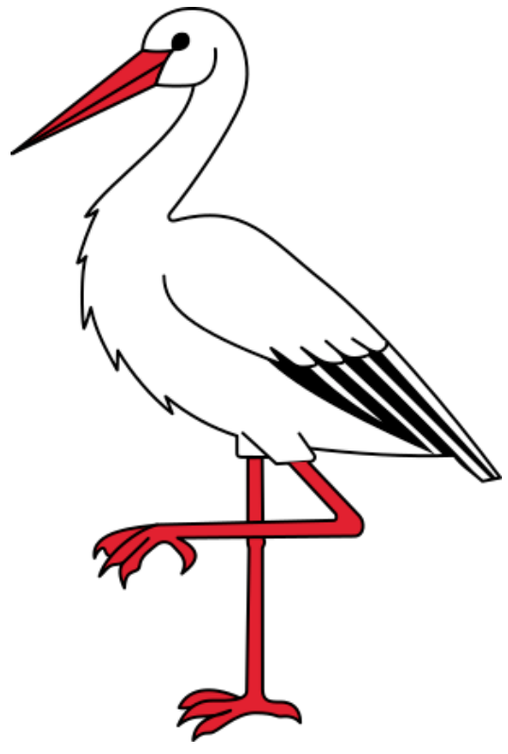
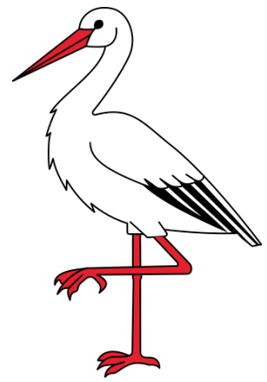
Reflecting on EIAS



Christian Gollwitzer

EuroTcl 2016

Reflection (on) EIAS

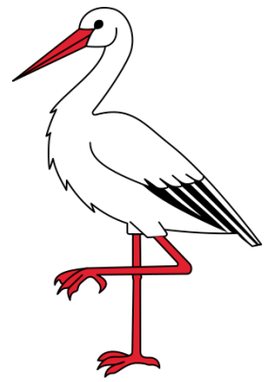


- Everything Is A String in Tcl < 8.0
- Everything is Indistinguishable from A String (Tcl ≥ 8.0) internally: Tcl_Obj, Tcl_ObjType
- TclValue is an extension, that lets you write your own Tcl_ObjType from within Tcl

Quick survey: how many of you...

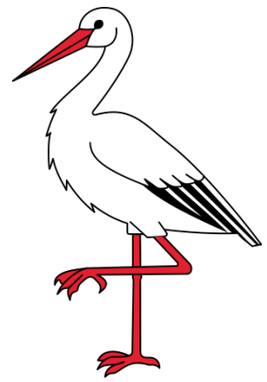
- ... have written extensions with the C API of Tcl?
- ... know what a Tcl_ObjType is?
- ... have implemented their own Tcl_ObjType?

Recapitulation on EIAS vs. EIfAS



Code	Tcl 7.6
`1 set a 3`	*put* 1-char string „3“ → a
`2 set b 4`	*put* 1-char string „4“ → b
`3 set c [expr {$a+$b}]`	*convert* string „3“ to bitpattern 0011 *convert* string „4“ to bitpattern 0100 *allocate memory* for the result *execute* addition → 0111 *convert* 0111 to string „7“ → c
`4 puts $c`	*print* string „7“ → stdout

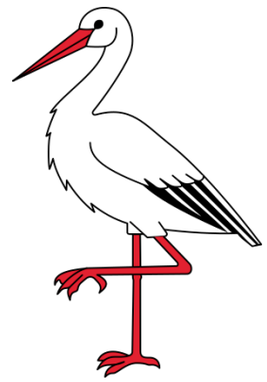
Recapitulation on EIAS vs. EIfAS



Code	Tcl 7.6
`1 set a 3`	*put* 1-char string „3“ → a
`2 set b 4`	*put* 1-char string „4“ → b
`3 set c [expr {$a+$b}]`	*convert* string „3“ to bitpattern 0011 *convert* string „4“ to bitpattern 0100 *allocate memory* for the result *execute* addition → 0111 *convert* 0111 to string „7“ → c
`4 puts $c`	*print* string „7“ → stdout

- actual work is minor - single machine instruction (~1 ns)
- Tcl spends most of the time *shimmering* (converting data types)
- ~100-1000× slower then equivalent C code (~1 μs)

