From `exit` to `set-does>`

A Story of Gforth Re-Implementation

M. Anton Ertl, TU Wien
Bernd Paysan, net2o

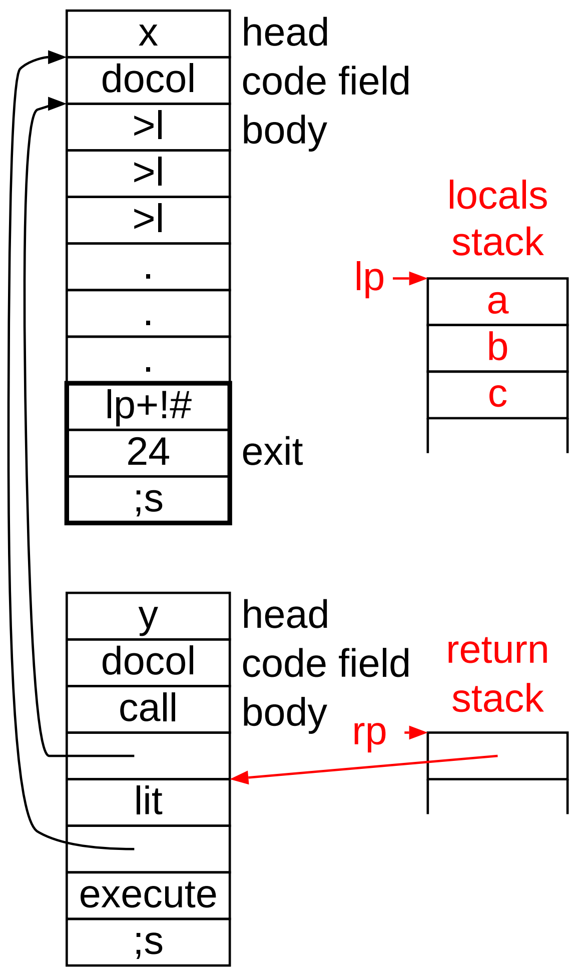
`[´] exit execute`

- `Exit` not tickable in Gforth 0.7
- Standard-conforming, but
- Several complaints over the years
- Traditionally `exit` was a normal word

