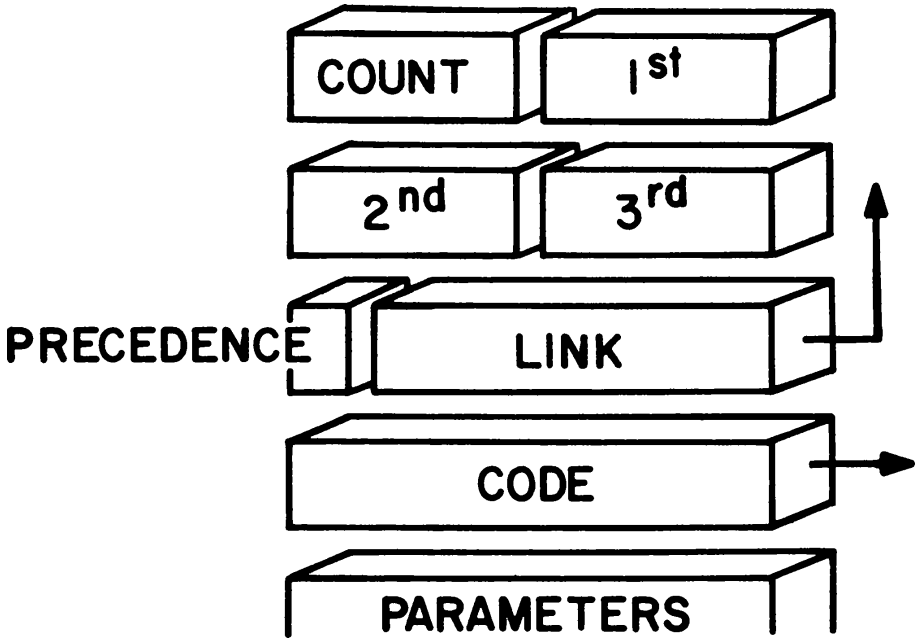
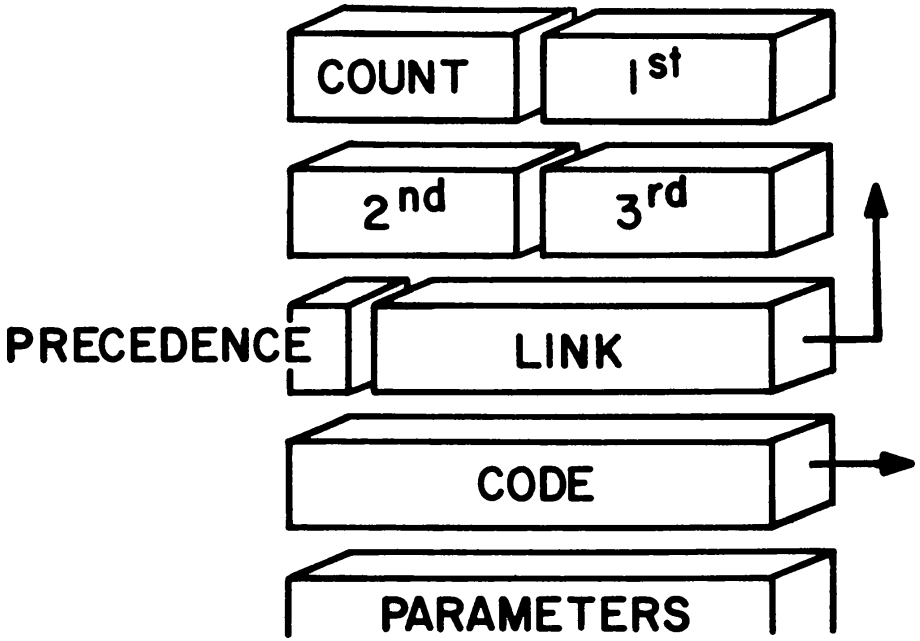
The new Gforth Header

