Principles of Programming Languages

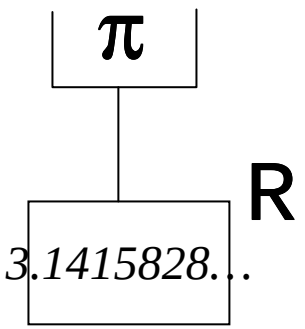
Lecture 10

Variables and Assignment

Variable ≠ Variable

(Math) (CS)

- **Math**

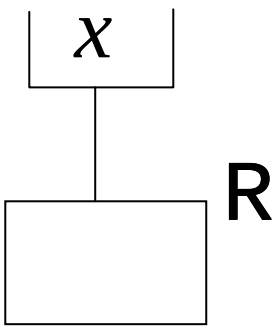


R

definite

singular

description



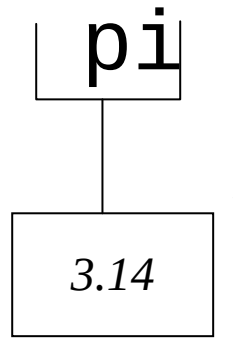
R

indefinite

singular

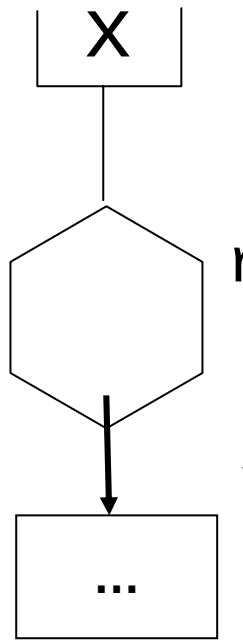
description

- **CS**



← **symbol**

← **referent**



← **symbol**

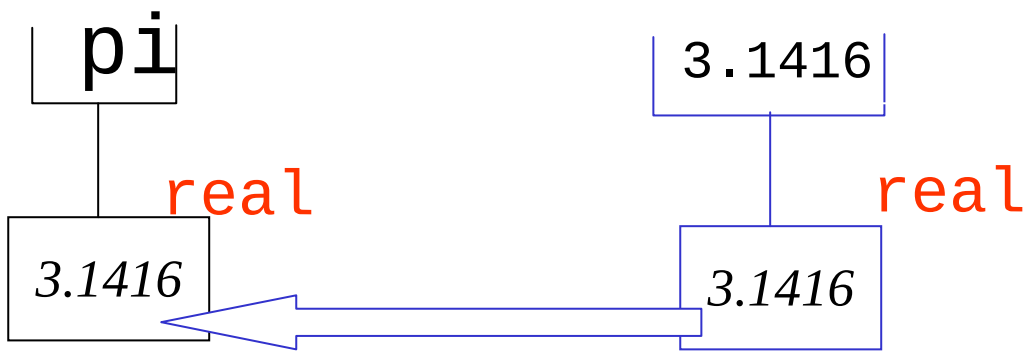
← **referent**

- **e.g. Prolog & FP (IFP)**

- **e.g. Pascal & Algol 68**

Constants

- Algol 68: **`real`** `pi = 3.1416;`
- C: (not quite) `#define PI 3.1416`
- Pascal: **`const`** `pi = 3.1416;`
- Ada: `pi:` **`constant`** `FLOAT := 3.1416;`

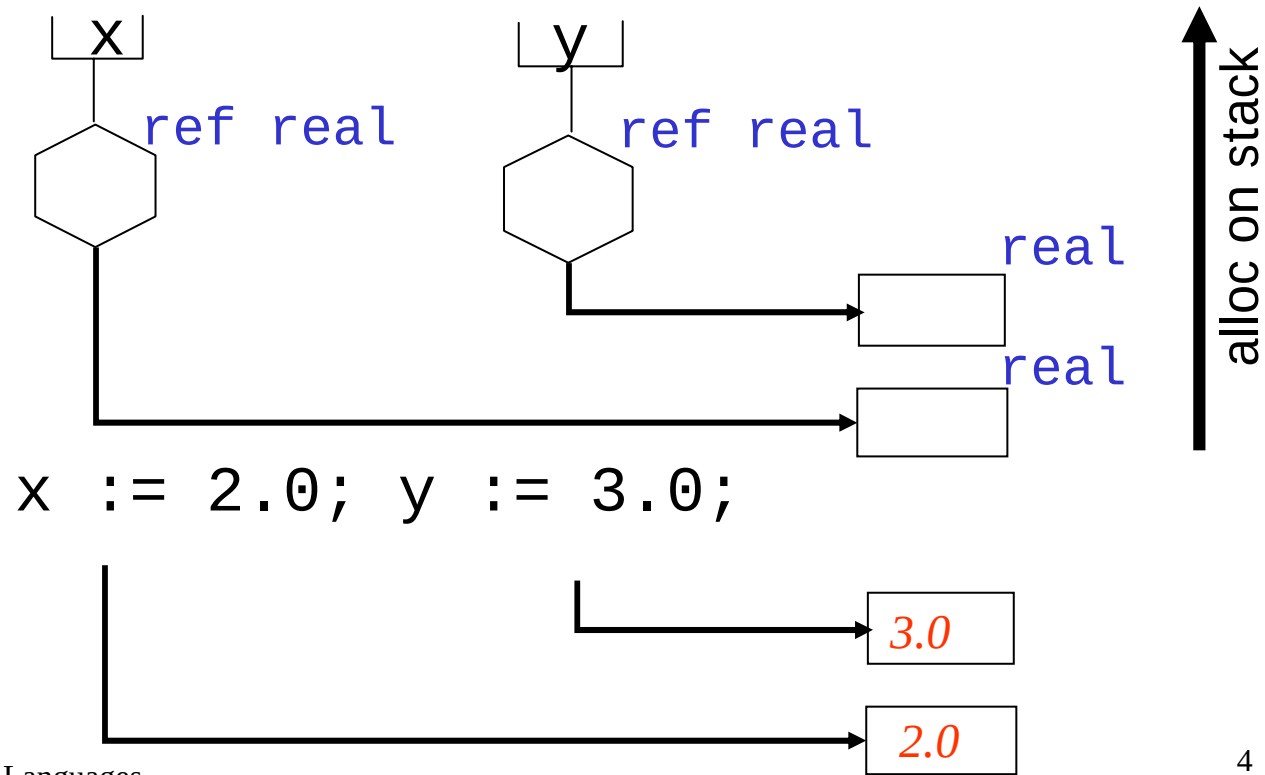


same logical status as a literal

- No reference available
- No `ref real` object exists
- `pi` replaced by literal
- No stack (or any other) space allocated (stored in instruction)
- *Pure definition*

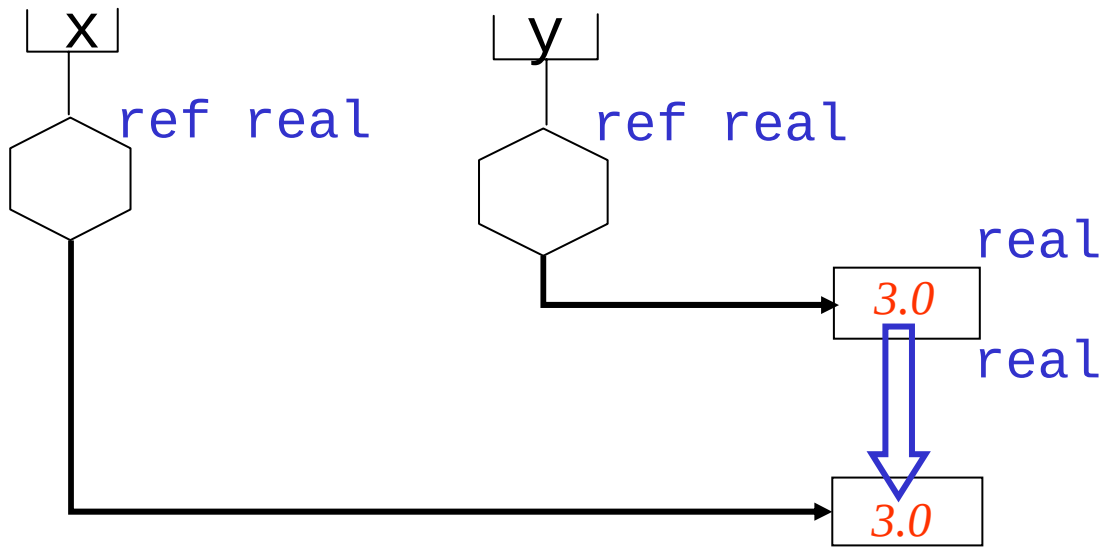
Variables

- Algol 68: **begin** real x,y;
- C: { float x,y;
- Pascal: **var** x,y: real;
- Ada: x,y: FLOAT;



