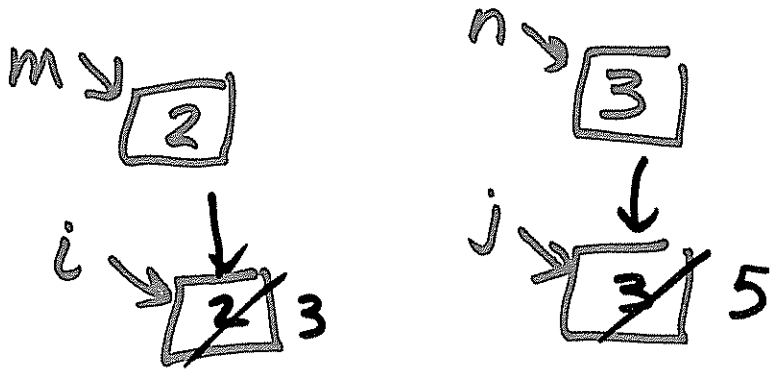
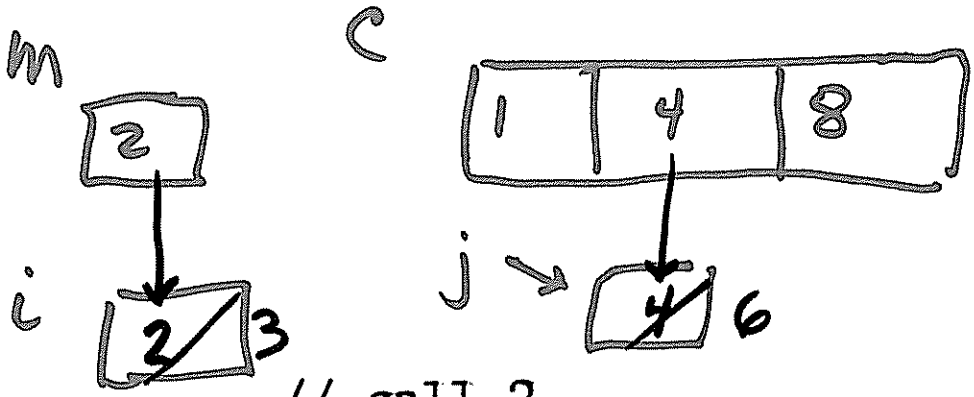
Pass By Value

Example for Passing Modes

```
{ c : array[1..10] of integer;
  m,n integer;
  procedure r (i , j : integer ) begin
      i := i + 1;
      j := j + 2
    end r;
  ...
  m := 2;
  n := 3;
  r(m,n);                  // call 1
  write m, n ;             // print 1

  m := 2;
  c[1] := 1;
  c[2] := 4;
  c[3] := 8;
  r(m,c[m]);               // call 2
  write m,c[1],c[2],c[3];  // print 2
}
```



2 3



2 1 4 8

Pass by Value

- Initial values of parameters copied from current values of arguments
- Final values of parameters are "lost" at return time (like local variables).
- Example:

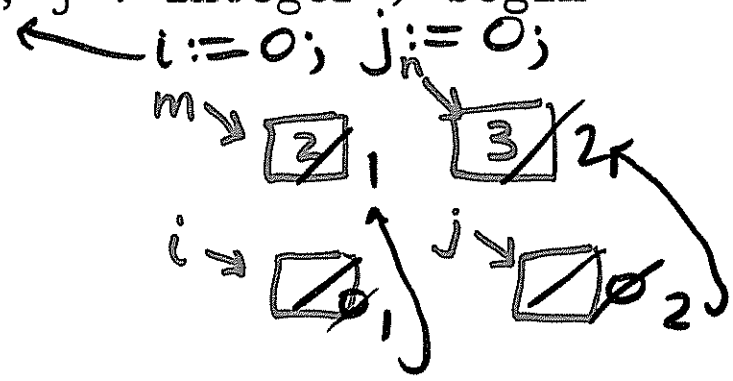
```
at call 1:  i = 2  j = 3
print 1:    2, 3
at call 2:  i = 2  j = 4
print 2:    2, 1 4 8
```

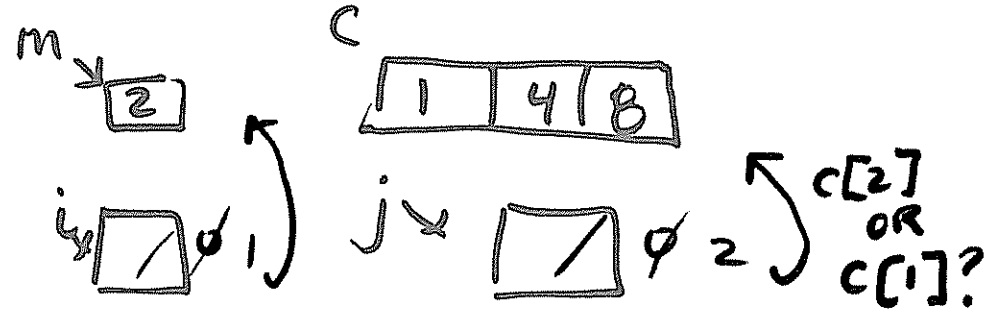
- Benefit: Arguments protected from changes in procedure.
- Problem: Requires copying of values: costs time and space, especially for large aggregates.