Bernd Paysan, net2o
M. Anton Ertl, TU Wien

Traditional Header

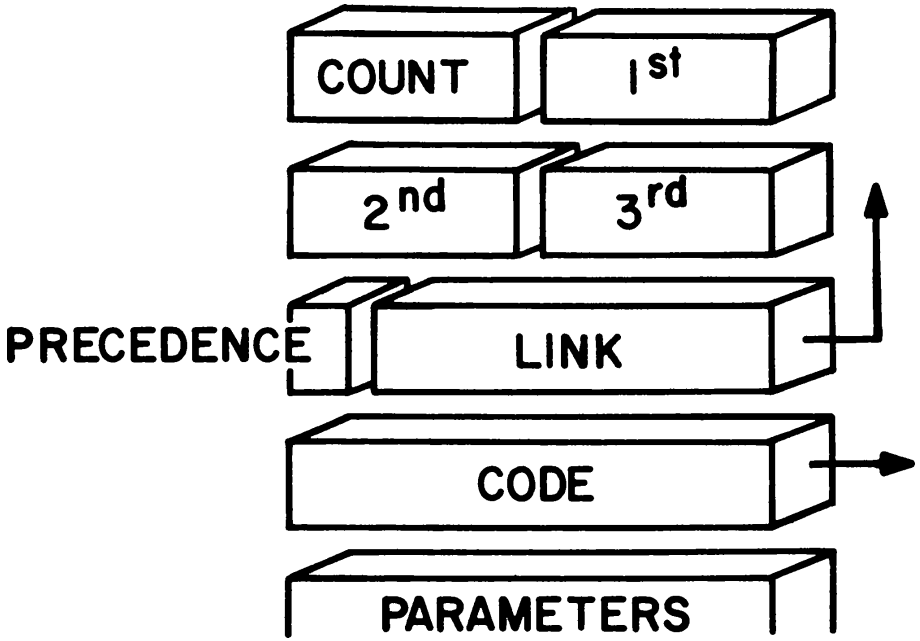


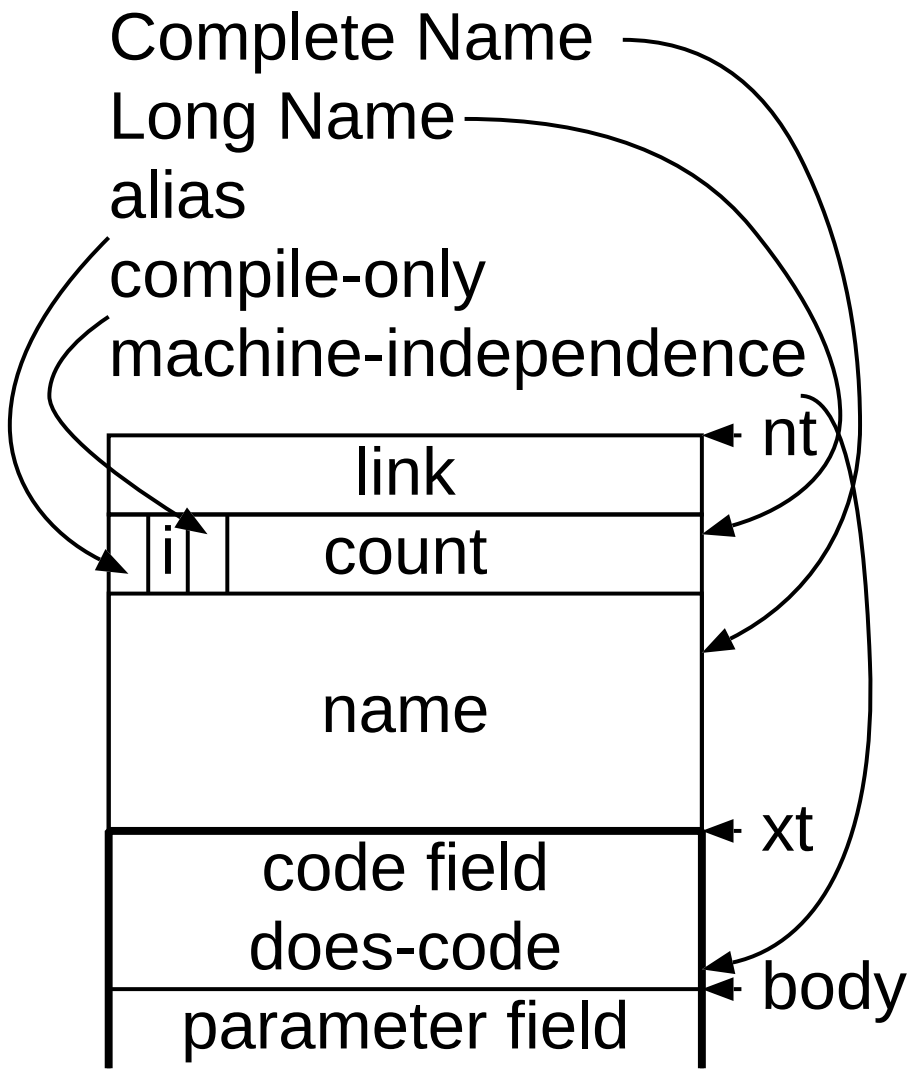
New Requirements



Complete Name
Long Name
alias
compile-only
machine-independence

Old Gforth Header





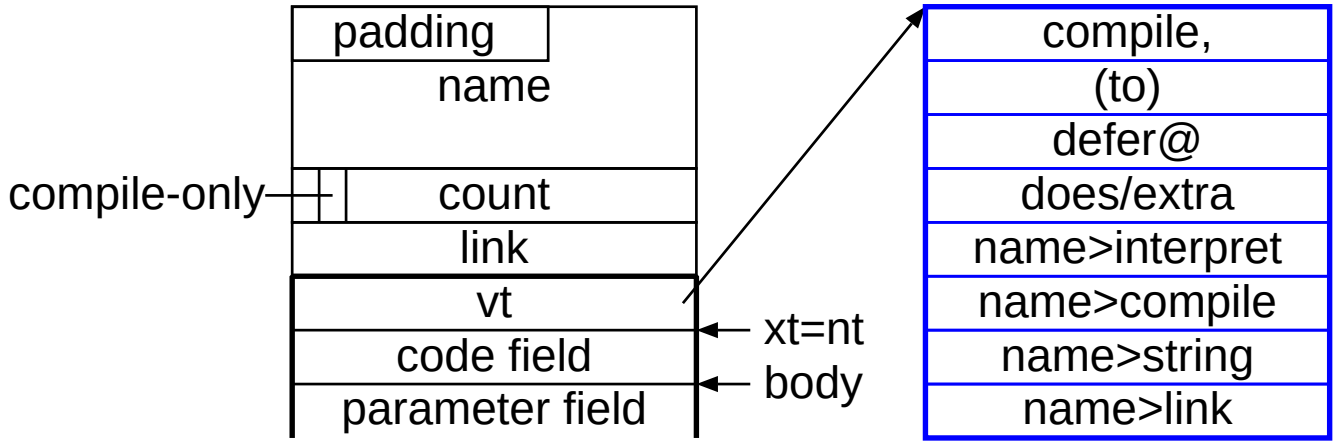
many `ifs`

More Requirements

