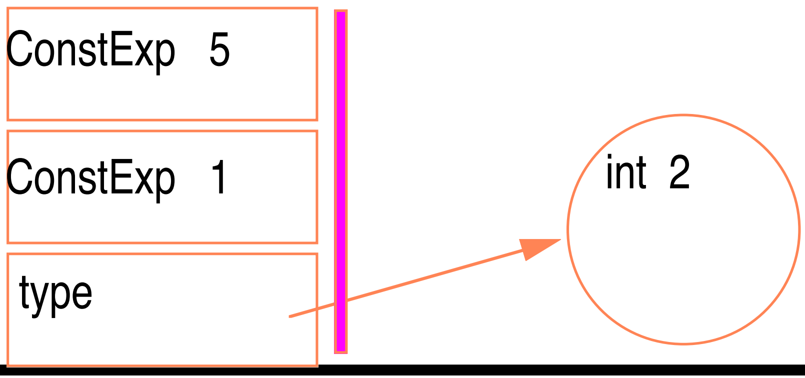
Range Definitions

integer range [1..5] one_five ;

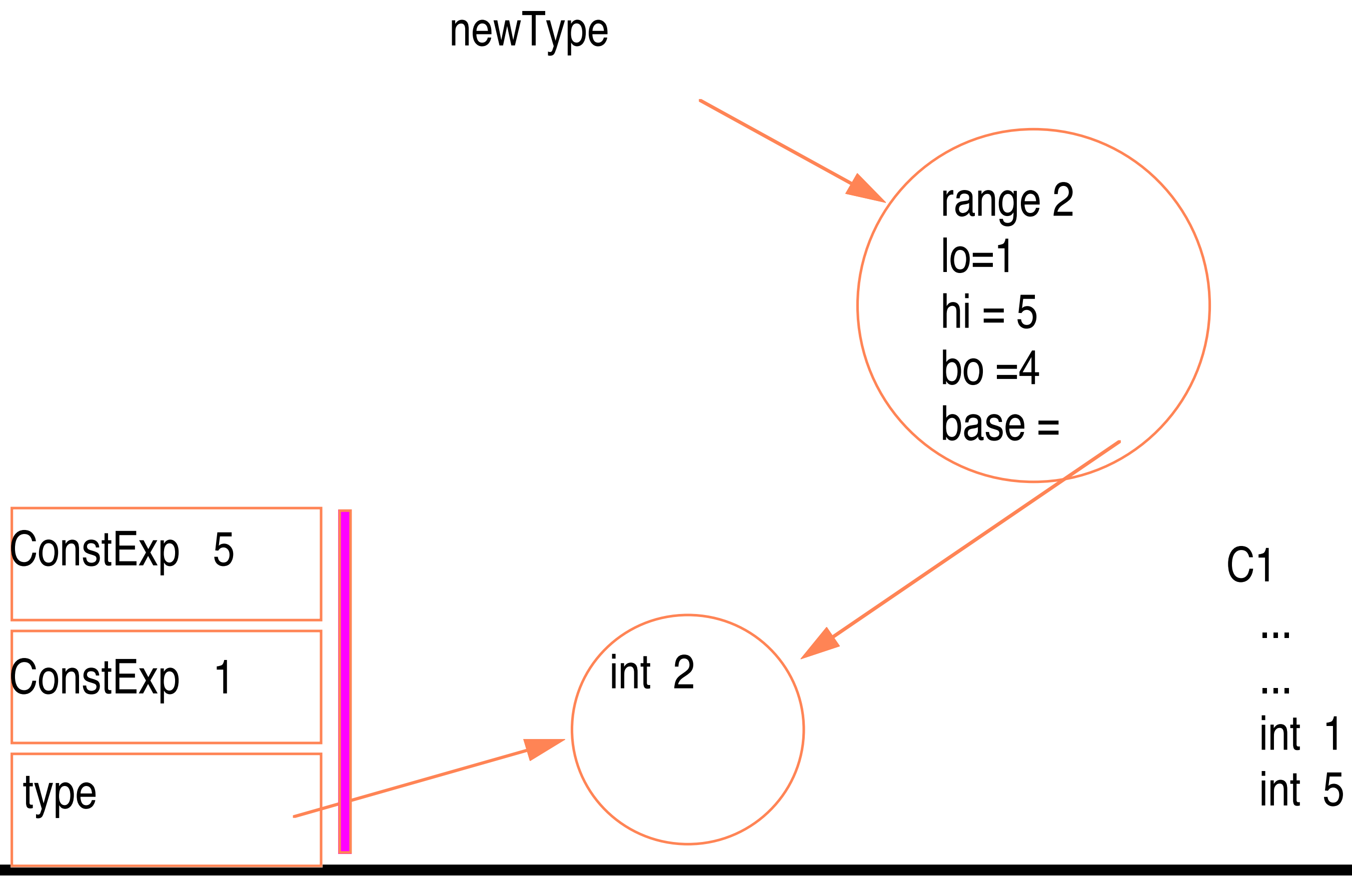
Indicates input parameters to the current semantic procedure.



DoMakeRange

Range Definitions

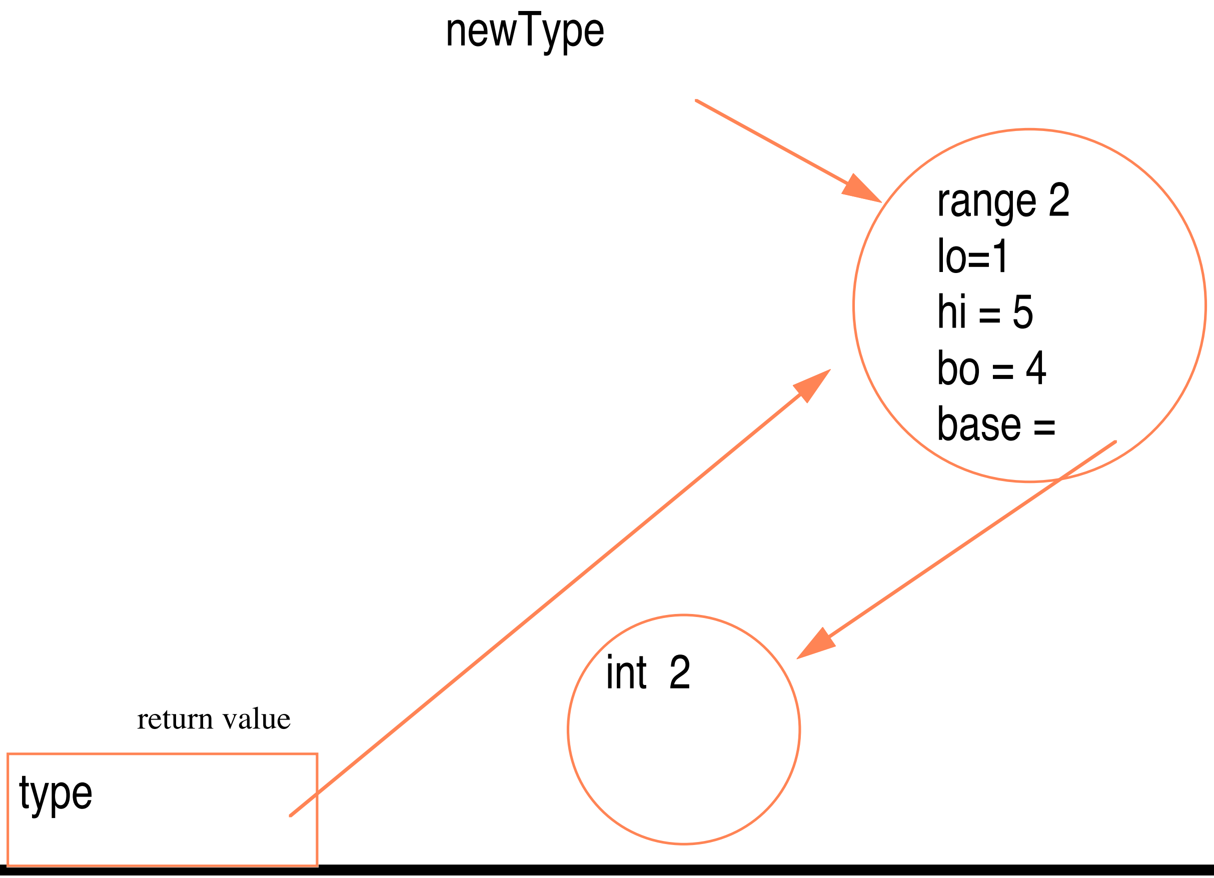
integer range [1..5] one_five ;



DoMakeRange

Range Definitions

integer range [1..5] one_five ;