Exit and locals

```
: x { a b c } ... ;
: y x ['] x execute ;
```

old



x
docol
>l
>l
>l
.
.
.
lp+!#
24
;s

head
code field
body
exit

locals
stack
lp
a
b
c

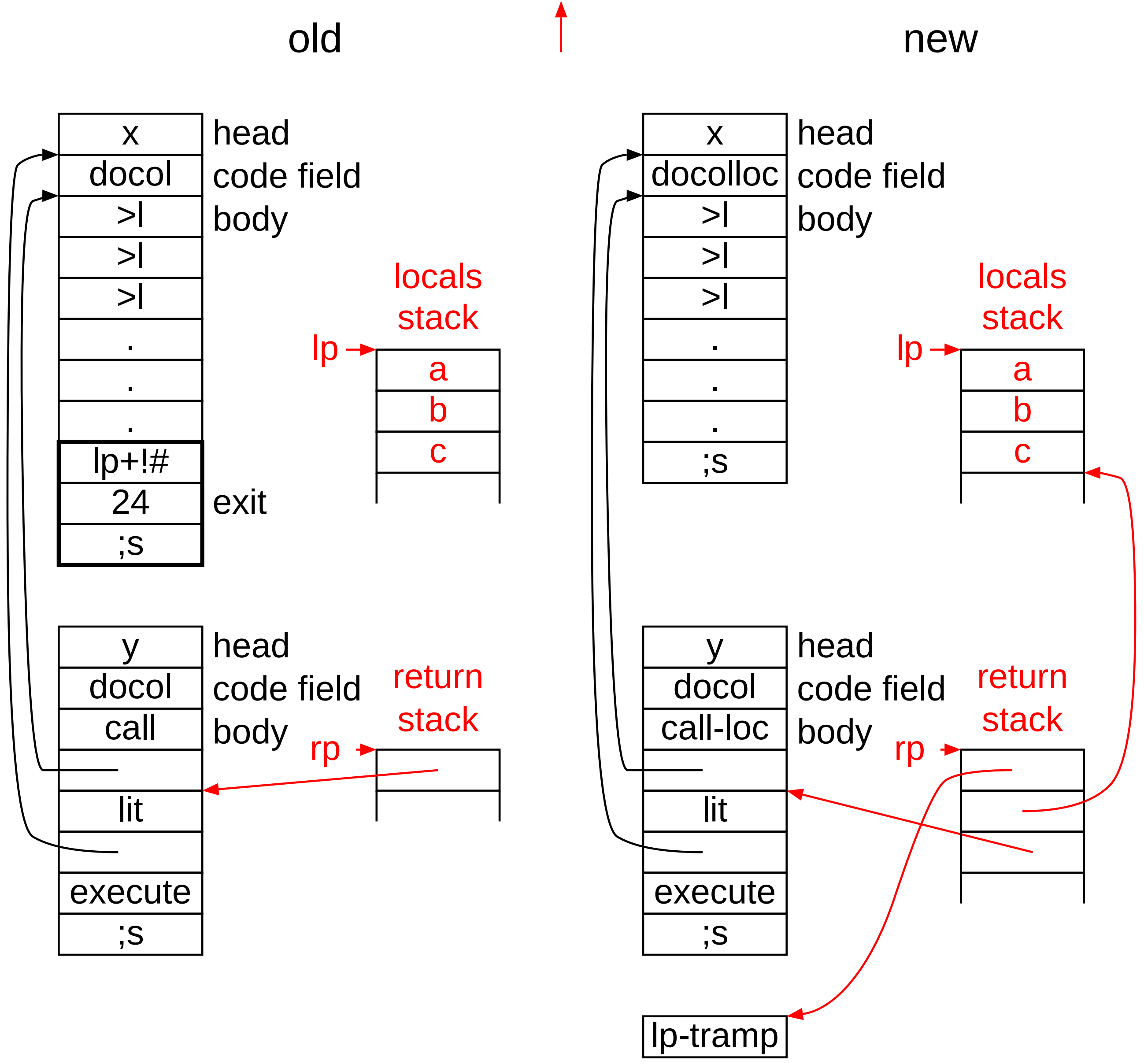
y
docol
call
lit
execute
;s

head
code field
body

return
stack
rp

New implementation

```
: x { a b c } ... ;
: y x ['] x execute ;
```



old

x
docol
>l
>l
>l
.
.
.
lp+!#
24
;s

head
code field
body
exit

locals
stack
lp
a
b
c

y
docol
call

lit