Assignment (after y gets 3)

- Algol 68: `x := y;`
- C: `x = y;`
- Bliss: `x ← .y`



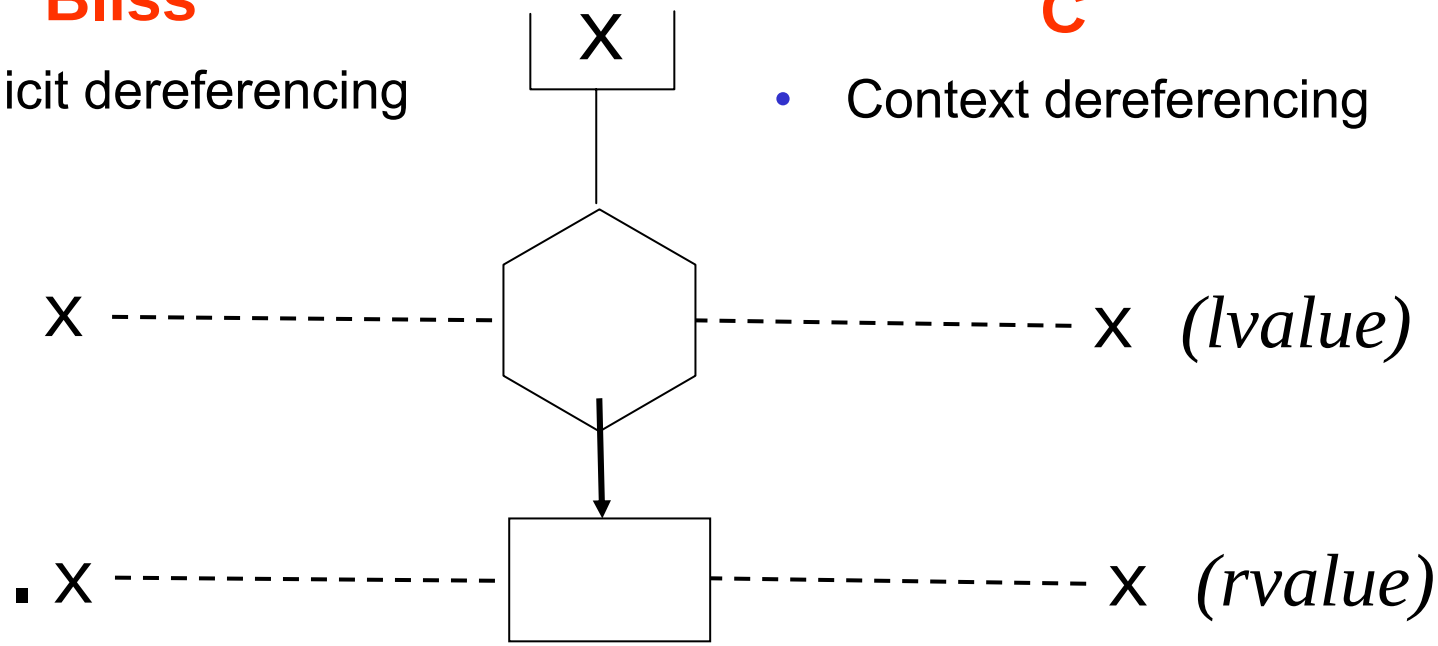
Variables: Bliss v. C

Bliss

- Explicit dereferencing

C

- Context dereferencing




```
x = .y + .z
a +.j ←.(a +.i)
```

```
x = y + z
a[j] = a[i]
```

lvalue *rvalue*

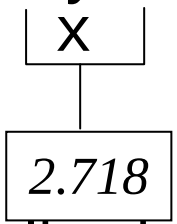
const

Synonym: `*(a+j) = *(a+i)`

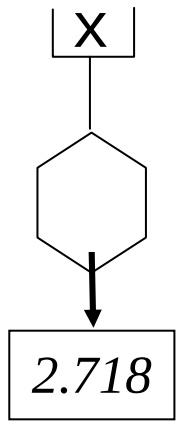
Only lvalue is actually computed

Algol 68 aside: identity declaration

- **`real`** `x := c` NOT same as **`real`** `x = c`
- Identity declaration **`real`** `x = 2.718;` means



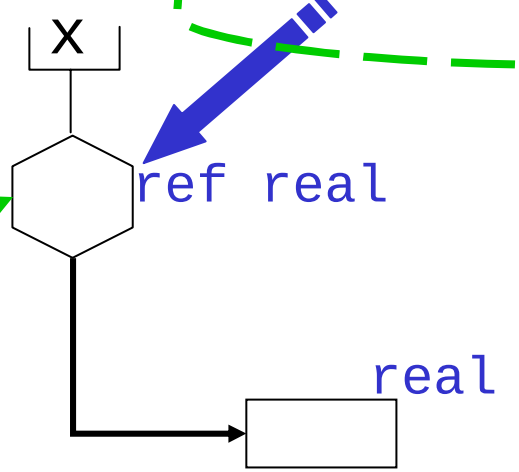
- Initialized variable **`real`** `x := 2.718;` means



- Why? Implicit *local* (stack) allocation

identity declaration (cont'd)

- **`real`** `x;` abbreviates **`loc real`** `x;` which abbreviates the following identity declaration:
- **`ref real`** `x =` **`loc real`** `;`

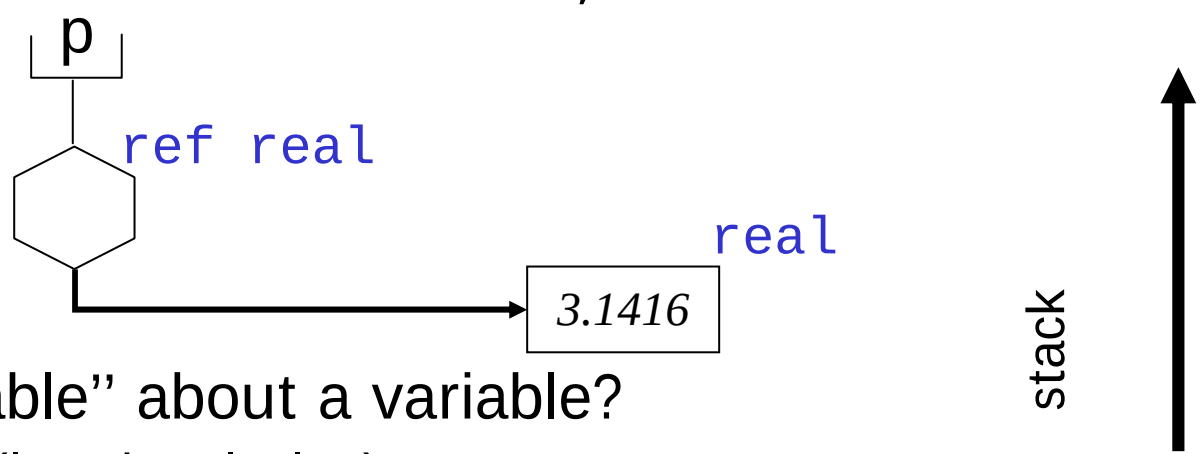


Local generator
Yields a `ref real`

`ref real` *(lvalue)*
Ascribed to identifier x

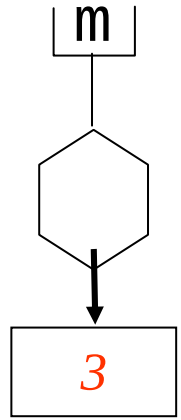
Initialized Variables

- Algol 68: `begin real p := 3.1416;`
- C: `{ float p = 3.1416;`
- Pascal: `var p: real; begin p := 3.1416;`
- Ada: `p: FLOAT := 3.1416;`