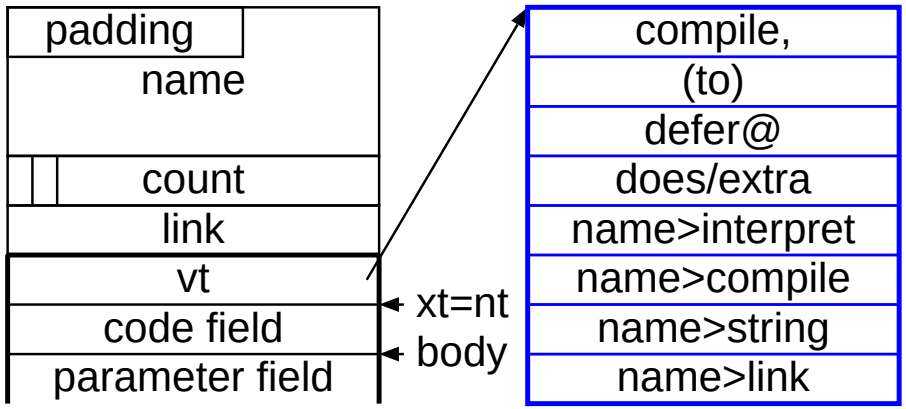
intelligent `compile,`
dual-semantics words
`synonym`
`to` for `value` variations
`defer` variations
reduce `if`s
(mostly) unify nt and xt

New Header

intelligent `compile,`
dual-semantics words
`synonym`
`to` for `value` variations
`defer` variations
reduce `ifs`
(mostly) unify nt and xt