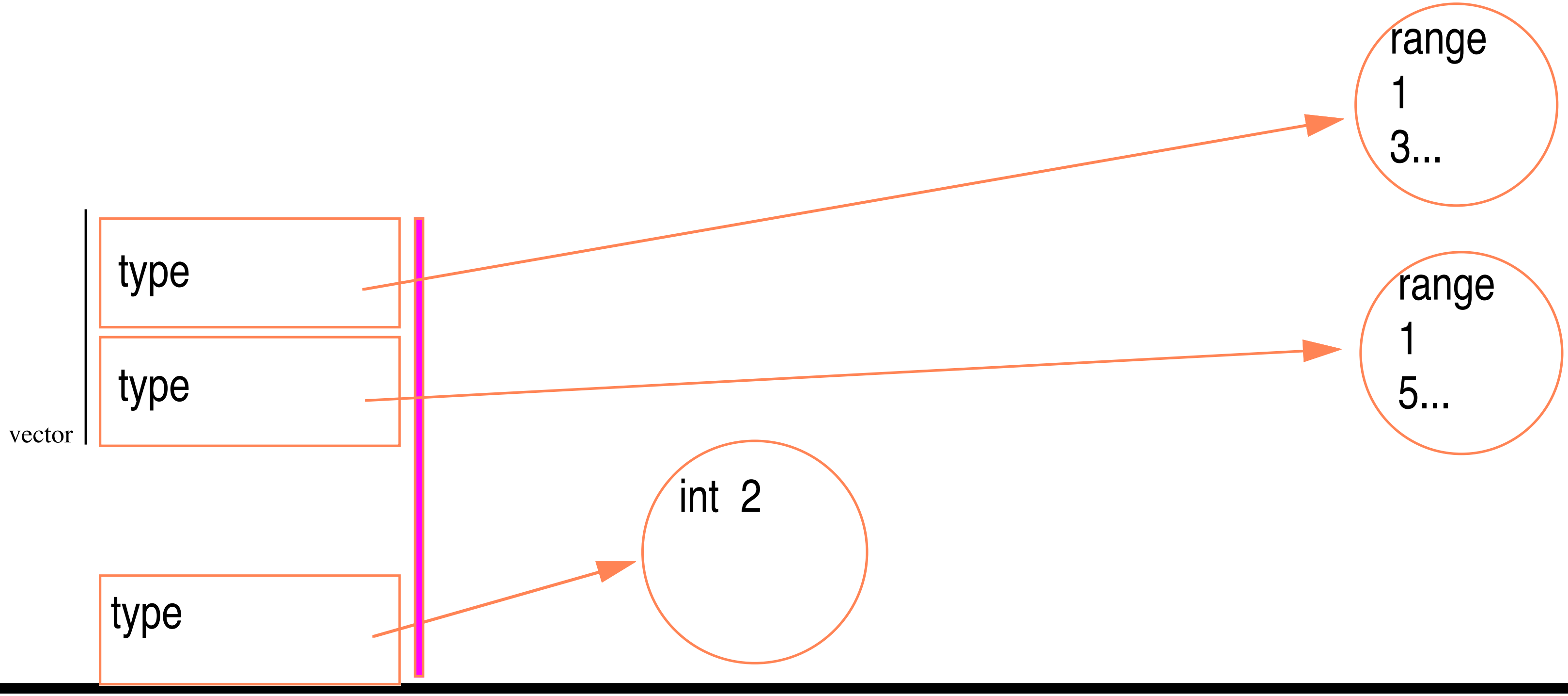


DoMakeRange

Array Definitions

integer array [one_five] [one_three] *a, b ;

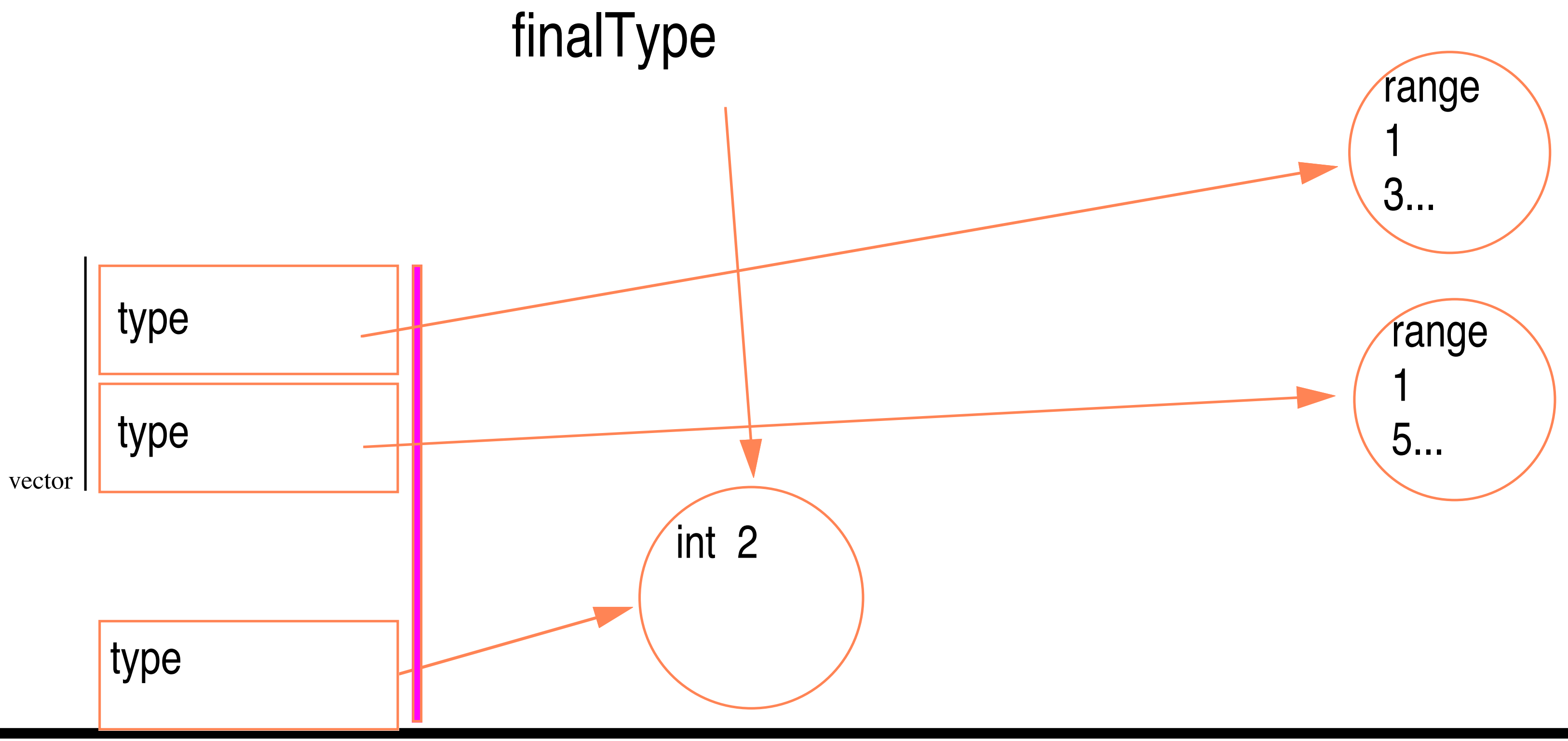
Assumes we have previously defined
integer range [1..5] one_five;
integer range [1..3] one_three;



DoMakeArray

Array Definitions

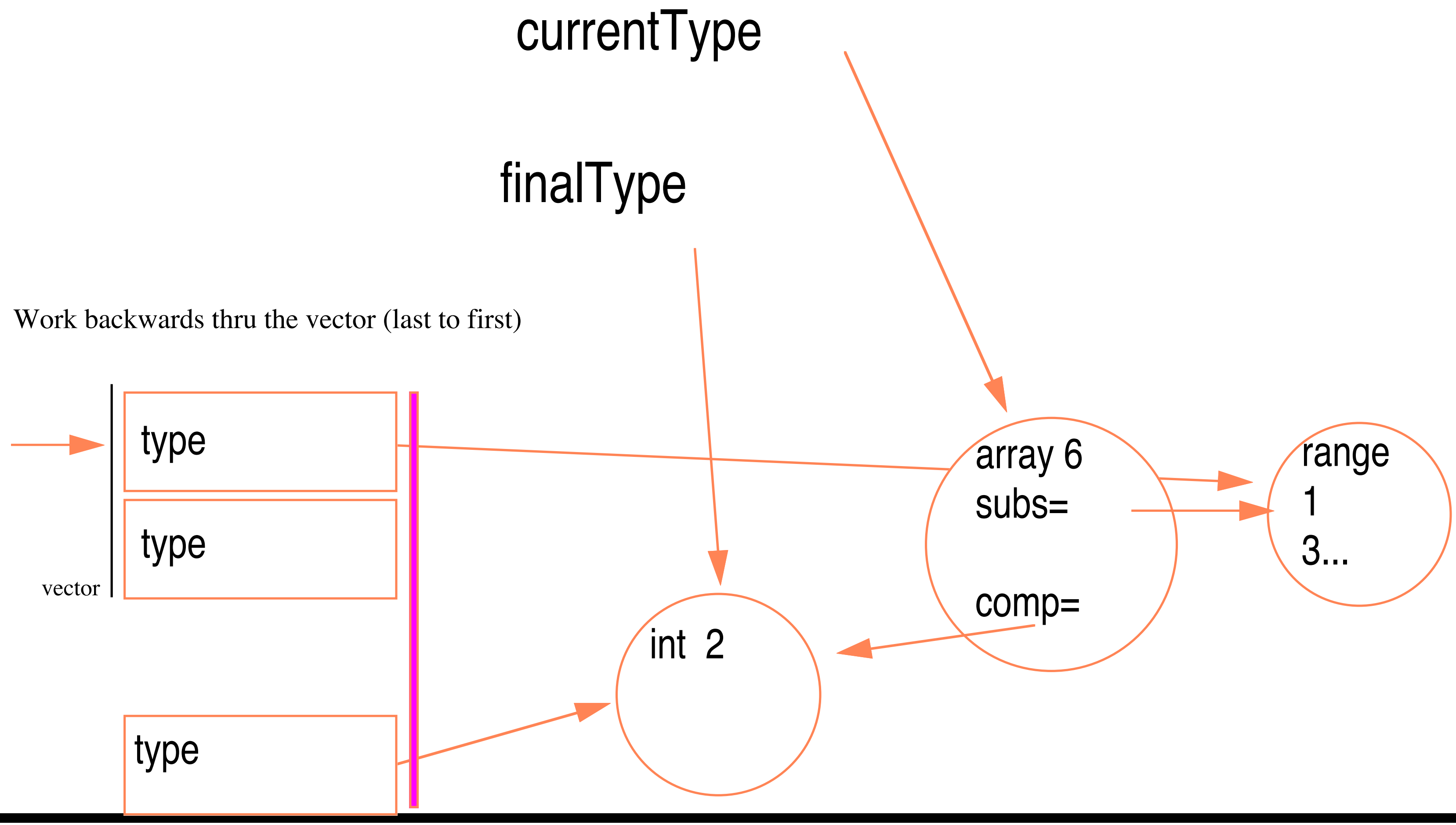
integer array [one_five] [one_three] *a, b ;