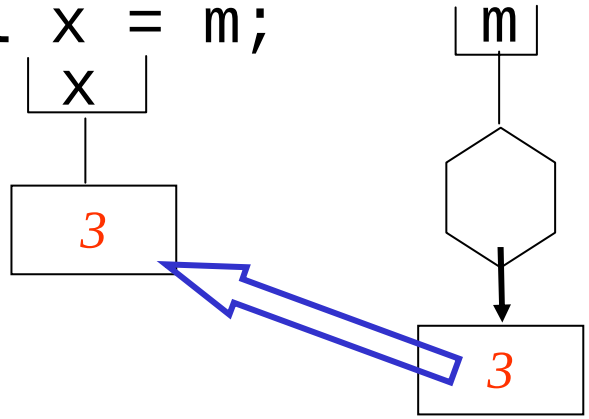


- What is ``variable'' about a variable?
  - Not reference (location, lvalue)
    - Ref cannot be changed; ``ascribed to'' or ``possessed by'' ident
  - Not identifier
  - Not type (in *most* languages)
  - The value (rvalue)

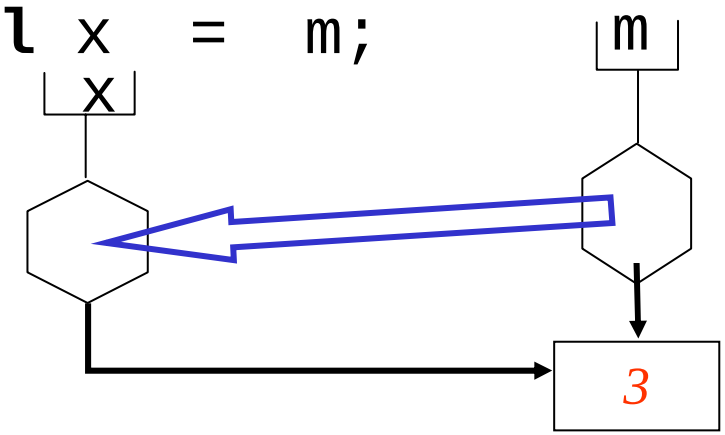
Algol 68 identity declaration: unifies variable & constant declaration



1. `loc real m := 3;`
2. `real x = m;`

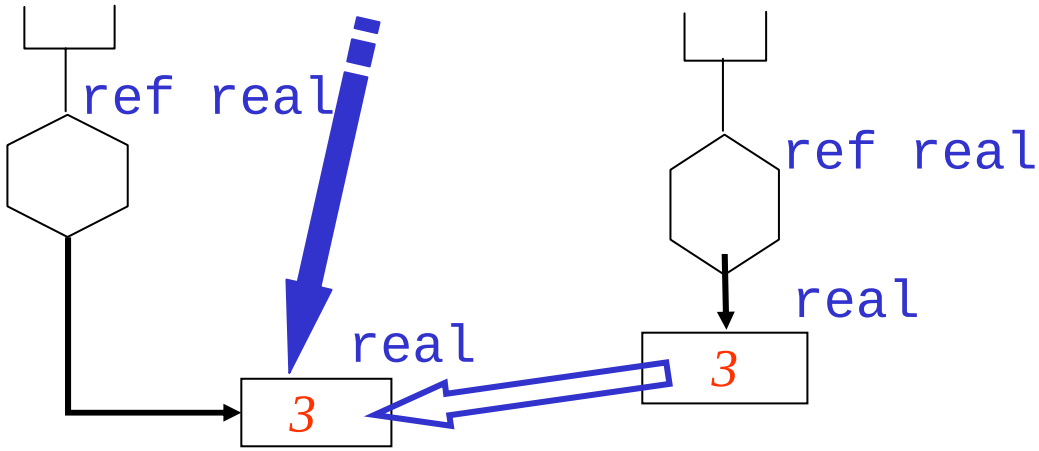


3. `ref real x = m;`

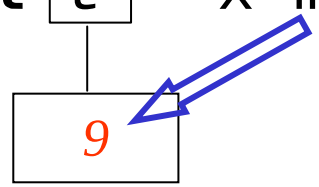


Algol 68 identity declaration (cont'd)

4. `real x := m;` ⇒ `loc real x:=m;` ⇒
`ref real x = loc real := m;`



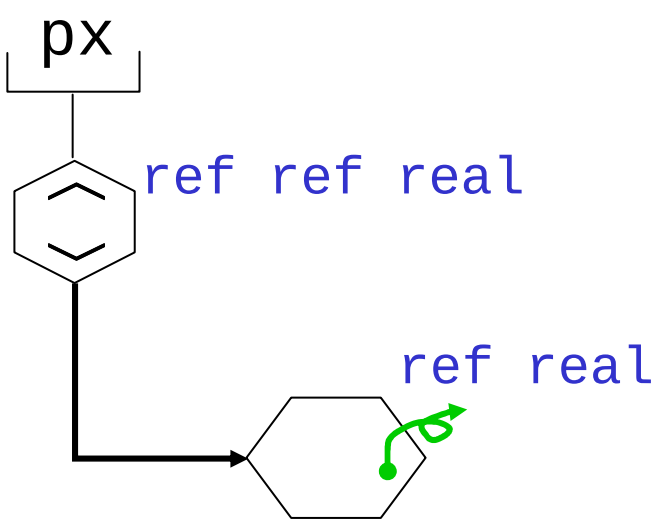
5. `real t = x*m;`



- **`t` const. In new environment**
- **`x*m` evaluated when declaration encountered**

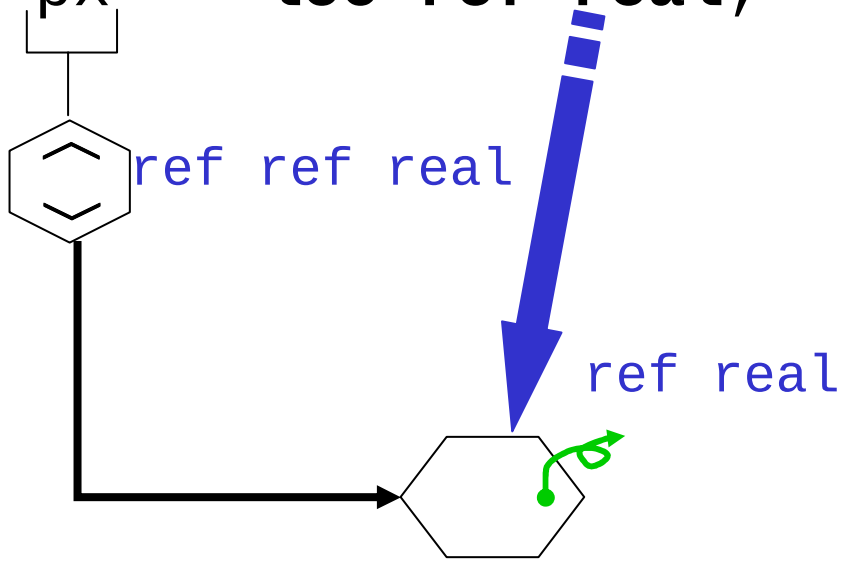
Pointers

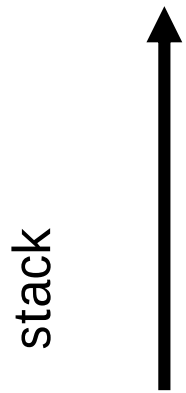
- Refs to variable refs
- Algol 68: `ref real px ;`
- C: `float *px ;`
- Pascal: `var px: ^real;`
- Ada: `type PTR is access FLOAT;`
  `px : PTR;` (PTR=``access type'' to ``base type'')