execute
;s

head
code field
body

return
stack
rp

new

x
docolloc
>l
>l
>l
.
.
.
;s

head
code field
body

locals
stack
lp
a
b
c

y
docol
call-loc

lit

execute
;s

head
code field
body

return
stack
rp

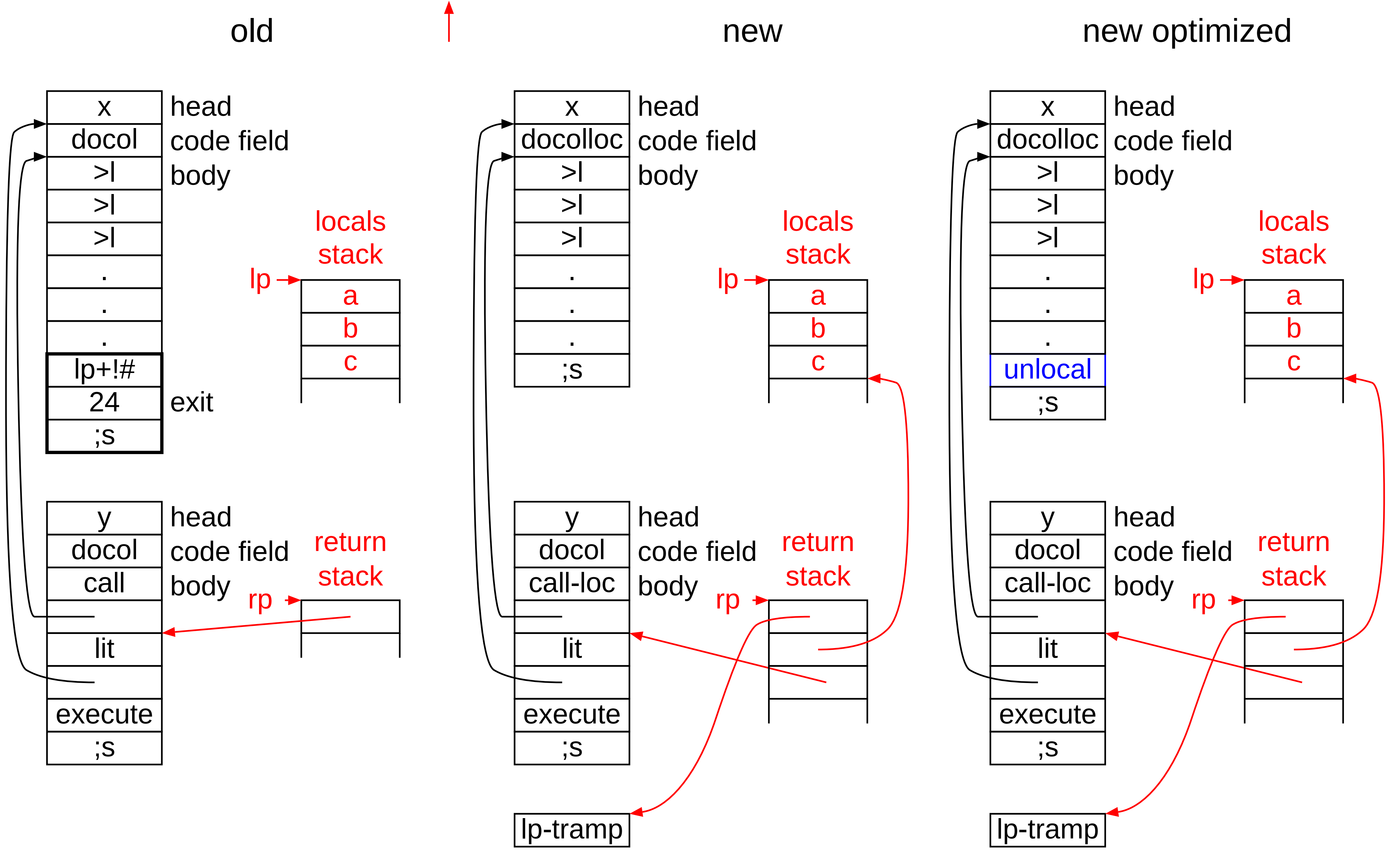
lp-tramp

Return-address manipulation

```
: foo r> drop unlocal exit ;
: bar { a } foo ;
```

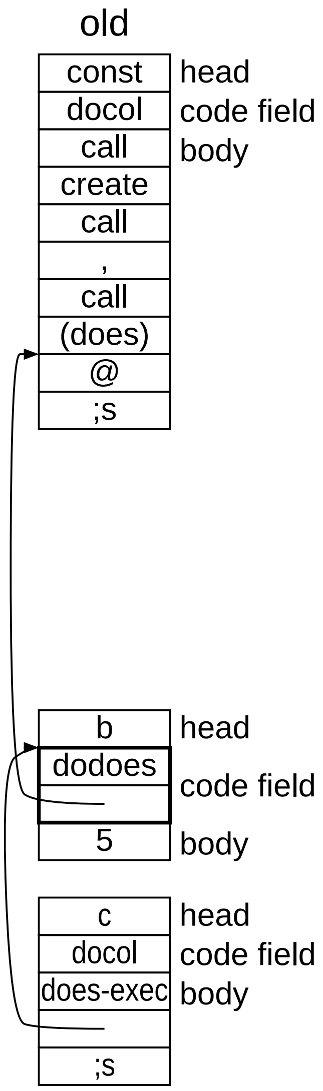
Optimization

```
: x { a b c } ... ;
: y x ['] x execute ;
```