PASS BY RESULT

Example for Passing Modes

```
{ c : array[1..10] of integer;
  m,n integer;
  procedure r (i , j : integer ) begin
      i := i + 1;
      j := j + 2
    end r;
  ...
  m := 2;
  n := 3;
  r(m,n);                  // call 1
  write m, n ;             // print 1

  m := 2;
  c[1] := 1;
  c[2] := 4;
  c[3] := 8;
  r(m,c[m]);               // call 2
  write m,c[1],c[2],c[3];  // print 2
}
```





c[1] updated

c[2] updated

Pass by Result (Example)

Suppose proc `r` initializes `i` and `j` to 0:

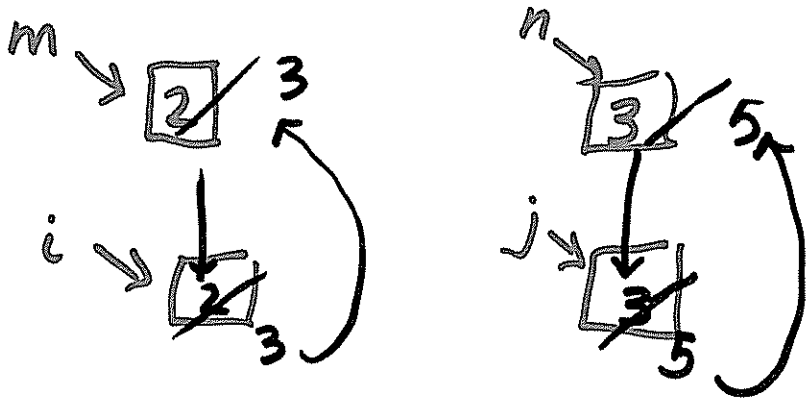
- `call 1:`
  - final values of `i` and `j`: 1 2
  - `m` and `n` are set to: 1 2
- `print 1:` 1 2
- `call 2:` more problematic
  - final values of `i` and `j`: 1 2
  - which element of `c` is modified, `c[1]` or `c[2]`?
- `print 2:`
  - If `c[1]` is modified:
  - If `c[2]` is modified:

m	c[1]	c[2]	c[3]
1	2	4	8
1	1	2	8

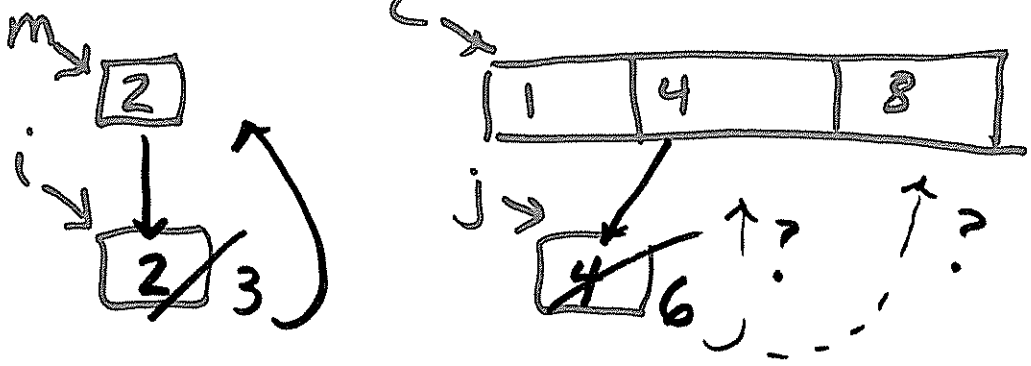
PASS BY "VALUE-RESULT"
PASS BY "COPY"

Example for Passing Modes

```
{ c : array[1..10] of integer;
  m,n integer;
  procedure r (i , j : integer ) begin
      i := i + 1;
      j := j + 2
    end r;
  ...
  m := 2;
  n := 3;
  r(m,n);                  // call 1
  write m, n ;             // print 1

  m := 2;
  c[1] := 1;
  c[2] := 4;
  c[3] := 8;
  r(m,c[m]);               // call 2
  write m,c[1],c[2],c[3];  // print 2
}
```