in DoMakeArray

Array Definitions

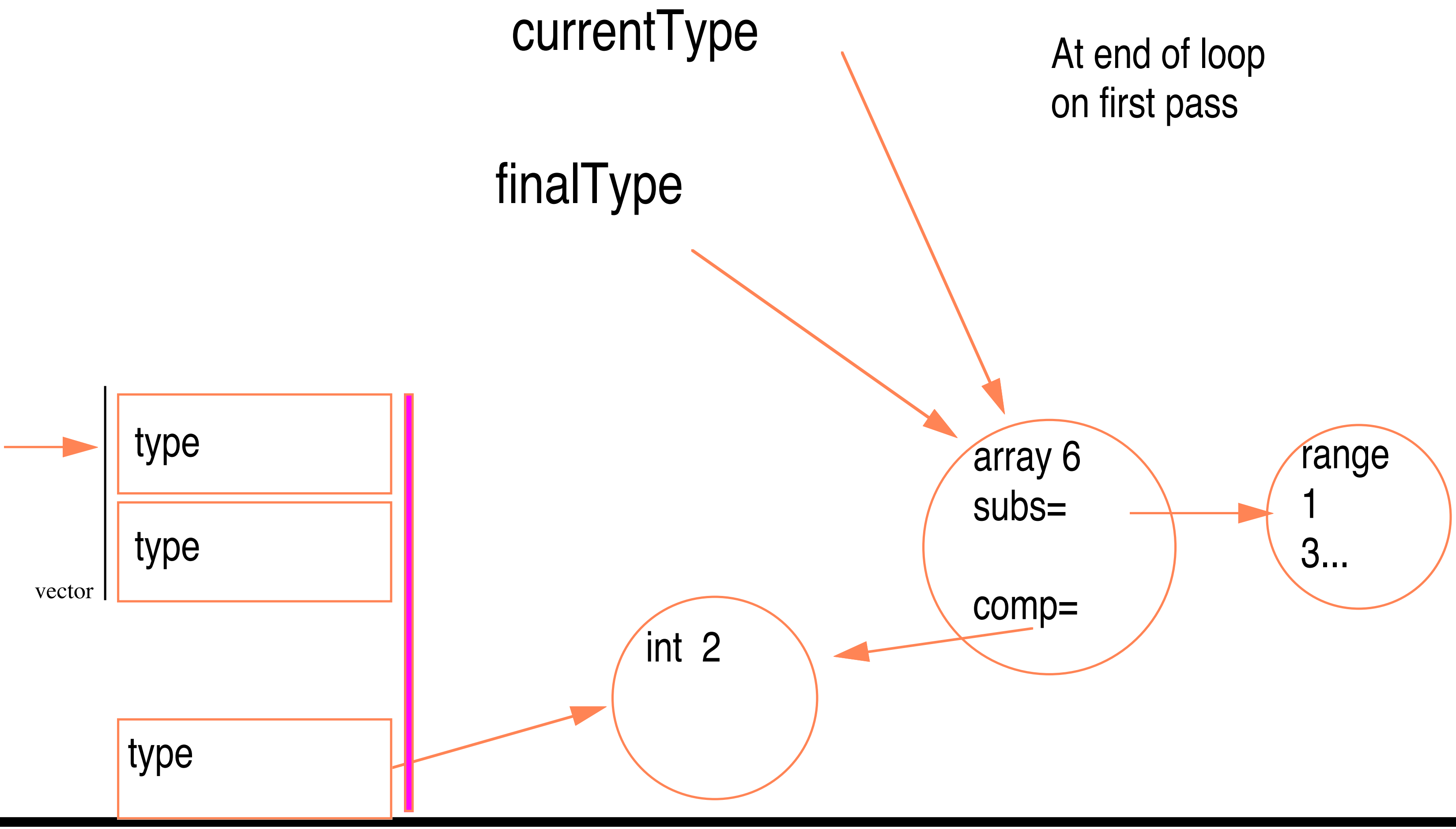
integer array [one_five] [one_three] *a, b ;



in DoMakeArray

Array Definitions

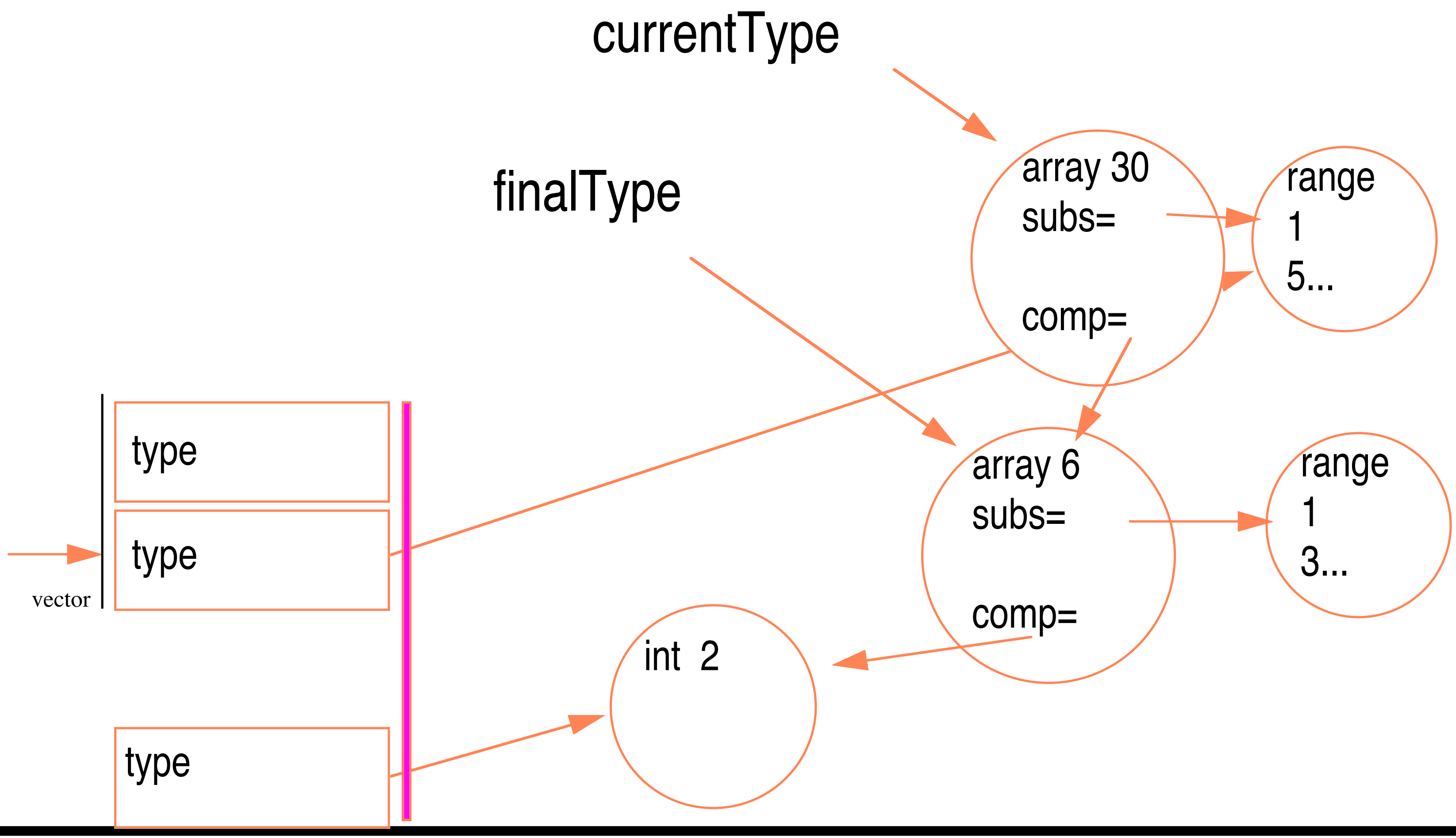
integer array [one_five] [one_three] *a, b ;