Algol 68 pointer declaration

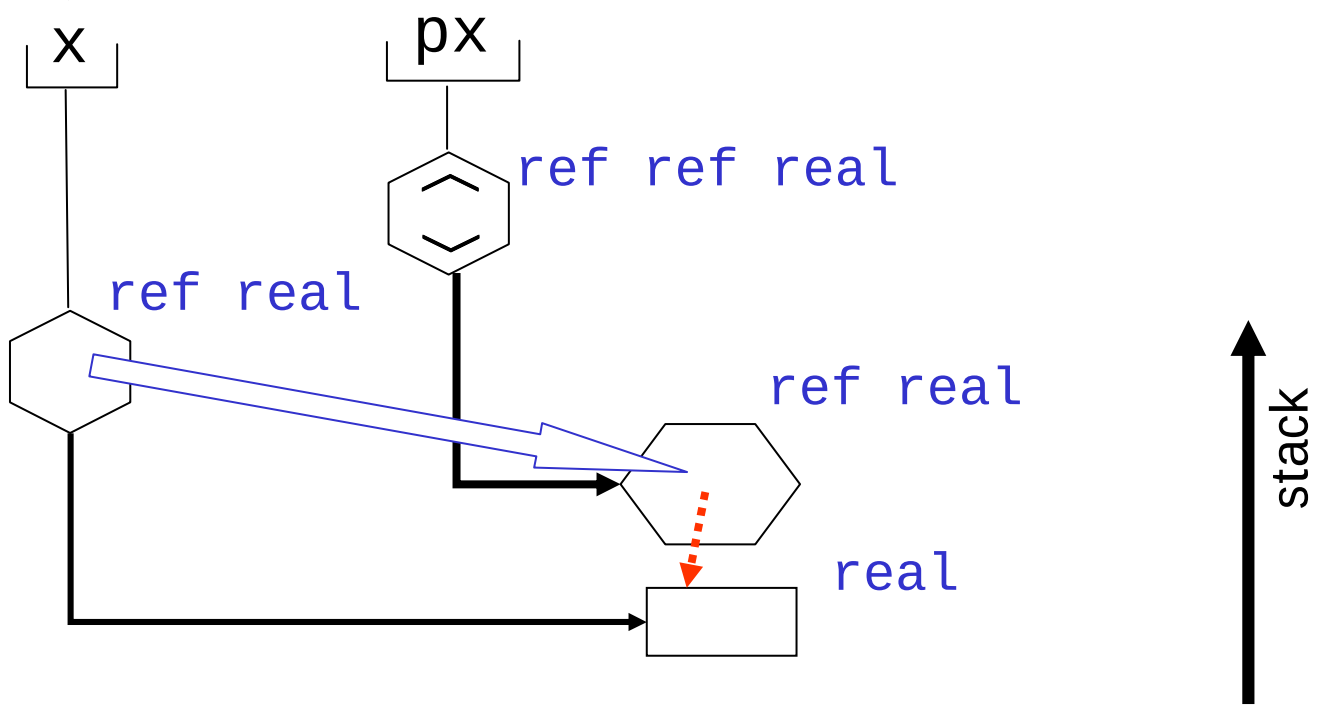
```
ref real px ⇒
 ref ref real px  = loc ref real;
```





Pointers (cont'd)

- Algol 68: `px := x ;`
- C: `px = &x ;`
- Bliss: `px ← x`

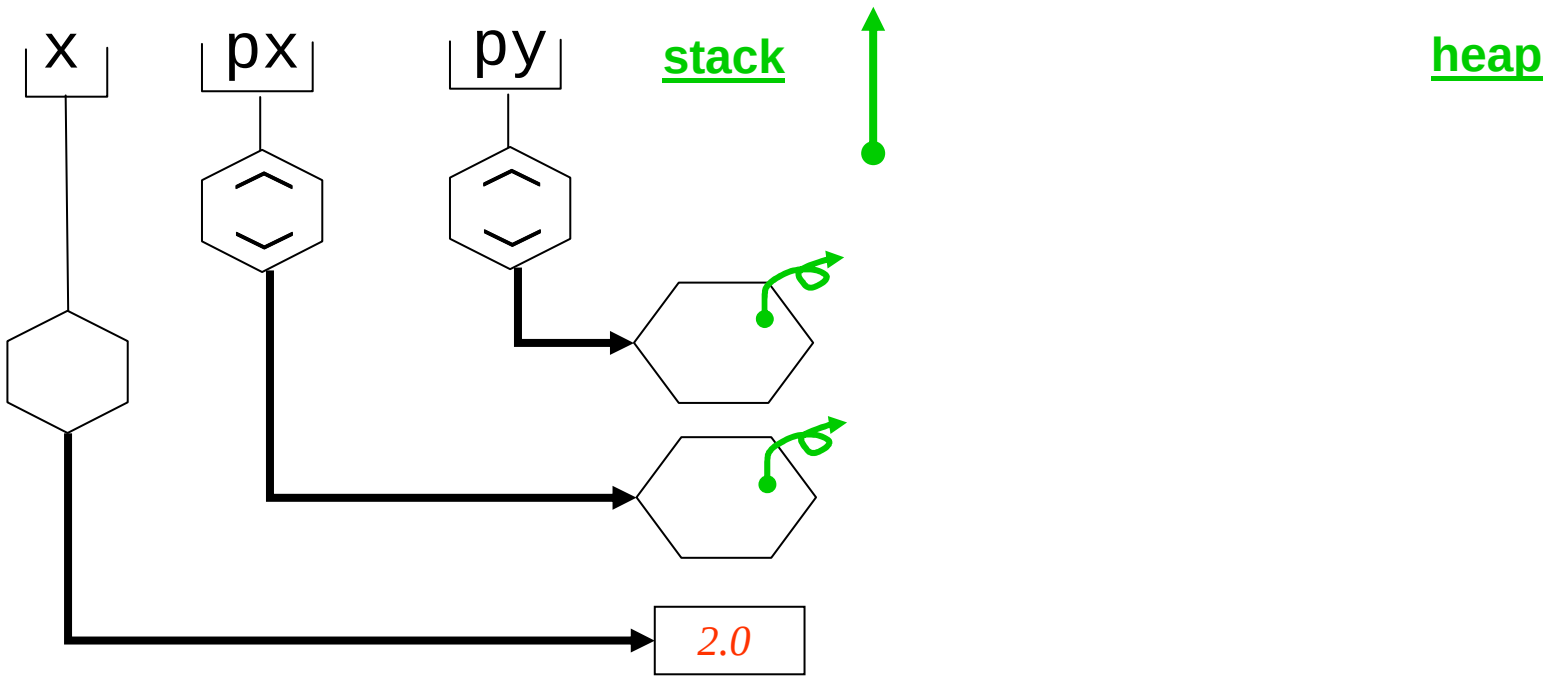


- NOT Pascal, Ada picture. Closest is:
  - Pascal: `new(px); px^ := x;` Ada: `px := new FLOAT(x);`

``anonymous variable''