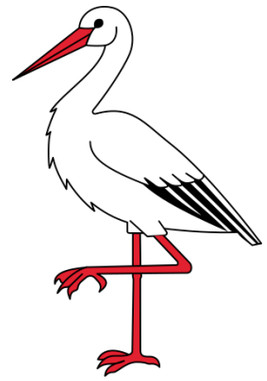
Recapitulation on EIAS vs. EIfAS



Code	Tcl 8
`1 set a 3`	*put* 1-char string „3“ → a
`2 set b 4`	*put* 1-char string „4“ → b
`3 set c [expr {$a+$b}]`	*convert* string „3“ to bitpattern 0011 → a *convert* string „4“ to bitpattern 0100 → b *allocate memory* for the result *execute* addition → 0111 → c
`4 puts $c`	*convert* 0111 to string „7“ → c *print* string „7“ → stdout

- Result of conversion is cached
- Pays off for repeated execution (2-3× faster, Tcl 8.6 vs. Tcl 7.6)

Recapitulation on EIAS vs. EIfAS



- Even larger speed-up for lists:

```
1 set l [lrepeat 1000 element]
2 time {set el [lindex $l 900]}
```

Tcl 8

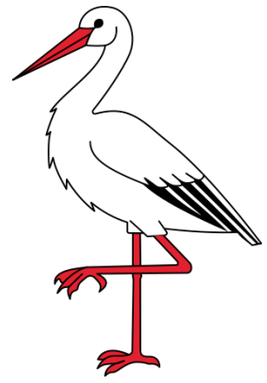
- Index 901st element in the cached list (`el=obj[900]`)
- 0.24 μs

Tcl 7.6

- Parse 900 elements, ignore them, until you arrive at 901st
- 26 μs

List access for a long list is 100× faster

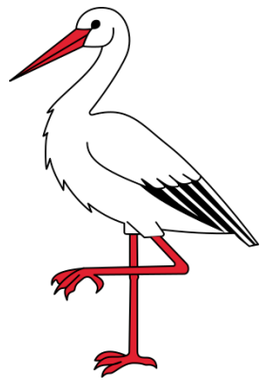
Tcl_Obj & Tcl_ObjType



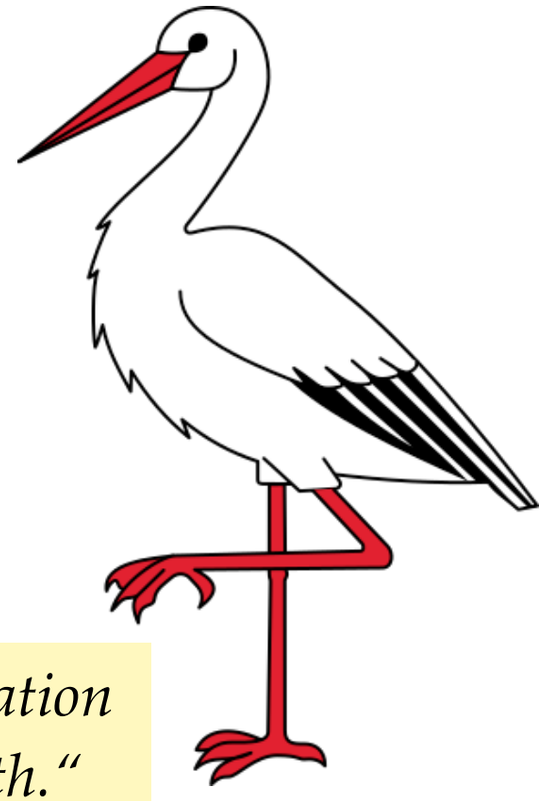
- Tcl 8.6 defines 33+ Tcl_ObjTypes
  `int, double, bignum, list, dict, ...`
- ... but also:
  `regexp, exprcode, parsedVarName, procbody, ....`
- Not noticeable from a standard Tcl script (except for speed), but can be inspected using `tcl::unsupported::representation` or a special tkcon inspector (Live demo)

EIAS today: „Everything is Indistinguishable from A String“

Tcl_Obj & Tcl_ObjType



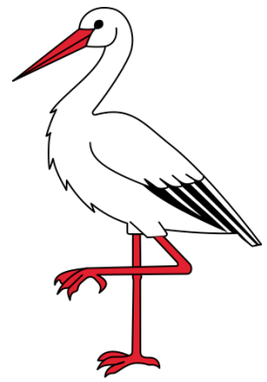
- Tcl_Objs store:
  - A string
  - An internal representation
  - One of them can be NULL
  - A Tcl_ObjType which defines what the internal representation **means**