in DoMakeArray

Array Definitions

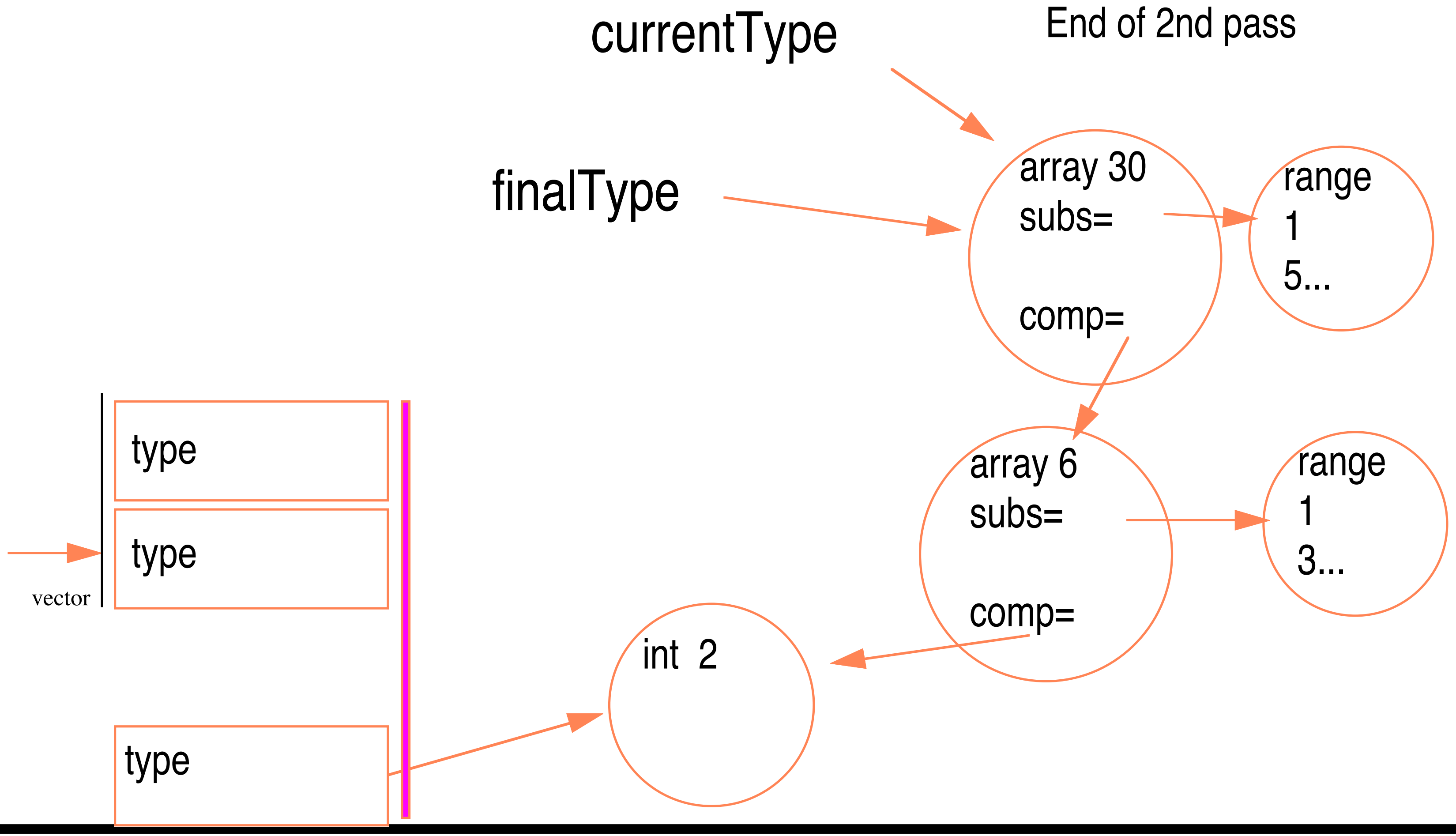
integer array [one_five] [one_three] *a, b ;



in DoMakeArray

Array Definitions

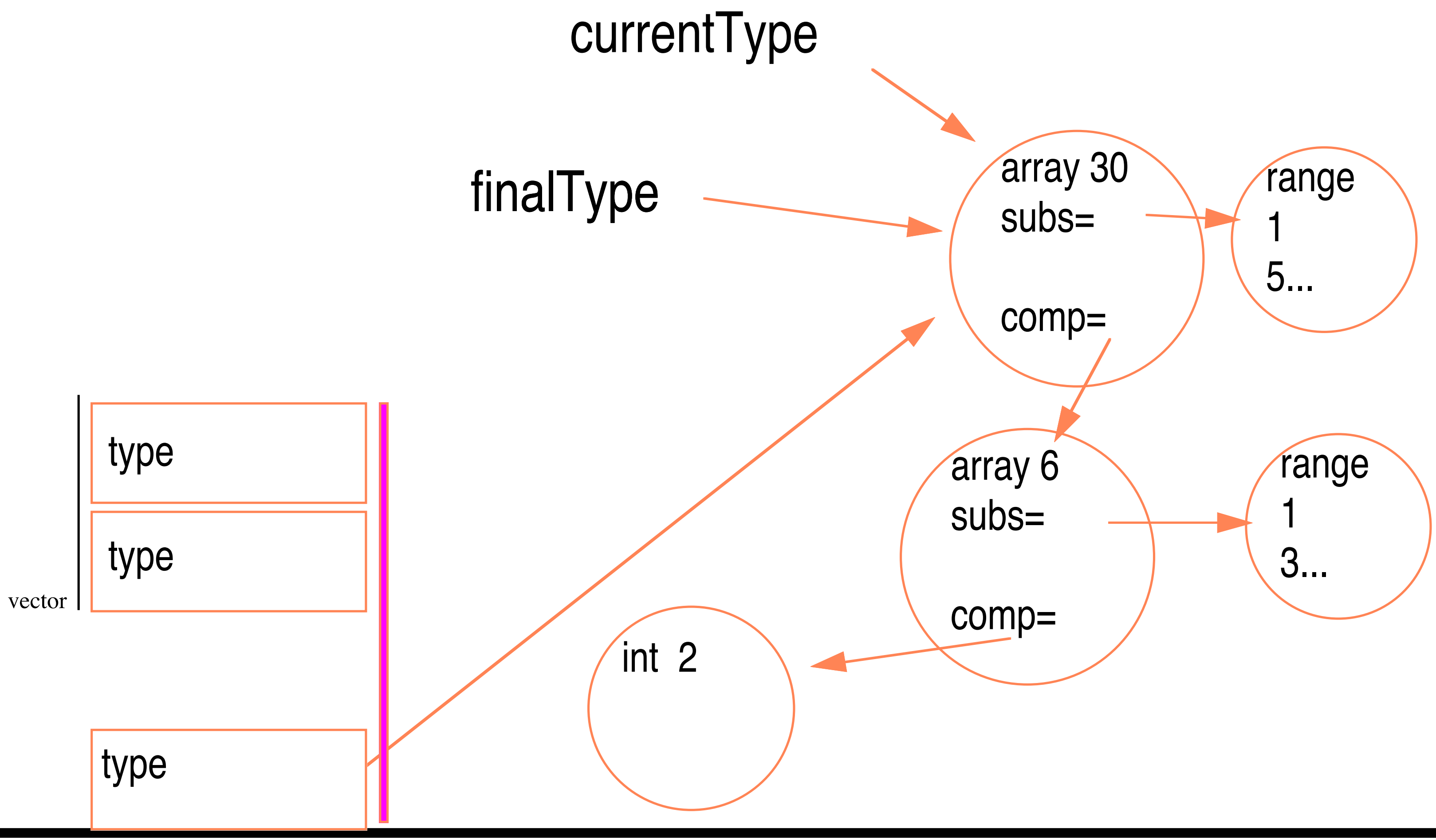
integer array [one_five] [one_three] *a, b ;



