup i [] +
  loop
  odrop ;
```

# Geschwindigkeit

- ohne Speicherzugriff gleiche Geschwindigkeit
- bei Feldzugriff Overhead für Typüberprüfung
  alternativ: Zugriff auf Felder von `this` (kein Overhead)
- bei Arrayzugriff Overhead für Typ- und Grenzüberprüfung
  alternativ: Array-Iterations-Wörter
- Garbage collection
  nur Objektvariablen, -felder, und -stack prüfen
  evtl. präzise statt konservativ
  evtl. kopierend

# Beispiel: Effizientere Array-Iteration

```
: array-sum ( array -- n )
  0 odup array-len 0 ado ( n1 n2 )
    +
  aloop ;
```

# Unmanaged Code

- Aus dem goldenen Käfig ausbrechen
  Programmierer verantwortet Lücken
- Wordlist `managed` am Anfang (in normalem Forth) erweitern
- Ausstiegsluke in normales Forth (z.B. `forth`)
  wird man üblicherweise zum Erweitern des Managed Forth verwenden
  bei Bedarf mit `seal` zuschweißen

# Zusammenfassung

- Fehler durch falsche Speicherzugriffe eliminieren
- keine Tags und keine statische Typprüfung
- Objektstack, `values`, Prüfung bei Feldzugriff, garbage collection
- Varianten möglich
- Overhead bei Feld- und Arrayzugriff
  kann reduziert werden
- Ausstiegsluke
- derzeit nur ein Konzept