What about `does>`?

```
: const create , does> @ ;
5 const b
: c b ;
```

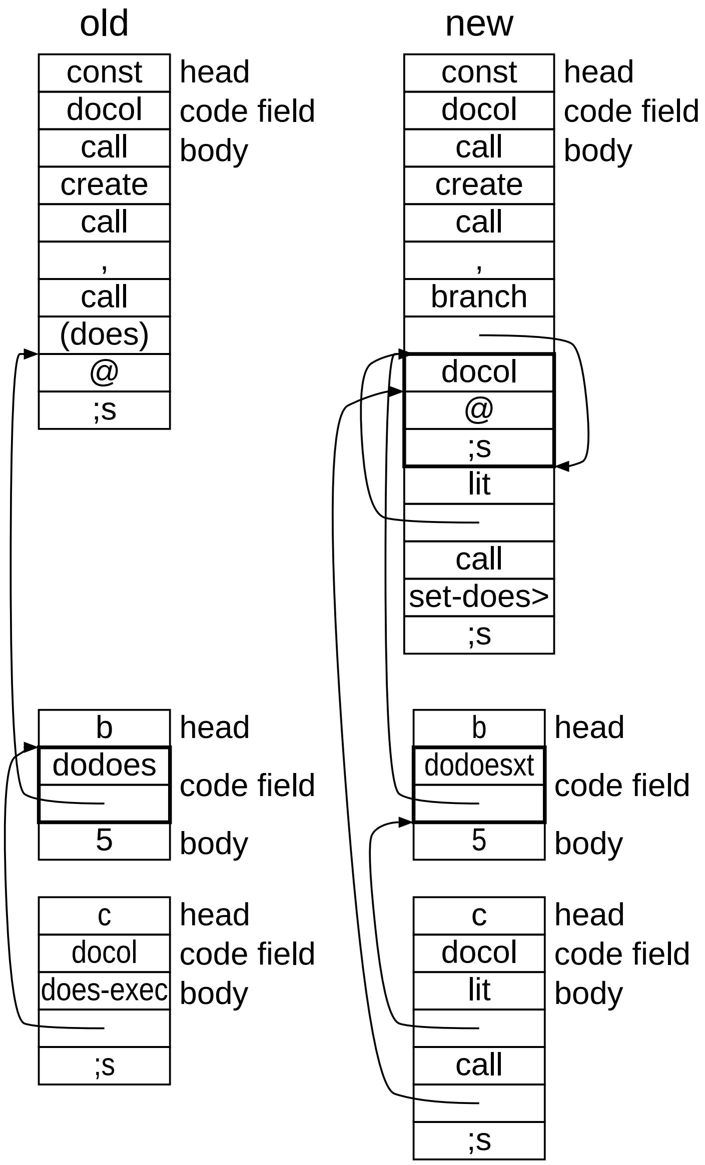


```
: def
  create
does>
  { a } [’] exit execute ;

def x
x
```

New does> implementation

```
: const create , does> @ ;
5 const b
: c b ;
```



old

const	head
docol	code field
call	body
create	
call	
,	
call	
(does)	
@	
;s	

b	head
dodoes	code field
5	body

c	head
docol	code field
does-exec	body
;s	

new

const	head
docol	code field
call	body
create	
call	
,	
branch	
docol	
@	
;s	
lit	
call	
set-does>	
;s	