in DoMakeArray

Array Definitions

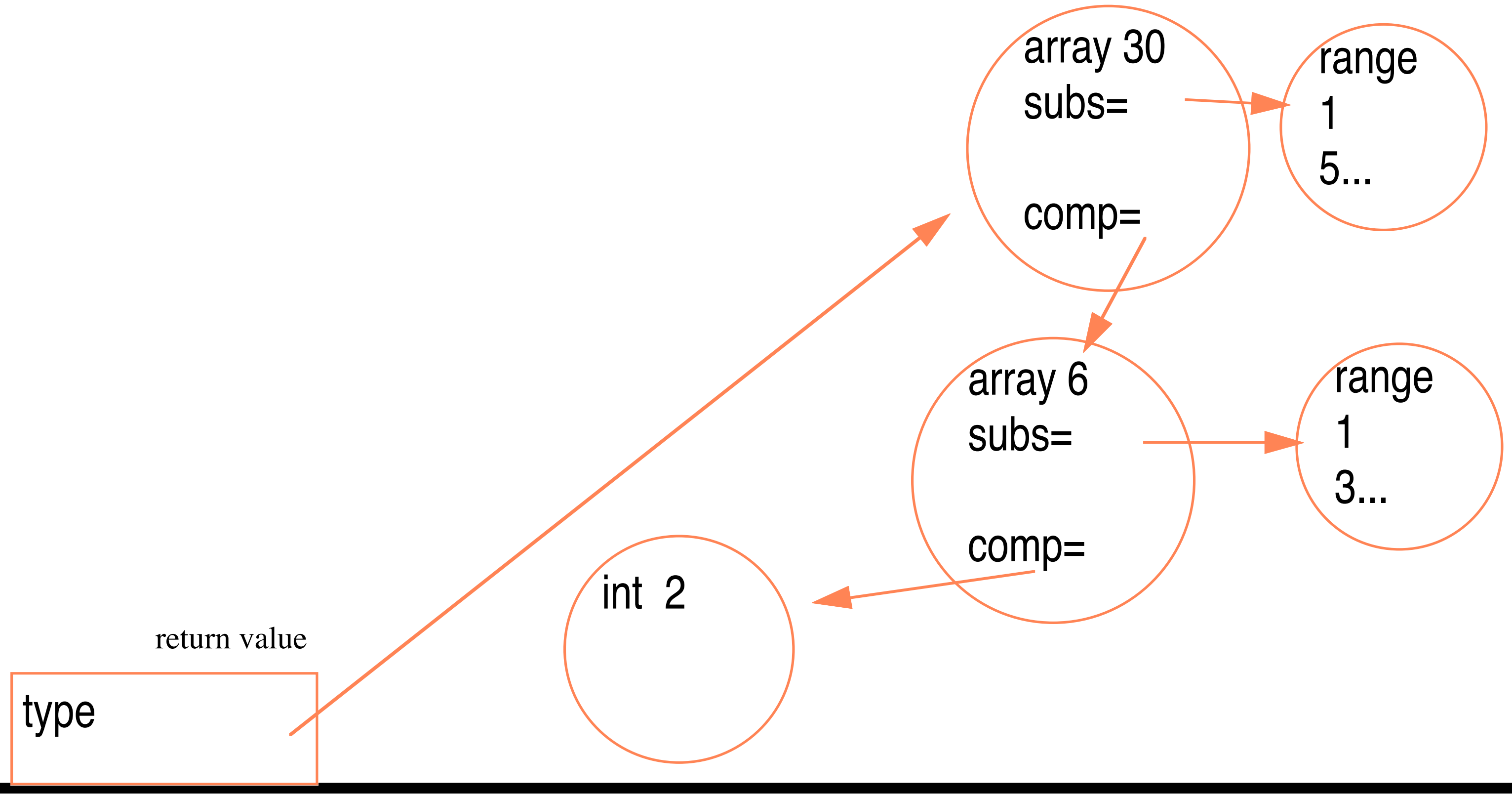
integer array [one_five] [one_three] *a, b
;



in DoMakeArray

Array Definitions

integer array [one_five] [one_three] *a, b ;



DoDeclVars exactly as before

Array Definitions

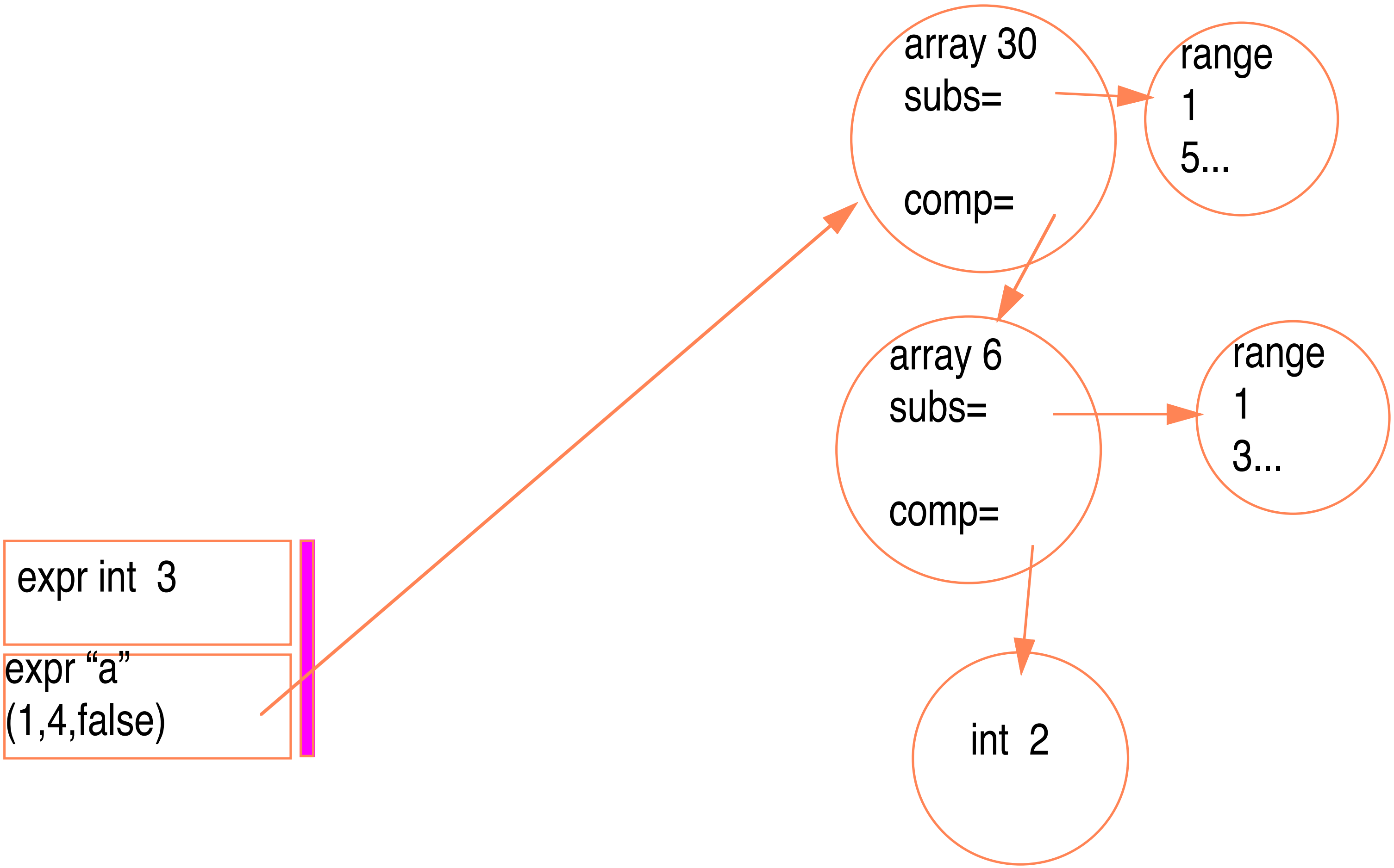
Notes:
Type pointers represent linked lists (if we have records then trees actually)

We may need to “walk” the list to process an array type.

The “head” pointer represents the type as a whole.