DKF: *„Tcl_Obj's are like storks. They have two legs, the internal representation and the string representation. They can stand on either leg, or on both.“*

- Tcl_ObjTypes store procedures to deal with the IntRep
  - A „constructor“ (SetFromAnyProc)
  - A „destructor“ (FreeIntRepProc)
  - A „copy constructor“ (DupIntRepProc)
  - A „serialization function“ (UpdateStringProc)

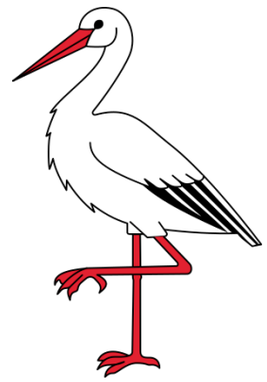
Introducing TclValue



- Extensions *can* create their own Tcl_ObjType
  by providing four C functions
- Can be very useful to speed up stuff (↗ VecTcl)
- Tcl code *by definition* cannot create new Tcl_ObjTypes,
  despite that Tcl is much easier to write than C

TclValue is an extension, which allows Tcl code to define new Tcl_ObjTypes

Ordered Sets

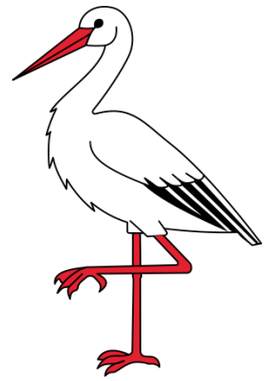


- Ordered Sets are unique lists of values, like struct::set
- Values can only occur once:
  {a b c} + {b} = {a b c}
- The keep the order of insertion
  {a c b} + {d b f} = {a c b d f}
- In pure Tcl either efficient or nice

```
dict merge {a * c * b *} {d * b * f *}
⇒ {a * c * b * d * f *}
```

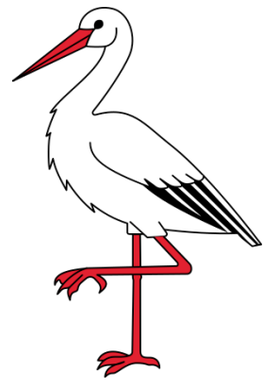
We want a dict without values, i.e.
the string rep is a list of the keys only

Ordered Sets



```
1  package require tclvalue
2
3  tclvalue::register oset {
4      variable intrep
5      constructor {list} {
6          set intrep {}
7          foreach l $list {
8              dict set intrep $l 1
9          }
10     }
11
12     method repr {} {
13         dict keys $intrep
14     }
15
16     method insert {key} {
17         dict set intrep $key 1
18     }
19 }
20
```

Ordered Sets



```
1  package require tclvalue
2
3  tclvalue::register oset {
4      variable intrep
5      constructor {list} {
6          set intrep {}
7          foreach l $list {
8              dict set intrep $l 1
9          }
10     }
11
12     method repr {} {
13         dict keys $intrep
14     }
15
16     method insert {key} {
17         dict set intrep $key 1
18     }
19 }
20
```

SetFromAnyProc

UpdateStringProc

FreeIntRepProc

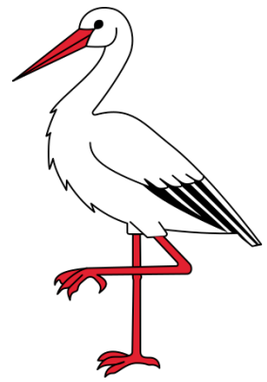
`destructor`

DupIntRepProc

`method <cloned>`

Modification function

Ordered Sets



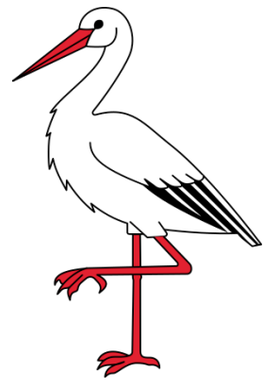
```
21 proc oset_create {args} {
22     tclvalue::new oset $args
23 }
24
25 proc oset_insert {varname value} {
26     upvar 1 $varname var
27     set oset [tclvalue::unshare var]
28     set intRep [tclvalue::shimmer $oset oset]
29     $intRep insert $value
30     tclvalue::invalidate $oset
31     set var $oset
32 }
```