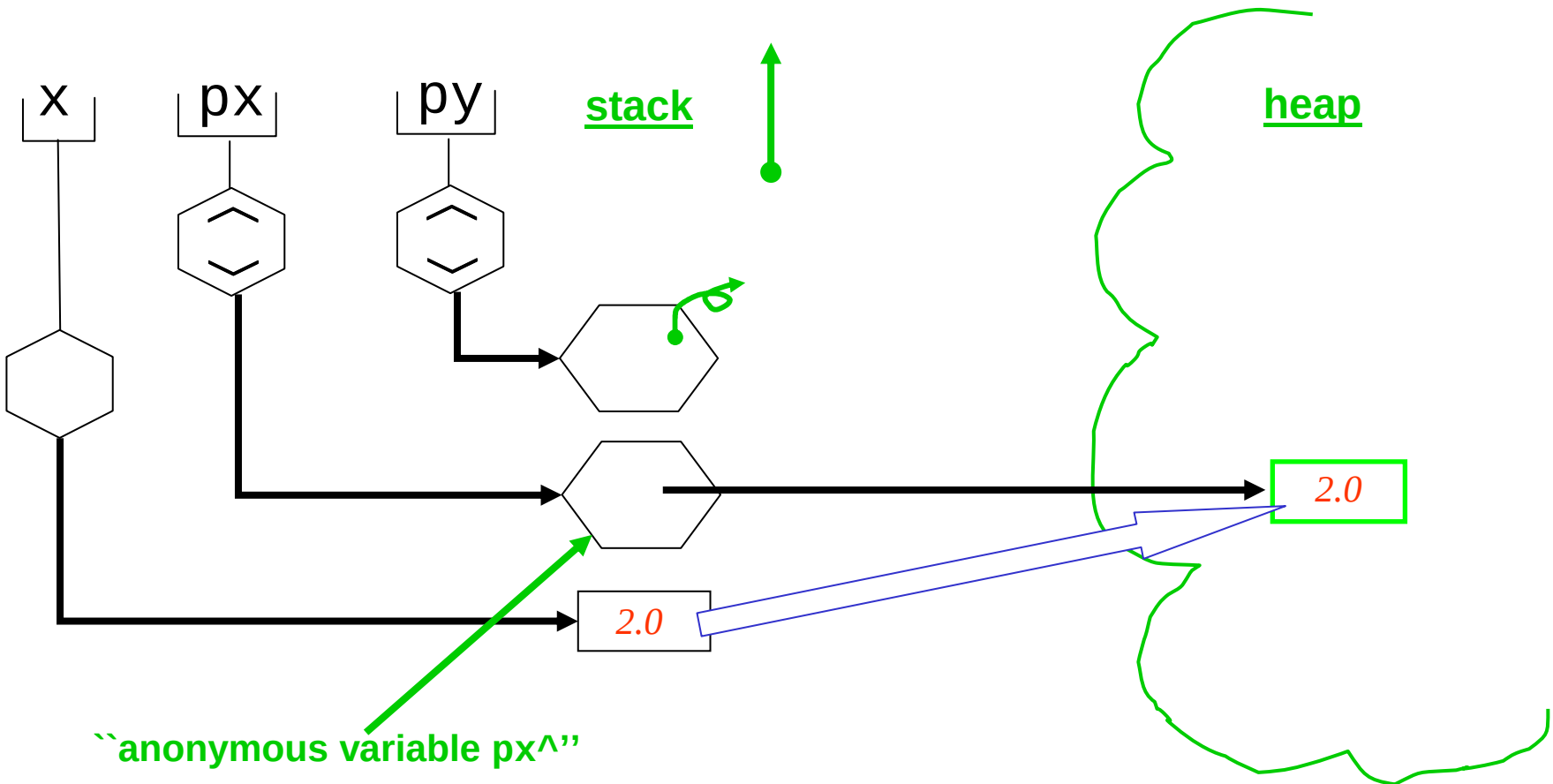
Pointers in Pascal/Ada: point into heap

- Defeats ``dangling pointers'' at block exit
- Pointers *cannot* point into the stack
- **`var`** `x: real;` **`var`** `px, py: ^real;`



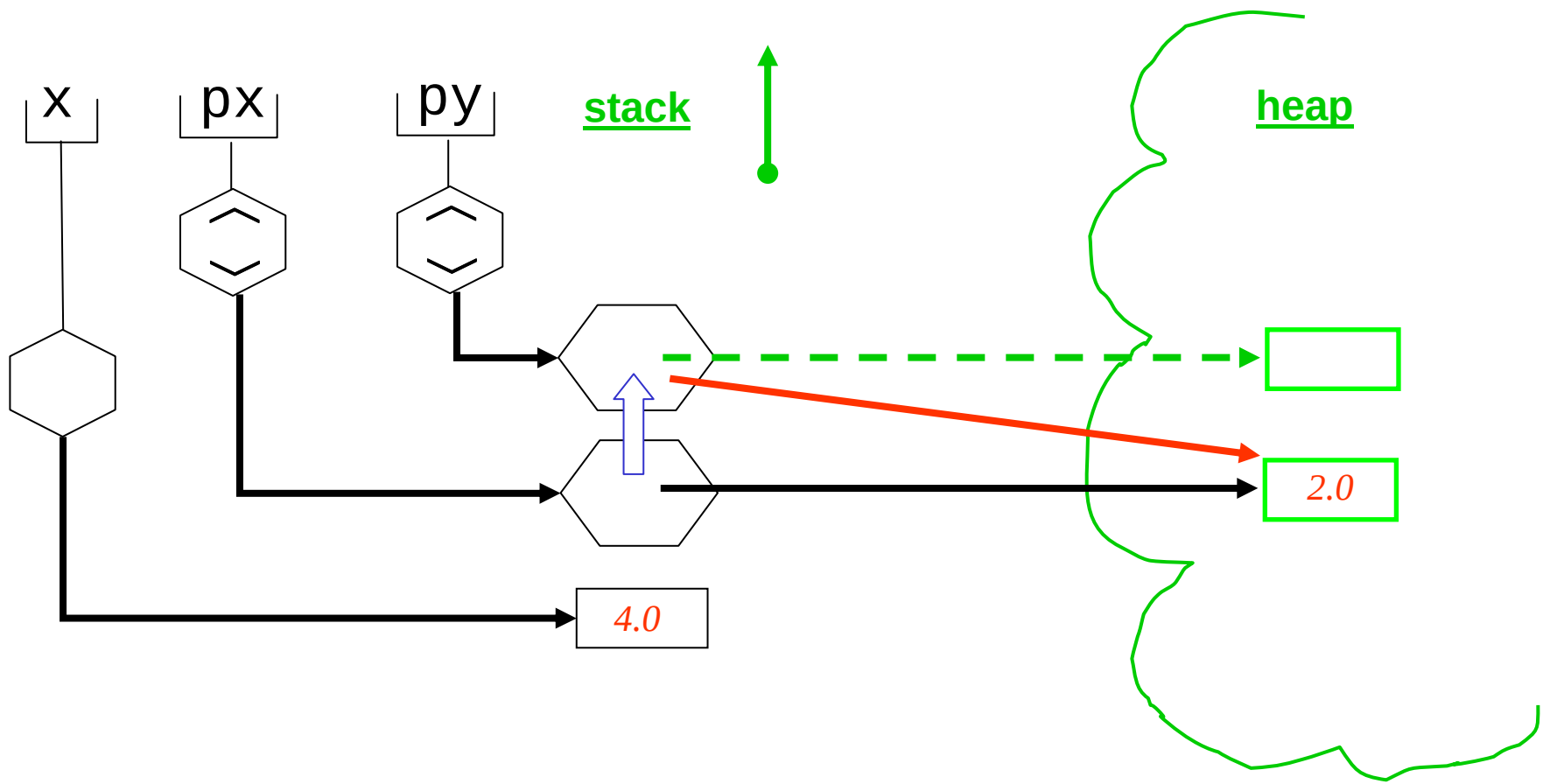
Pascal/Ada pointers (cont'd)