Subscript reduction

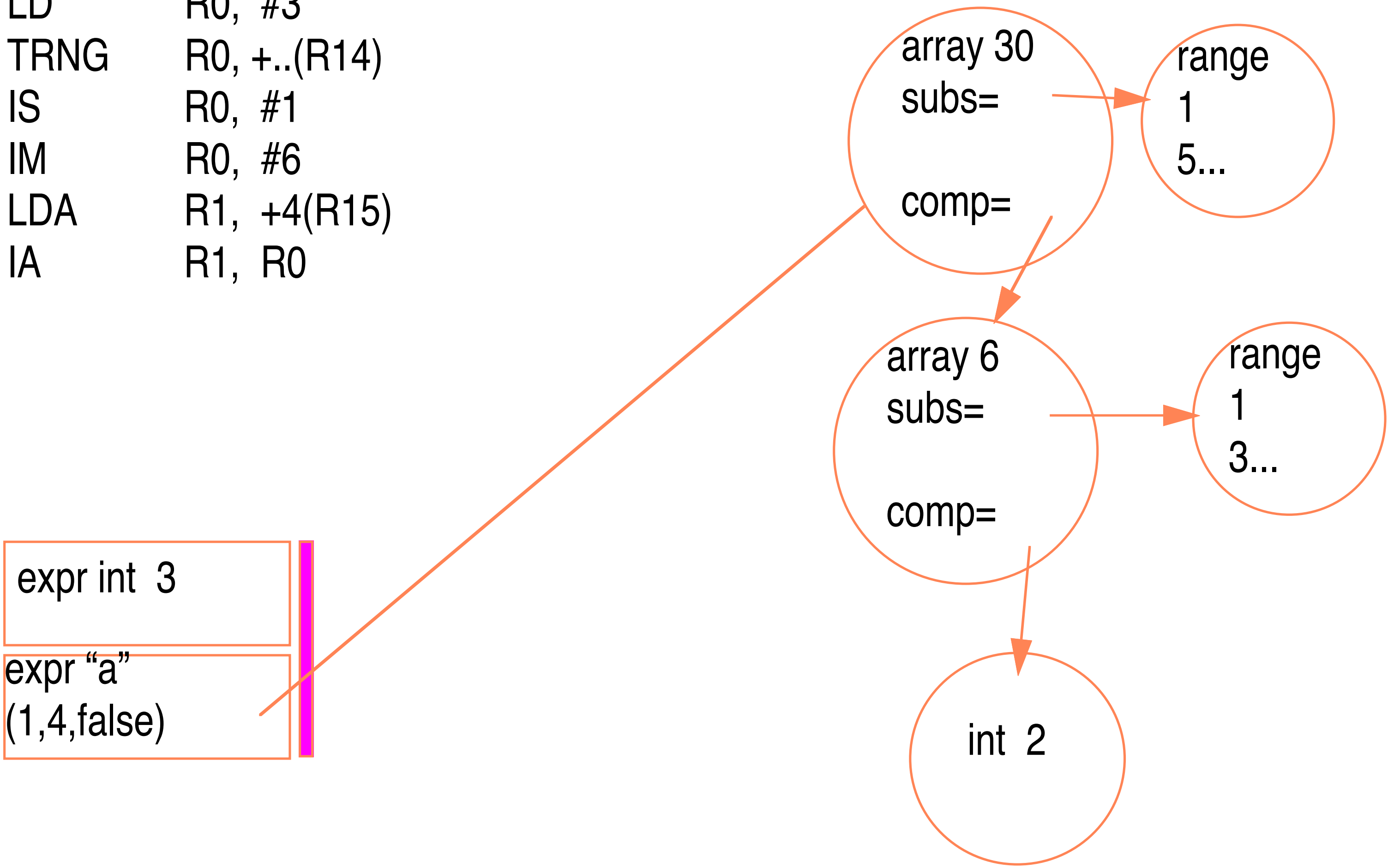
x := a[3] • [m] ;



Subscript reduction

x := a[3] • [m] ;

```
LD      R0,  #3
TRNG    R0, +..(R14)
IS      R0,  #1
IM      R0,  #6
LDA     R1,  +4(R15)
IA      R1,  R0
```

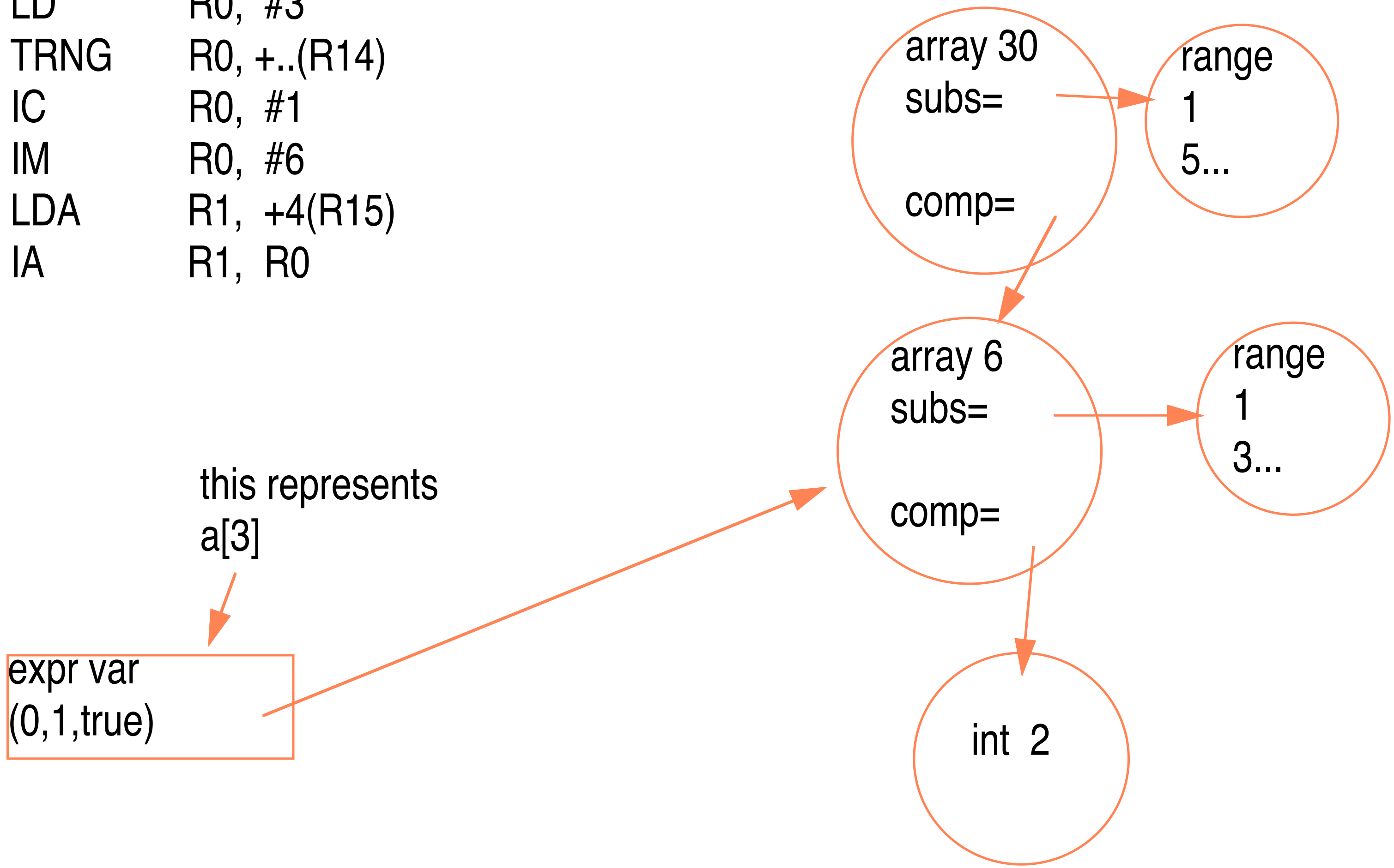


in Subscript

Subscript reduction

x := a[3] • [m] ;

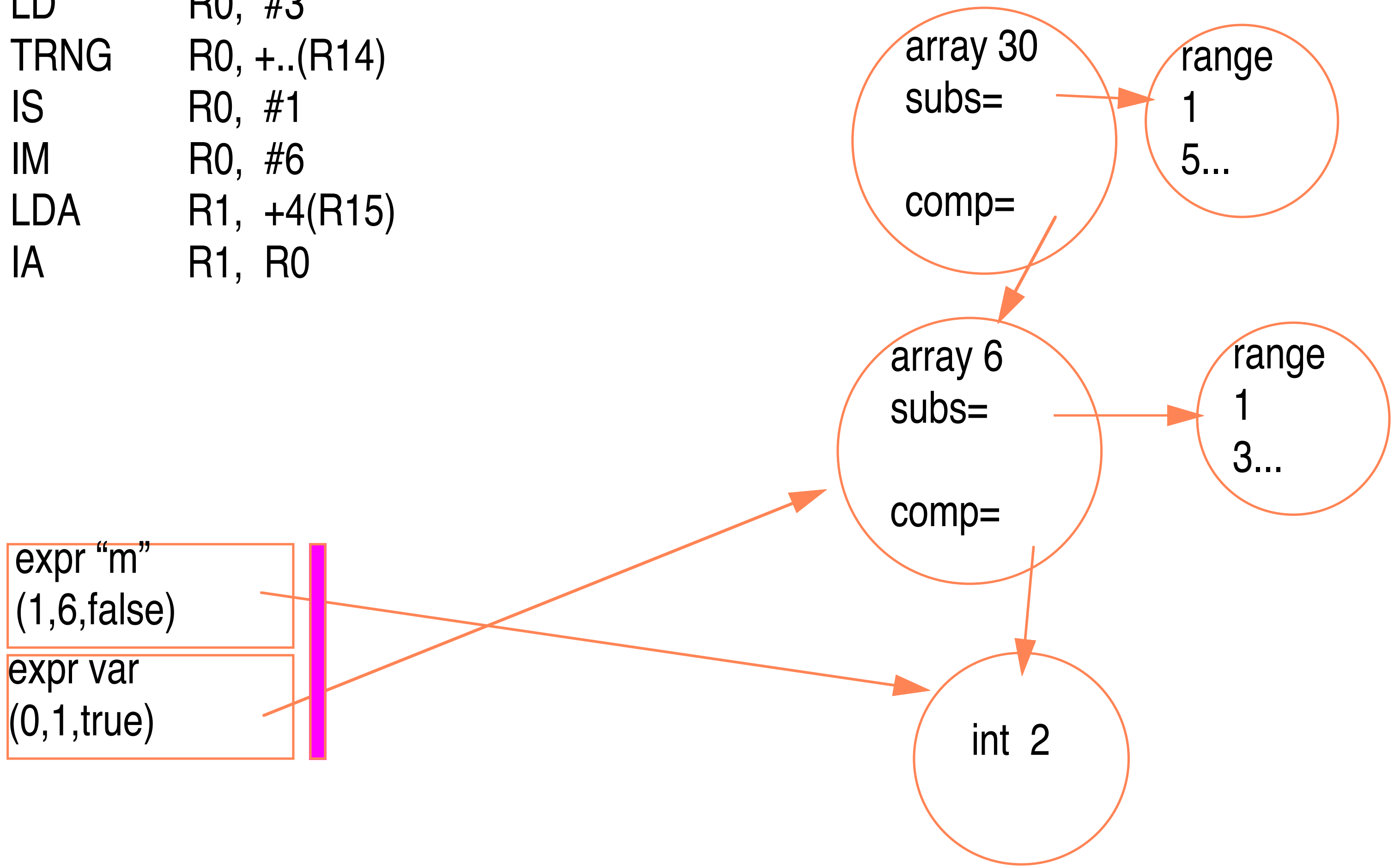
```
LD      R0,  #3
TRNG    R0, +..(R14)
IC      R0,  #1
IM      R0,  #6
LDA     R1,  +4(R15)
IA      R1,  R0
```