Tcl_NewOsetObj()

Tcl_DuplicateObj

Tcl_ConvertToType

Ordered Sets



```
21 proc oset_create {args} {
22     tclvalue::new oset $args
23 }
24
25 proc oset_insert {varname value} {
26     upvar 1 $varname var
27     set oset [tclvalue::unshare var]
28     set intRep [tclvalue::sh                    et]
29     $intRep insert $
30     tclvalue::invalid                $oset
31     set var $oset
32 }
```

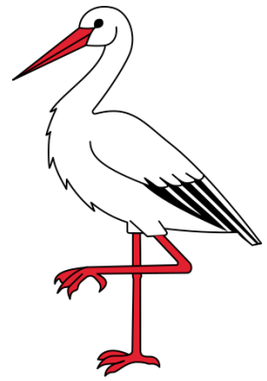
Tcl_NewOsetObj()

Tcl_DuplicateObj

Tcl_Con

100% EIAS compatible

Generator-based for loop

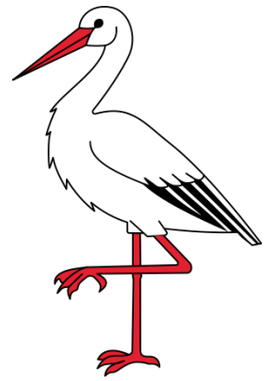


```
foreach x $o { puts $x }
```

- `foreach` expects a list and shimmers `o` to a string : (and then to list : /
- Generators yield a value each iteration
- EIAS prevents foreach to work on both lists and generators
- 95% EIAS-compatible (white magic)

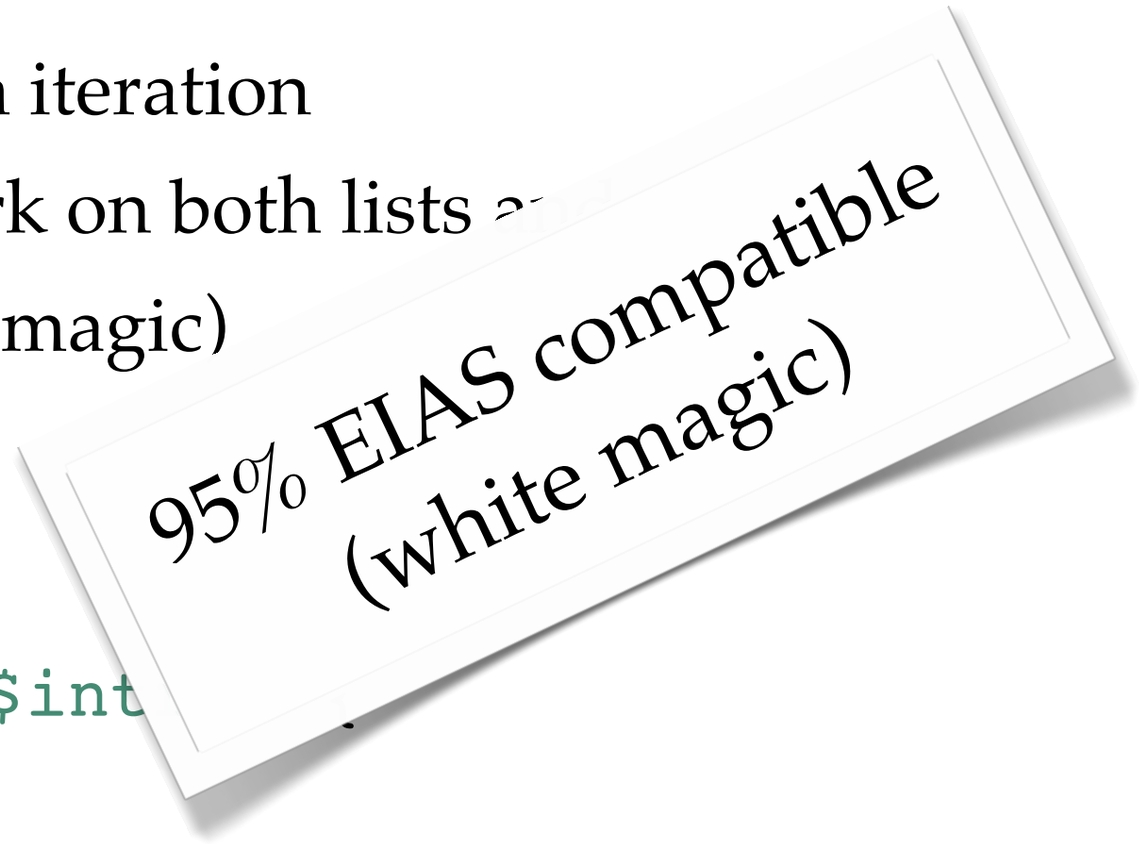
```
1 method iterate {} {
2     yield
3     dict for {value _} $intrep {
4         yield $value
5     }
6     return -code break
7 }
```