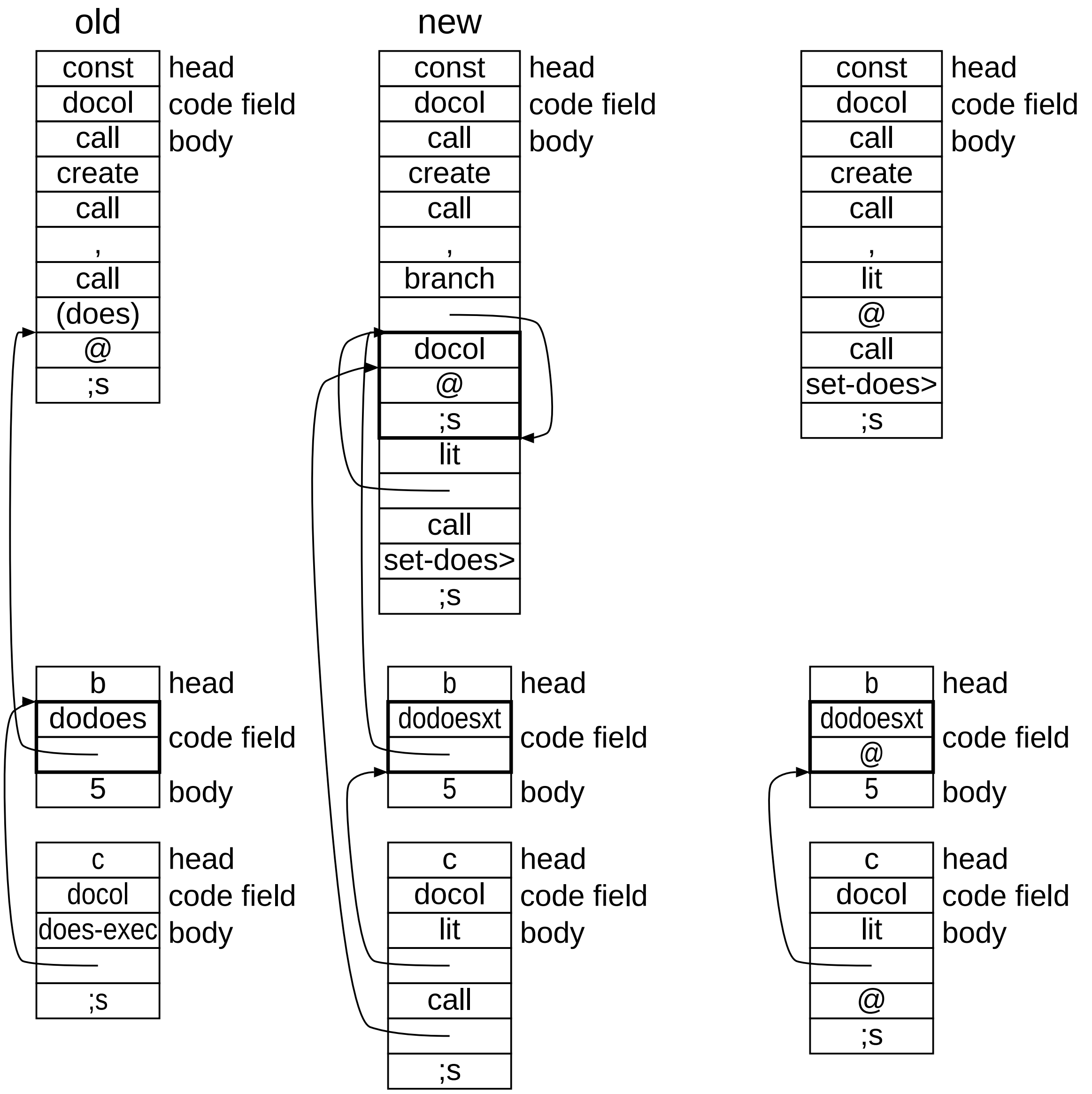
b	head
dodoesxt	code field
5	body

c	head
docol	code field
lit	body
call	
;s	

Set-does>

```
: const create , does> @ ;
5 const b
: c b ;
```

```
: const create , ['] @ set-does> ;
5 const b
: c b ;
```



old

const | head
docol | code field
call | body
create
call
,
call
(does)
@
;s

b | head
dodoes | code field
5 | body

c | head
docol | code field
does-exec | body
;s

new

const | head
docol | code field
call | body
create
call
,
branch
docol
@
;s
lit
call
set-does>
;s

b | head
dodoesxt | code field
5 | body

c | head
docol | code field
lit | body
call
;s

const | head
docol | code field
call | body
create
call
,
lit
@
call
set-does>
;s

b | head
dodoesxt | code field
@
5 | body

c | head
docol | code field
lit | body
@
;s

Other set-does> benefits

```
: dofield ( -- )
does> ( name execution:
        addr1 -- addr2 )
    @ + ;

: dozerofield ( -- )
    immediate
does> ( name execution: -- )
    drop ;

: field ( align1 off1
          align size "name"
          --  align2 offset2 )
    2 pick >r create-field r> if
        dofield
    else
        dozerofield
    then ;
```