Setters



setter	stack effect	sets
`set-execute`	`( addr -- )`	code field `compile,`

`set-does>`
`does>`
	`( xt -- )`	code field
		`does`
		`compile,`

`set-optimizer` `opt:`	`( xt -- )`	`compile,`
`set->int`	`( xt -- )`	`name>interpret`

`set->comp`
`compsem:`
`immediate`
	`( xt -- )`	`name>compile`

`set-to`	`( xt -- )`	`(to)/defer!`
`set-defer@`	`( xt -- )`	`defer@`
`set->string`	`( xt -- )`	`name>string`
`set->link`	`( xt -- )`	`name>link`

Example: `Immediate`

```
: imm>comp ( nt -- xt1 xt2 )
    name>int ['] execute ;

: immediate ( -- )
    ['] imm>comp set->comp ;

: default-name>comp ( nt -- xt1 xt2 )
    name>int ['] compile, ;

defer name>x
: [ ['] name>interpret is name>x false state ! ;
: ] ['] name>compile   is name>x true  state ! ;

\ in text interpreter:
  ( c-addr u ) 2dup find-name dup if
    nip nip name>x execute
  else
    ( ... number handling ... ) then
```

Example: `Constant`

```
: constant ( x "name" -- )
  create ,
  ['] @ set-does>
  [: >body @ ]] literal [[ ;] set-optimizer ;
```

Example: `Fvalue`

```
: fvalue-to ( r xt-fvalue -- )
    >body f! ;
opt: drop ]] >body f! [[ ;

: fvalue ( r "name" -- ) \ float-ext
  fconstant
  [: >body ]] Literal f@ [[ ;] set-optimizer
  ['] fvalue-to set-to ;
```

Example: `Defer`

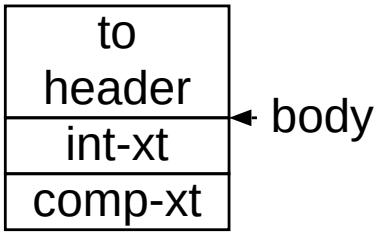
```
: value-to ( x xt -- )
  >body ! ;
opt: ( xt -- ) \ run-time: ( x -- )
  drop ]] >body ! [[ ;

: defer-defer@ ( xt1 -- xt2 )
  >body @ ;
opt: ( xt -- )
  drop ]] >body @ [[ ;

: perform @ execute ;

: defer ( "name" -- )
  create ['] abort ,
  ['] perform set-does>
  [: >body ]] literal perform [[ ;] set-optimizer
  ['] value-to set-to
  ['] defer-defer@ set-defer@ ;
```

Example: Dual-semantics: `To`

```
: to-int ( v "name" -- )
  parse-name find-name dup 0= -13 and throw
  (to) ;

: to-comp ( compilation: "name" -- )
          ( run-time: v -- )
  parse-name find-name dup 0= -13 and throw
  ]] literal (to) [[ ; immediate

: to-name>comp ( nt -- xt1 xt2 )
  drop ['] to-comp ['] execute ;

: to to-int ;
' to-name>comp set->comp
```

Example: `Interpret/compile:`

```
: i/c>comp ( nt -- xt1 xt2 )
    >body cell+ @ [’] execute ;

: interpret/compile: ( int-xt comp-xt "name" --)
    defer , latestxt defer!
    [: >body @ ;] set->int
    [’] i/c>comp set->comp
    [’] no-to set-to
    [’] no-defer@ set-defer@ ;

’ to-int ’ to-comp interpret/compile: to
```