Generator-based for loop



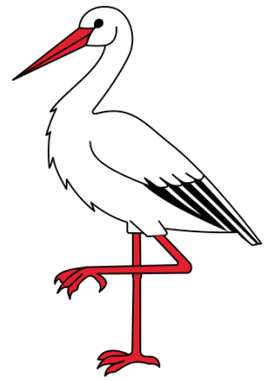
```
foreach x $o { puts $x }
```

- `foreach` expects a list and shimmers
  o to a string : (
  and then to list : /
- Generators yield a value each iteration
- EIAS prevents foreach to work on both lists a
- 95% EIAS-compatible (white magic)




```
1 method iterate {} {
2     yield
3     dict for {value _} $int
4         yield $value
5     }
6     return -code break
7 }
```

Garbage-collected SQL interface



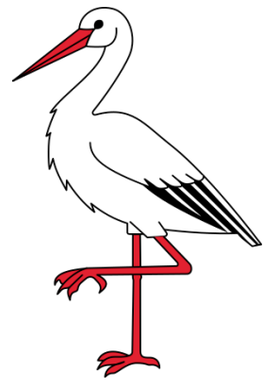
- Most natural interface returns a list of results

```
1 set db [connect some.db]
2 foreach row [query $db {SELECT * FROM table}] {
3     puts $row
4 }
5
```

- Problem: copying all results from the server to the client

```
1 set db [tdbc::sqlite3::connection new some.db]
2 set stmt [$db prepare {SELECT * FROM table}]
3 set res [$stmt execute]
4 $res foreach -as lists -- row {
5     puts $row
6 }
7 $res close
8 $stmt close
```

Garbage-collected SQL interface



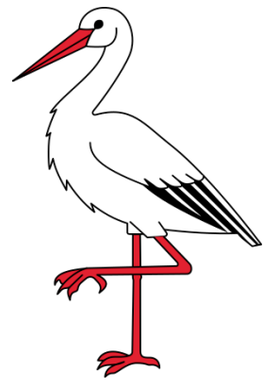
- Most natural interface returns a list of results

```
1 set db [connect some.db]
2 foreach row [query $db {SELECT * FROM table}] {
3     puts $row
4 }
5
```

- Solution: Use a Tcl_ObjType for a prepared statement
- A 2nd one with iterators as a the resultset
- Resultsets are non-repeatable

80% EIAS compatible
(black magic)

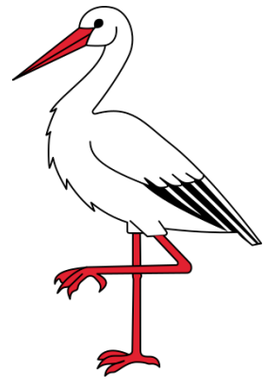
Problems & Warts in TclValue



- Uses a slave interp, because eval doesn‘t nest
- Shimmering to string is too restrictive. Want to shimmer to list, to dict, to double, ...
- Generator interface is rudimentary and slow
  - `foreach` is scripted.
  - `lindex, llength, lset,` ... missing
- TclValue will stop working in Tcl 9

> TclValue is meant as an experiment to get inspiration what is needed for Tcl 9

Conclusion



- Tcl_Obj & Tcl_ObjType were great ideas
- Extensibility is stuck:
  conversion from Foo to list shimmers twice via **string :(**
- Scriptable Tcl_ObjType allows easy experiments & cheating
  to explore the limits of the current systems
- Potential feature of future Tcl 9 ?
- Generators & garbage collection are cool features, but not
  100% EIAS compatible (black magic / white magic ?)