Subscript reduction

x := a[3] [m] • ;

```
LD      R0,  #3
TRNG    R0, +..(R14)
IS      R0,  #1
IM      R0,  #6
LDA     R1,  +4(R15)
IA      R1,  R0
```



Subscript(second call)

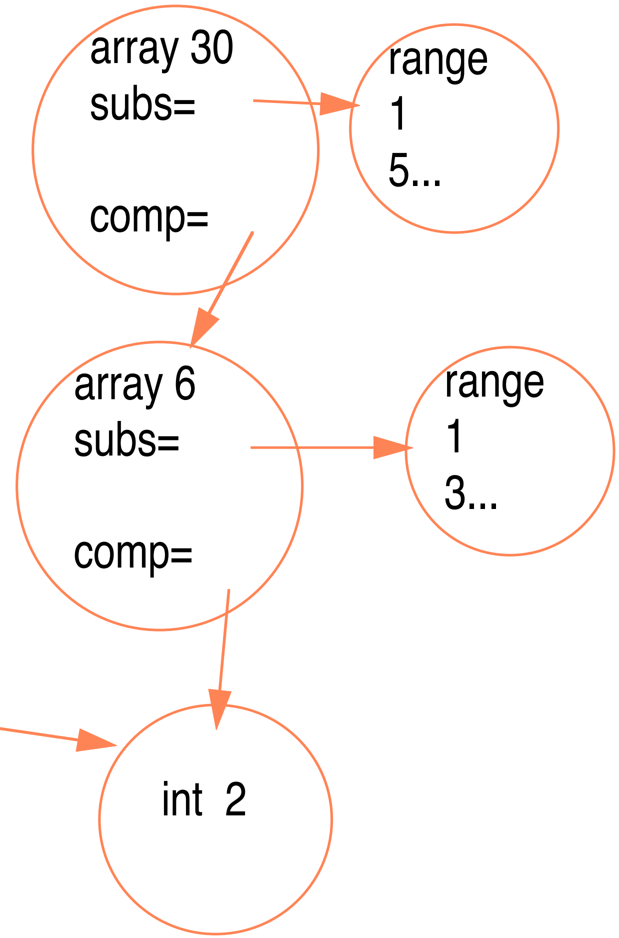
Subscript reduction

x := a[3] [m] • ;

```
LD      R0,  #3
TRNG    R0, +..(R14)
IS      R0,  #1
IM      R0,  #6
LDA     R1,  +4(R15)
IA      R1,  R0
LD      R0,  +6(R15)
TRNG    R0,  +..R14)
IS      R0  #1
IM      R0,  #2
IA      R1,  R0
```

expr “m”
(1,6,false)

expr var
(0,1,true)

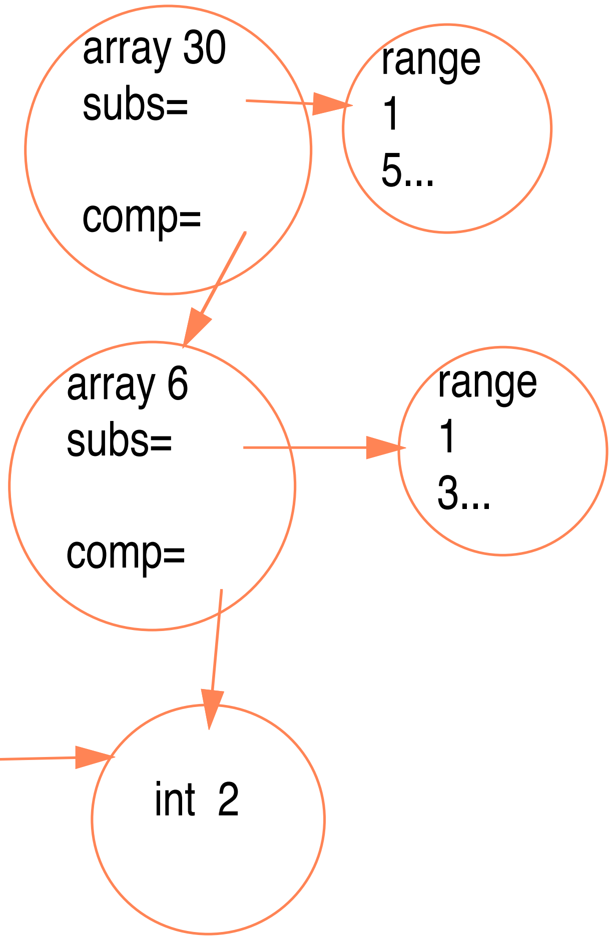


in Subscript(second call)

Subscript reduction

x := a[3] [m] • ;

```
LD      R0,  #3
TRNG    R0, +..(R14)
IS      R0,  #1
IM      R0,  #6
LDA     R1,  +4(R15)
IA      R1,  R0
LD      R0,  +6(R15)
TRNG    R0,  +..R14)
IS      R0  #1
IM      R0,  #2
IA      R1,  R0
```



This represents
a[3] [m]

return value

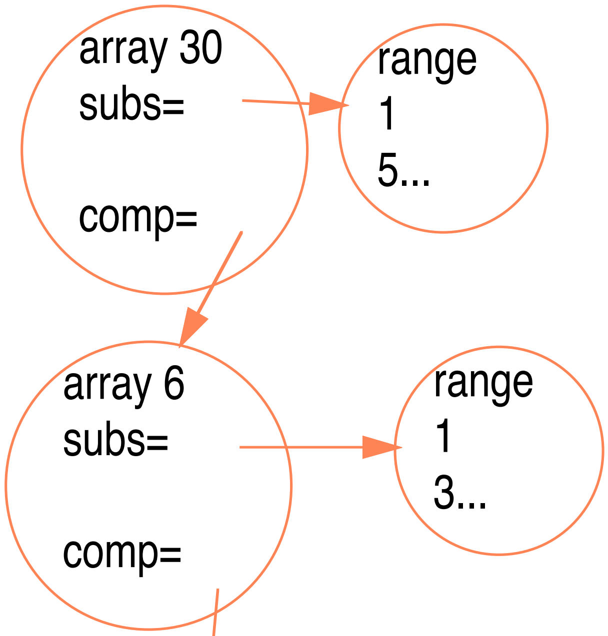
expr var
(0,1,true)

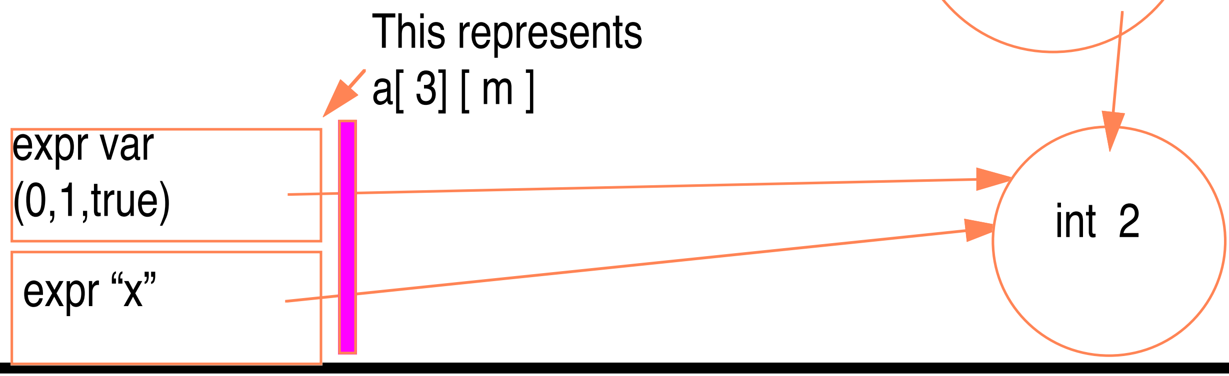
in Subscript(second call)

Subscript reduction

x := a[3] [m] ;•

```
LD      R0, #3
TRNG    R0, +..(R14)
IS      R0, #1
IM      R0, #6
LD      R1, +4(R15)
IA      R1, R0
LD      R0, +6(R15)
TRNG    R0, +..R14)
IS      R0, #1
IM      R0, #2
IA      R1, R0
```





Assign: exactly as before

Subscript reduction

Notes:

ReduceSubscript handles only one subscript. It is called multiple times if there are many.