- `new(px);  px^ := x;`



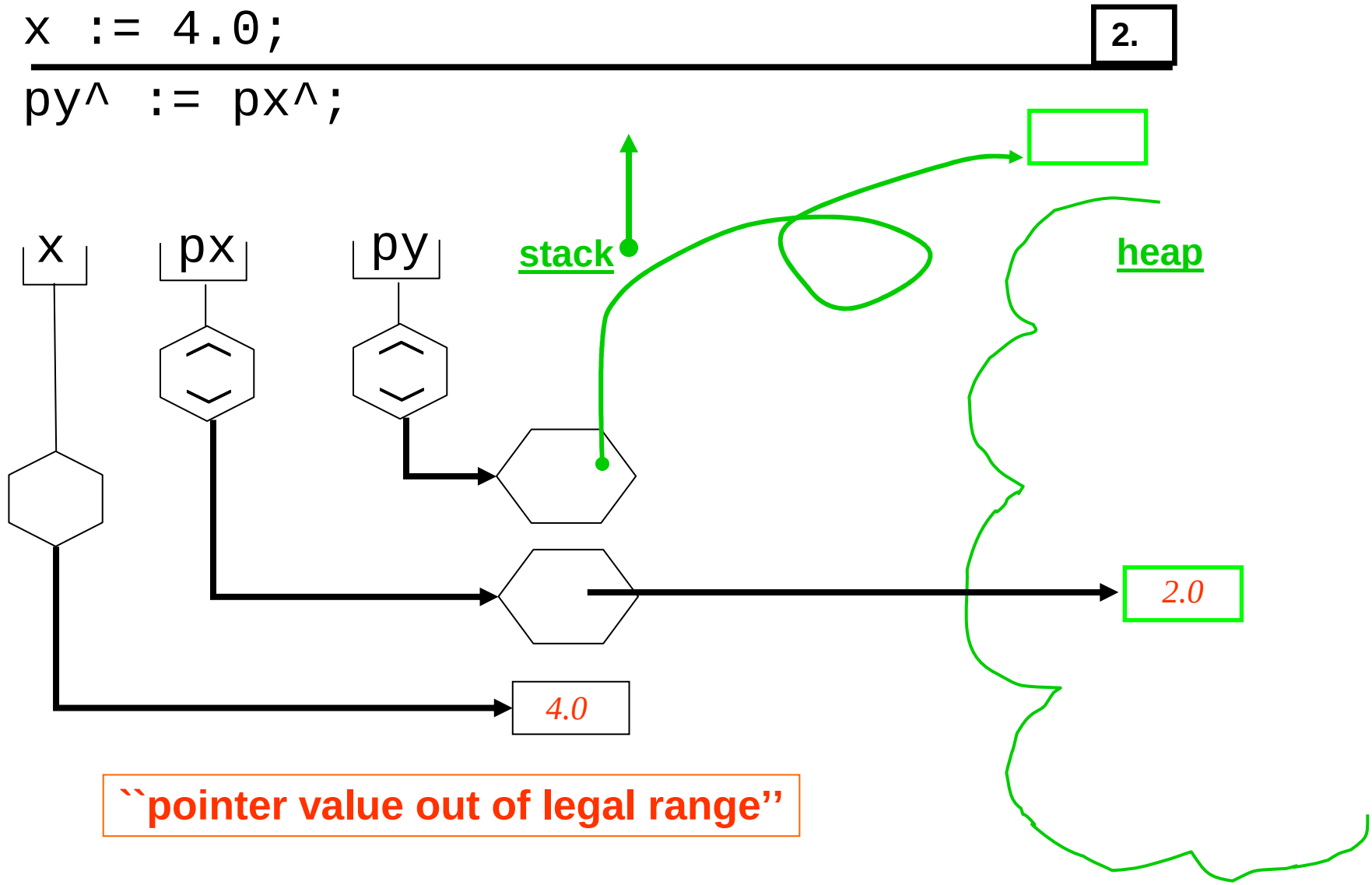
Pascal/Ada pointers (cont'd)

- `x := 4.0;`
- `new(py);  py := px;`



Pascal/Ada pointers (cont'd)

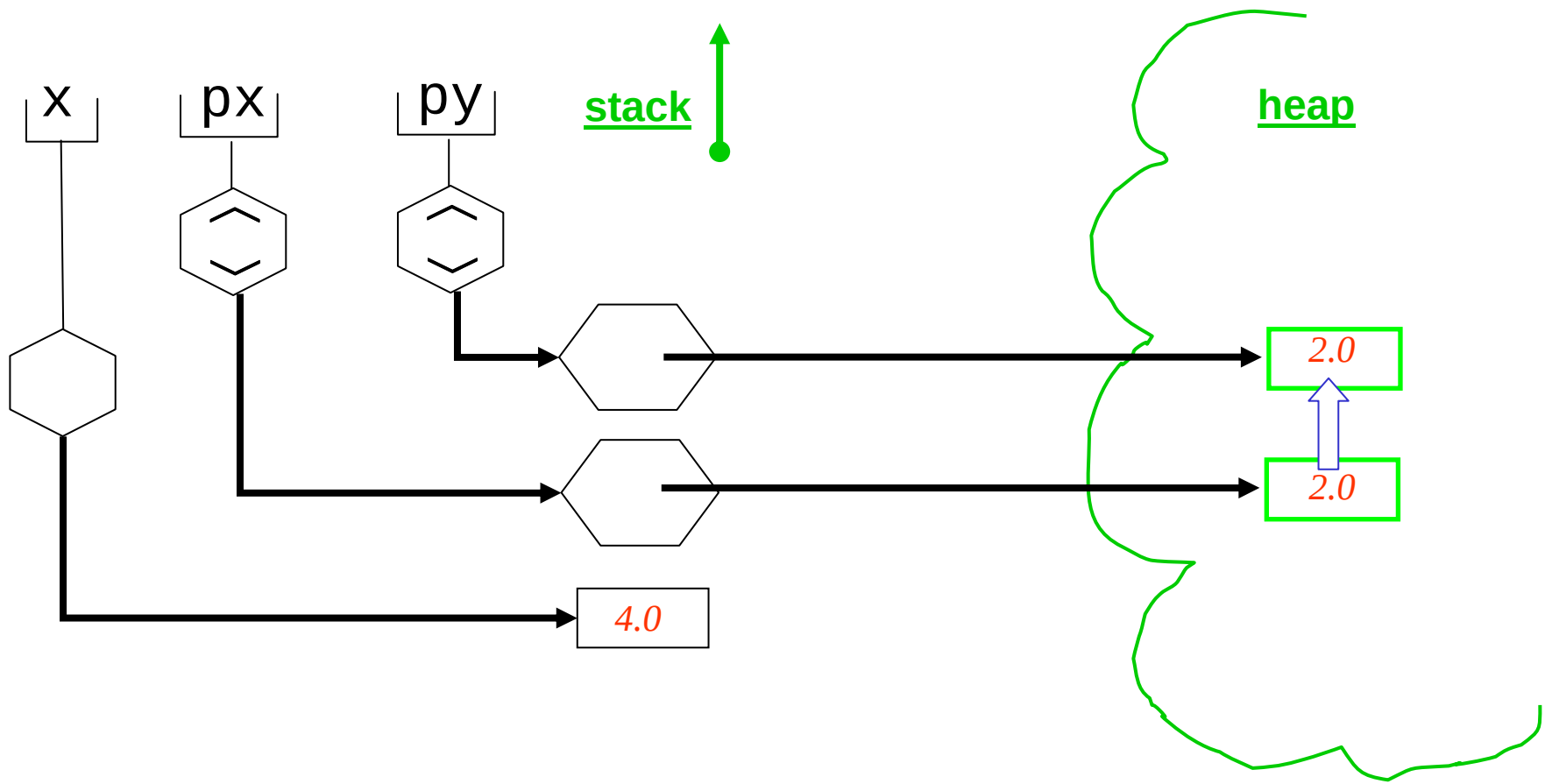
- `x := 4.0;`
- `py^ := px^;`



``pointer value out of legal range''