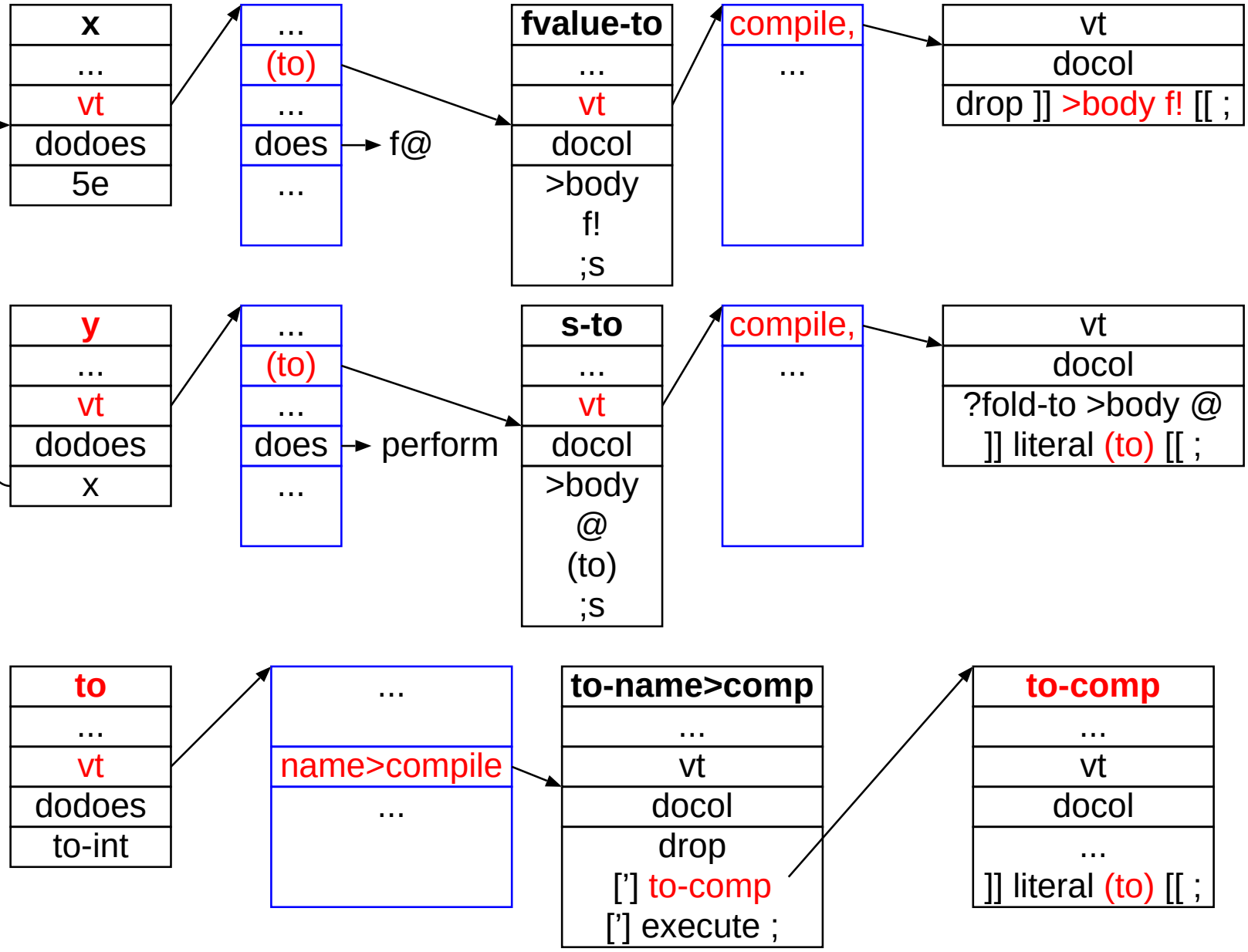


Example: Synonym

```
: s-to ( val nt -- )
  >body @ (to) ;
opt: ( xt -- )
  ?fold-to >body @ ]] literal (to) [[ ;

: s-defer@ ( xt1 -- xt2 )
  >body @ defer@ ;
opt: ( xt -- )
   ?fold-to >body @ ]] literal defer@ [[ ;

: synonym ( "name" "oldname" -- )
  defer
  parse-name find-name dup 0= -13 and throw
  dup lastxt defer!
  compile-only? if compile-only then
  [: >body @ compile,        ;] set-optimizer
  [: >body @ name>interpret ;] set->int
  [: >body @ name>compile   ;] set->comp
  ['] s-to set-to
  ['] s-defer@ set-defer@ ;
```