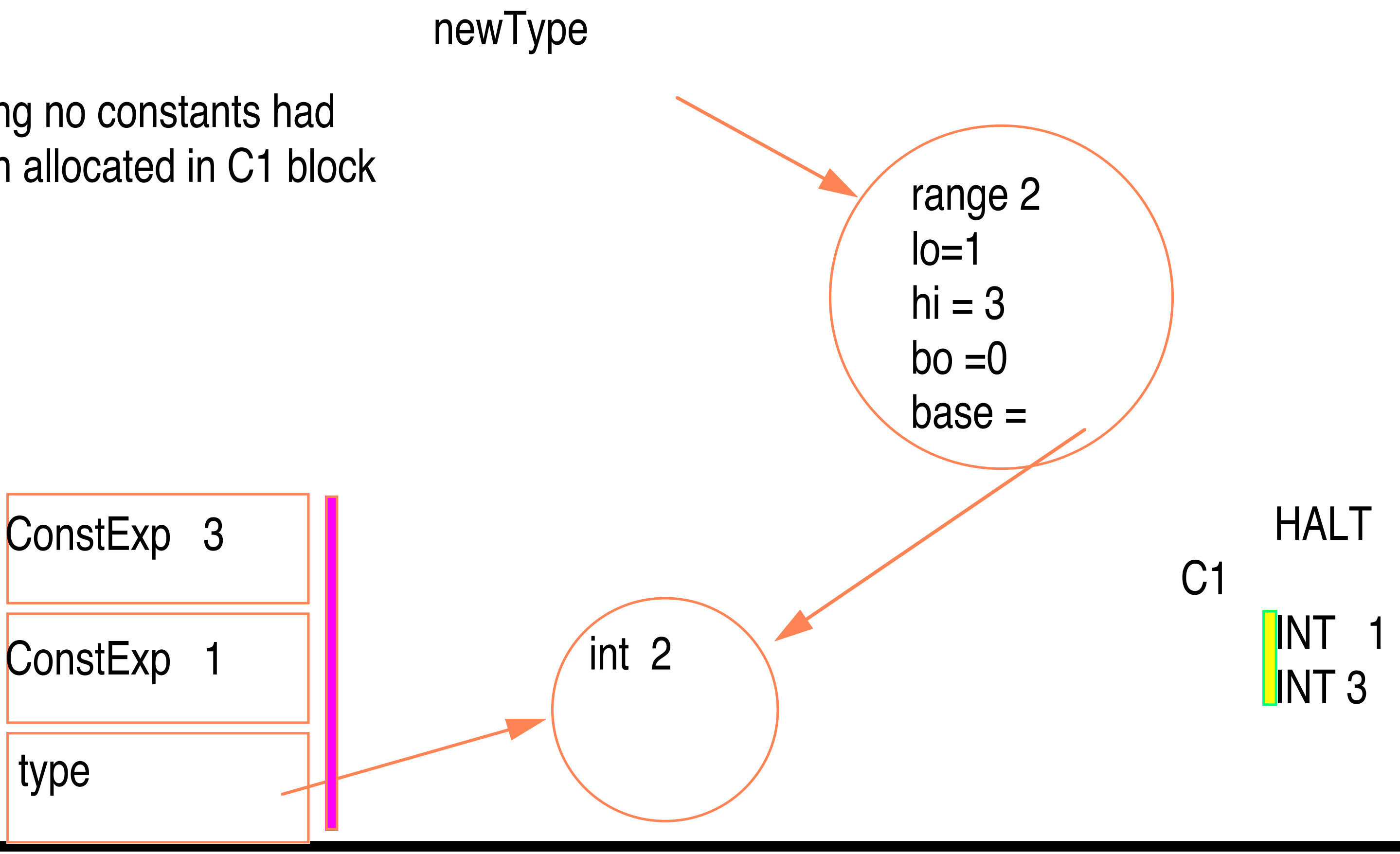
It needs to check its two entries for legality--top = integer type, next has array type.

Range Checking

integer range [1..3] one_three ;

Bounds offset for this array would be +0 _(R14)

Assuming no constants had yet been allocated in C1 block

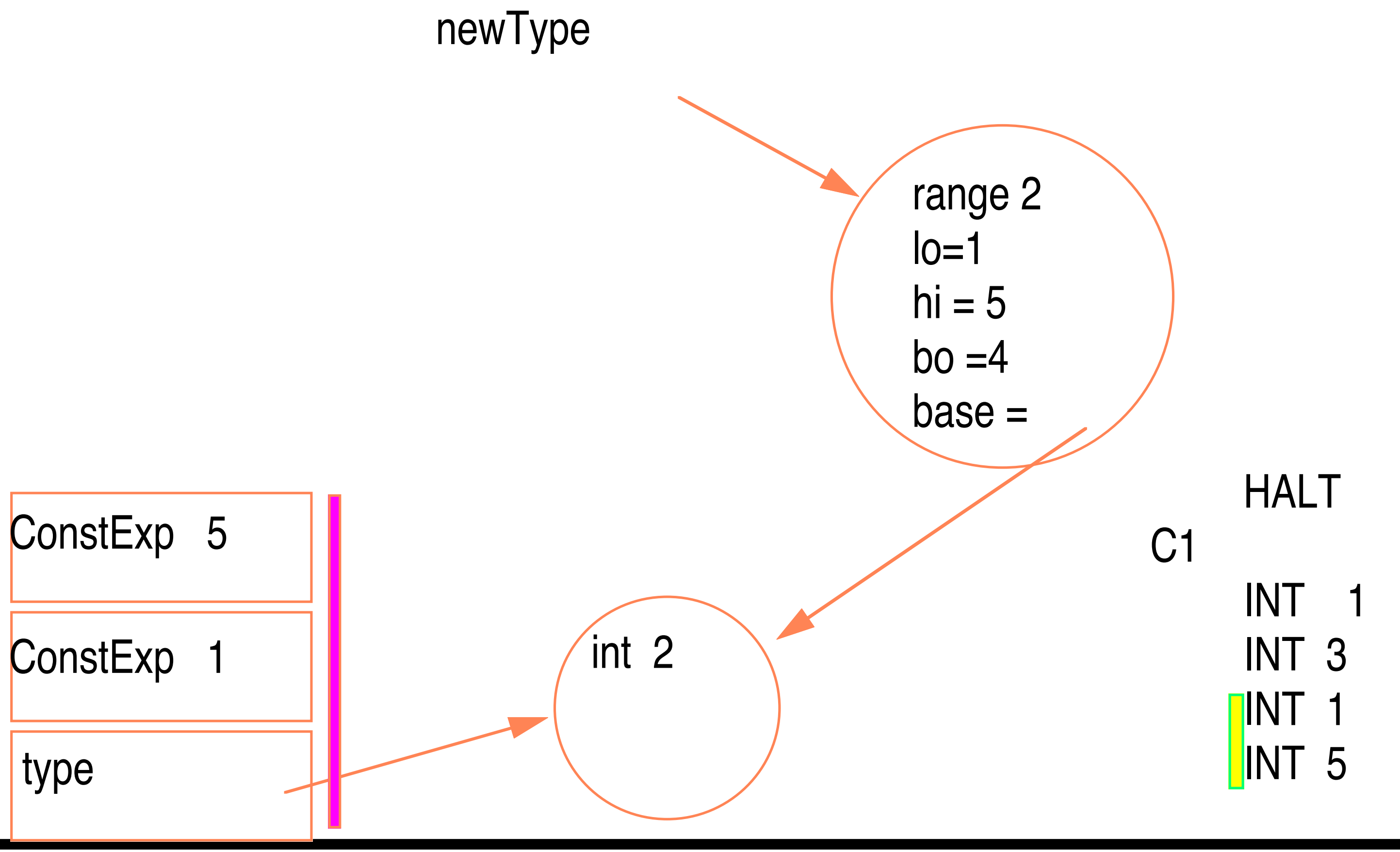


back in DoMakeRange

Range Checking

integer range [1..5] one_five ;

Bounds offset for this array would be + 4(R14)



In actuality, you enter two constants into the constant table, lobound and highbound