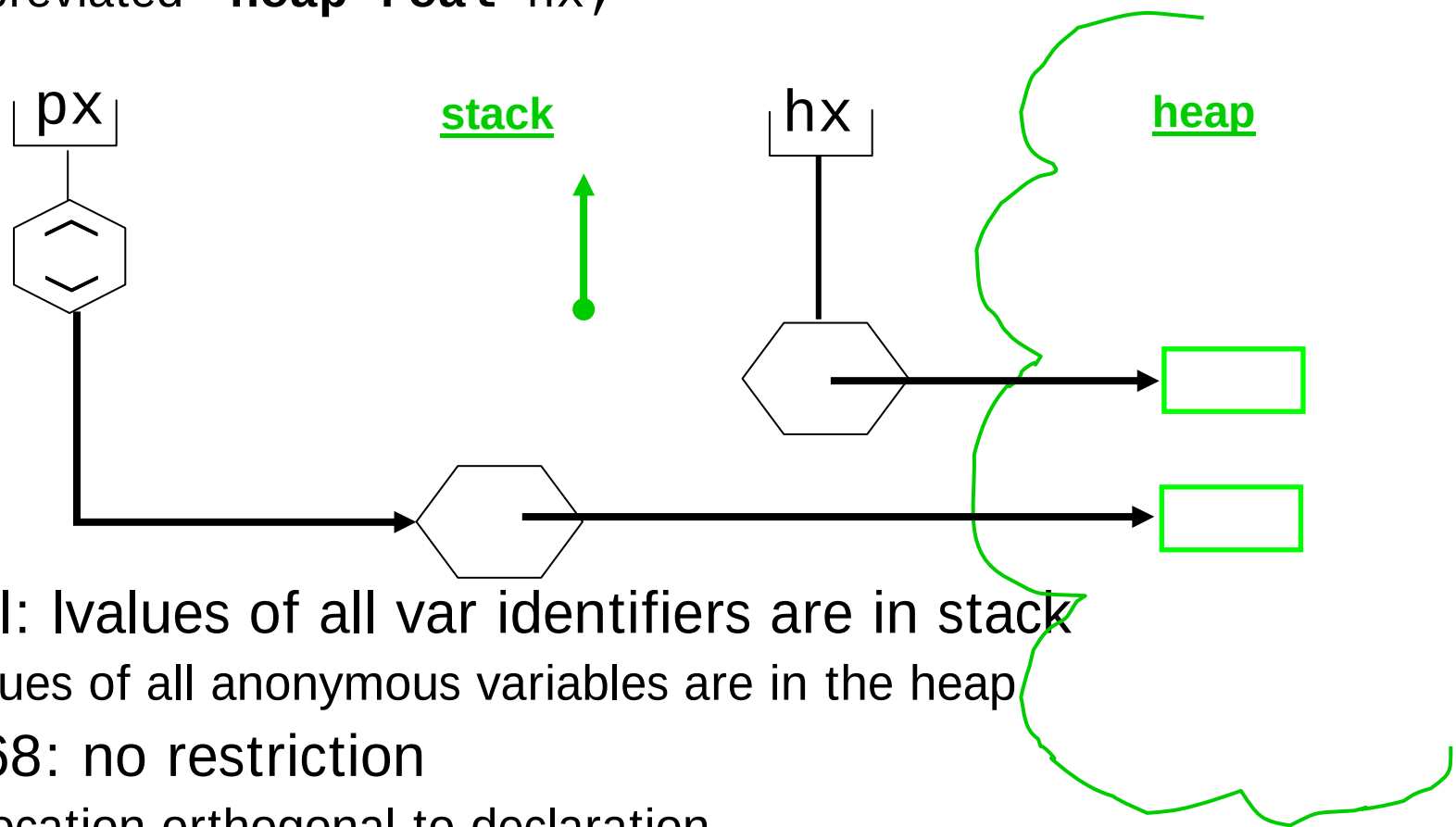
Pascal/Ada pointers (cont'd)

- `x := 4.0;`
- `new(py);  py^ := px^;`



Algol68 has a heap allocator

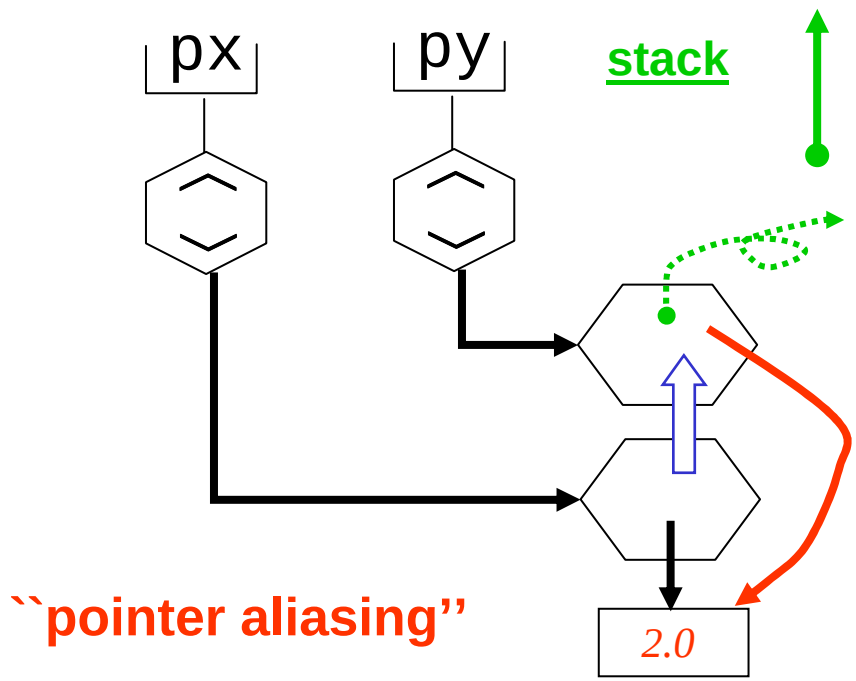
- **`ref ref real`** px = **`heap real;`**
- **`ref real`** hx = **`heap real;`**
  - Abbreviated **`heap real`** hx;

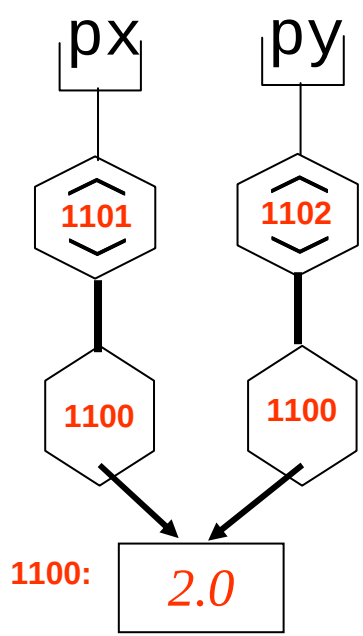


- Pascal: lvalues of all var identifiers are in stack
  - lvalues of all anonymous variables are in the heap
- Algol68: no restriction
  - allocation orthogonal to declaration

Refs and Aliasing

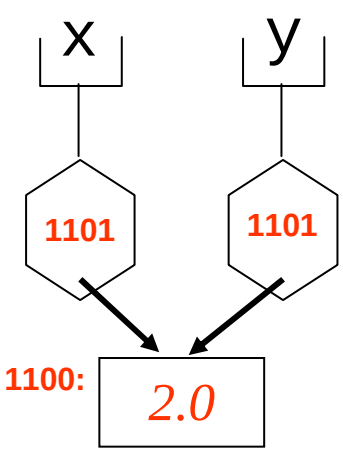
- Algol68: **`ref real`** `py; py := px;`