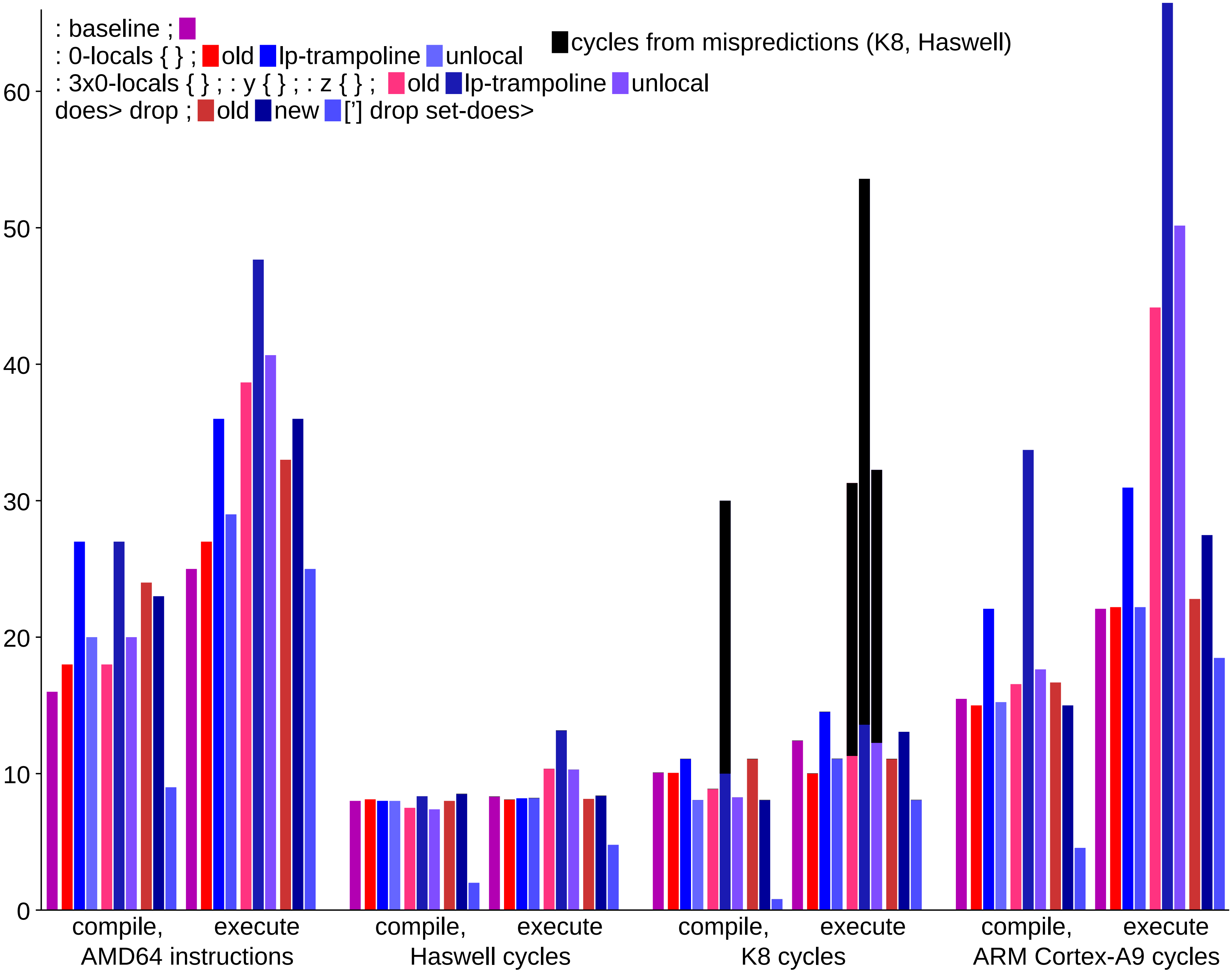
```
: field ( align1 off1
          align size "name"
          --  align2 offset2 )
    2 pick >r create-field r> if
        [: @ + ;] set-does>
    else
        [: ;] set-does> immediate
    then ;
```

Other `set-does>` benefits

```
: doconst does> @ ;

: const ( n "name" -- )            : const ( n "name" -- )
  create , doconst                   create , ['] @ set-does>
  [: >body @ postpone literal ;]     [: >body @ postpone literal ;]
  set-opt ;                          set-opt ;
```

Performance



Native-code caveats

- Return has to match call
- otherwise misprediction (20 cycles penalty)

```
push data needed for cleaning up locals
call rest-of-definition
clean up locals
ret
rest-of-definition:
...
ret
```

Conclusion

- **`['] exit execute`** ⇒
  new locals cleanup ⇒ **`unlocal`** ⇒ optimization
  ⇒ new **`does>`** implementation ⇒ **`set-does>`**

- Helpful:
  intelligent **`compile,`**
  quotations

- **`set-does>`**: more flexibility

- Performance similar with optimizations
  `set-does>` can save a call level