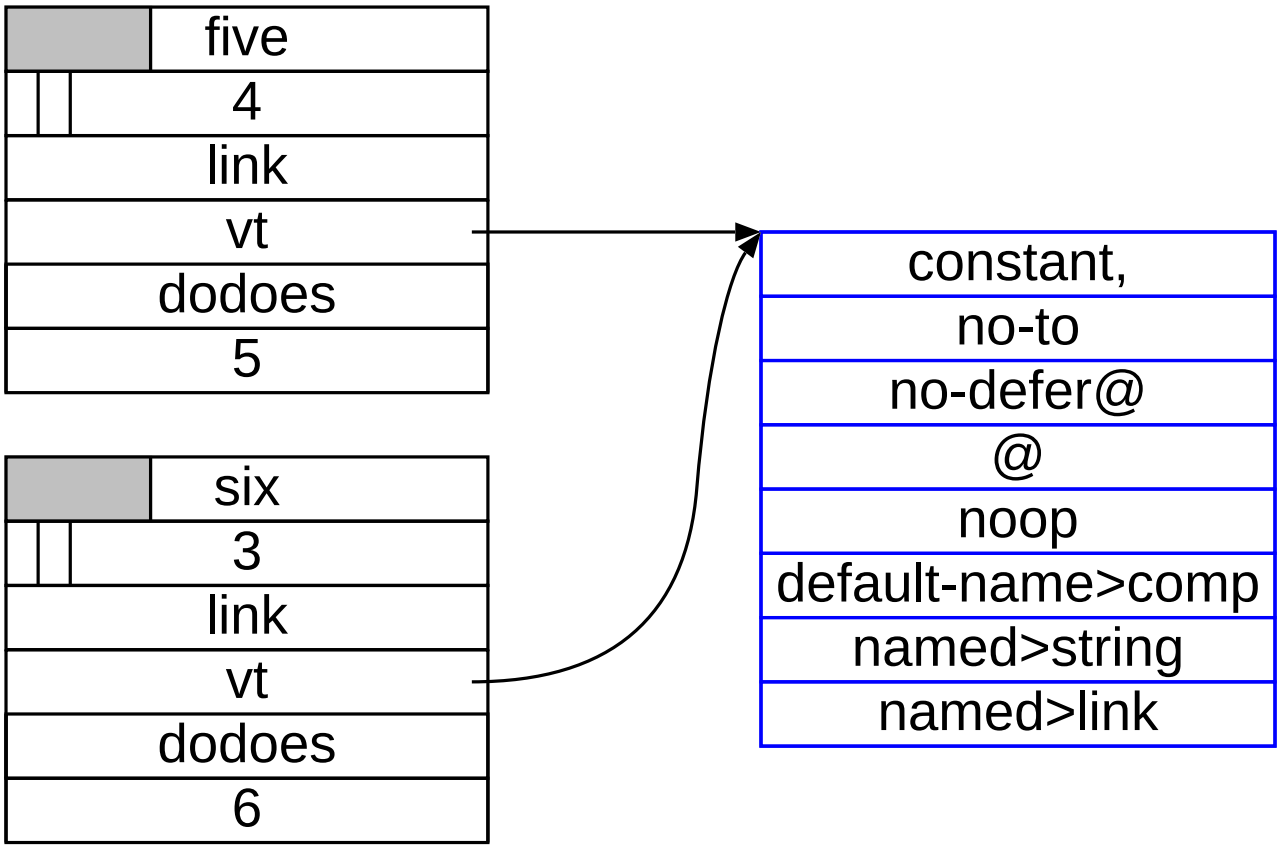
Example: data structures

```
5e fvalue x
synonym y x

: foo to y ;
      ⇓
: foo
 [ ’ x >body ]
 literal f! ;
```



Deduplication

5 constant five
6 constant six



117 vts in Gforth (4000 words)

Create from a prototype

```
create constant-prototype
['] @ set-does>
[: >body @ ]] literal [[ ;] set-optimizer

: constant ( x "name" -- )
  ['] constant-prototype create-from , reveal ;
```

NT=XT?

- For most words: nt = xt

- Exceptions:

  `synonym`

  `alias`

  `interpret/compile:`

- Plausible results for the exceptions

Out-of-band data

- Hash table
- `locate` data

Why `name>comp` *and* `compile,`?

- compilation semantics: `name>compile`
- code generation, optimization: `compile,`
  equivalent to
  `: compile, ]] literal execute [[ ;`
  Equivalence is widely used

Conclusion

- New requirements ⇒ header redesign
- Prototype-based object-oriented approach
- Method dispatch
  replaces `if`-cascades
  better extensibility
- Supports
  optimization, dual-semantics, `synonym`, `to`, `defer` variations
- Deduplication and `create-from`
- nt=xt, with exceptions