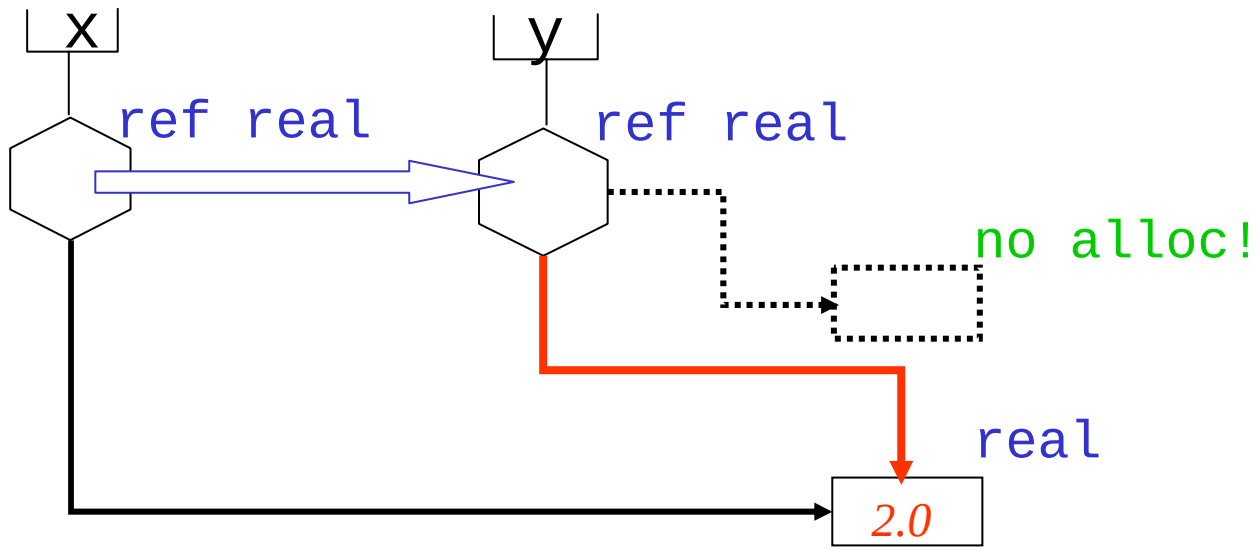




Refs and Aliasing (cont'd)

- Algol 68: **`ref real`** `y = x;` (identity declaration)

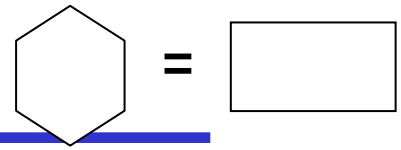




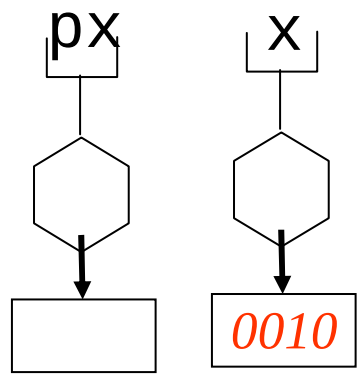
``name aliasing''

``equivalencing (Fortran)''

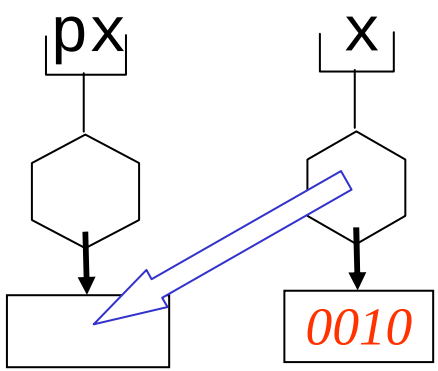
Bliss: untyped bit strings



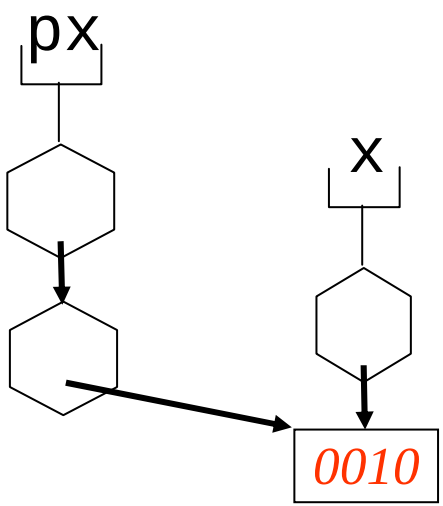
- **local** x; x ← 2; **local** px;



- px ← x



``ptr to x stored in px''

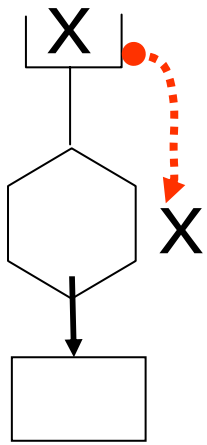


picture after assignment

Semantics of Variable Uses: C v. Bliss

- On LHS

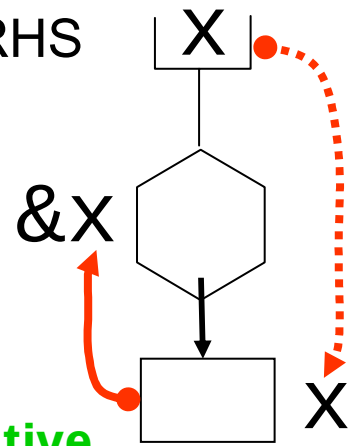
C:



= / ←

=

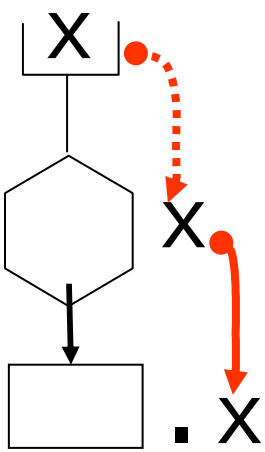
- On RHS



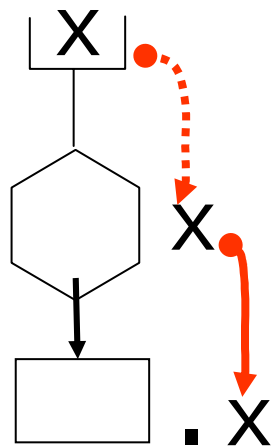
Meaning of x **context-sensitive**

Not ``referentially transparent''

Bliss:



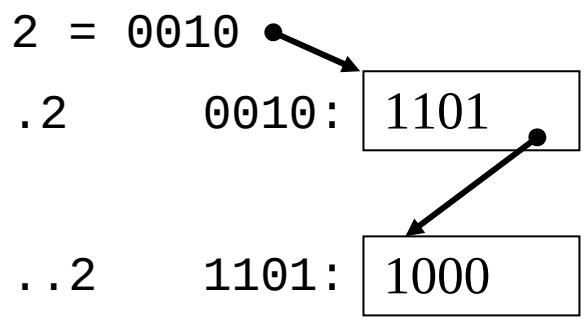
←



Meaning of x **is** ``referentially transparent''

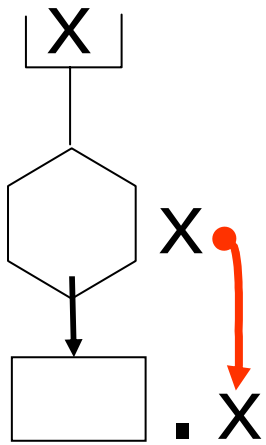
Variable Uses: C v. Bliss (cont'd)

- **C:** `&` **not** a unary operator
  - `&(x+y)` meaningless
  - `&&x` meaningless (can't reverse a pointer)
  - `&2` meaningless
- **Bliss:** `.` **is** a unary operator
  - `.(x+y)` sensible
  - `. .x` sensible
  - `.2` sensible

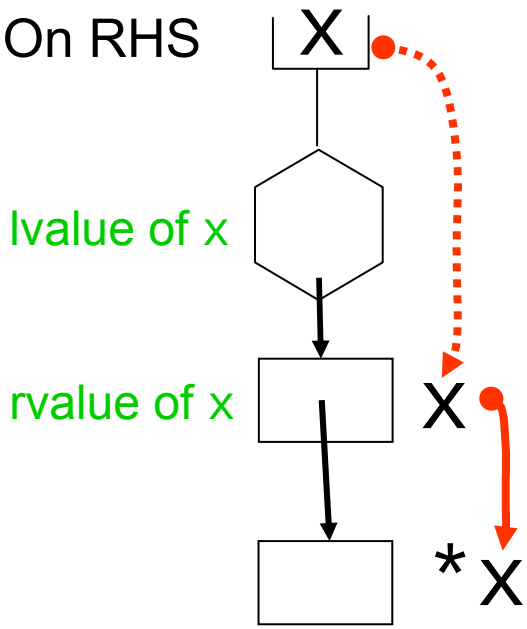


C * ≠ Bliss.

Bliss:



C: • On RHS



Bliss variable occurrences are ``referentially transparent''
Semantics of `x is independent of context`