3 5



if c[3] mod 3 → 1 4 6

if c[2] mod 3 → 1 6 8

Pass by Value-Result (Example)

- `call 1:`
  - initial: i = 2 j = 3
  - final: i = 3 j = ~~5~~
  - return: `m` and `n` set to: 3 ~~5~~
- `print 1:` 3 ~~5~~
- `call 2:`
  - initial: i = 2 j = 4
  - final: i = 3 j = 6
  - return: which element of c is modified, c[2] or c[3]? Depends

    ~~Remember~~
- `print 2:`
  - if c[2] is modified: 3
  - if c[3] is modified:

m c[1] c[2] c[3]

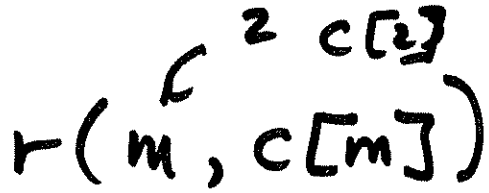
if c[2] is mod 3 1 4 8

if c[3] is mod 3 1 4 6

Further Specifying Pass by Result

With pass by result or pass by value-result, order of assignments and address computations is important.

Options:



1. Perform return address computations at call time:

   On second return:

   `m` set to 3; `c[2]` set to 6

   `print 2:` 3 1 6 8

Further Specifying Pass by Result (cont'd)

2. Perform return address computations at return time:

   (a) Before any assignments:

   On second return: same as above, but might not be if procedure has side-effects

   (b) Just before that assignment, in order:

   On second return:

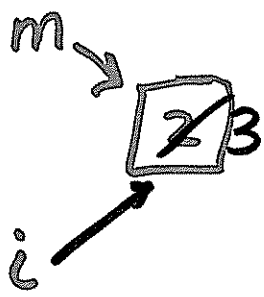
   `m` set to 3; `c[3]` set to 6

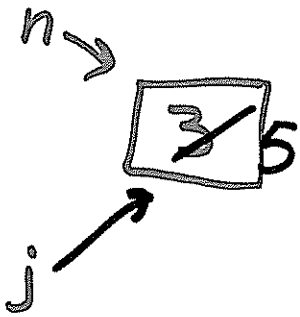
   `print 2:` 3 1 4 6

PASS BY REFERENCE

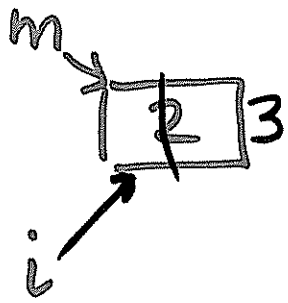
Example for Passing Modes

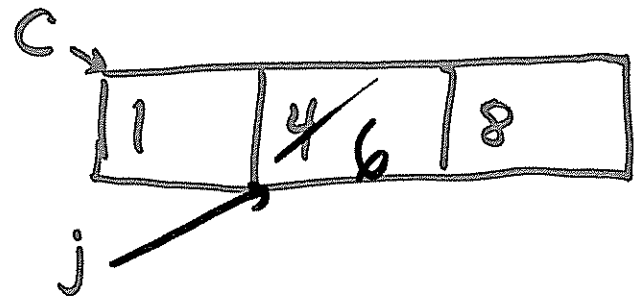
```
{ c : array[1..10] of integer;
  m,n integer;
  procedure r (i , j : integer ) begin
      i := i + 1;
      j := j + 2
    end r;
  ...
  m := 2;
  n := 3;
  r(m,n);                  // call 1
  write m, n ;             // print 1

  m := 2;
  c[1] := 1;
  c[2] := 4;
  c[3] := 8;
  r(m,c[m]);               // call 2
  write m,c[1],c[2],c[3];  // print 2
}
```





3 5





3 1 6 8

Pass by Reference (Example)

- `call 1:`
  - initial: i = 2 j = 3
  - final: i = 3 j = 5
  - return: `m`, `n` are: 3 5
- `print 1:` 3 5
- `call 2:` m: 2; c[m]: c[2] = 4
  - initial: i = 2 j = 4
  - final: i = 3 j = 6
  - return: `m`, `c[2]` are: 3 6
- `print 2:` m: 2; c[1]: 1; c[2]: 4; c[3]: 3

  3 1 6 8

PASS BY NAME

Example for Passing Modes

```
{ c : array[1..10] of integer;
  m,n integer;
  procedure r (i , j : integer ) begin
      i := i + 1;
      j := j + 2
    end r;
  ...
  m := 2;
  n := 3,
  r(m,n);                  // call 1
  write m, n ;             // print 1

  m := 2;
  c[1] := 1;
  c[2] := 4;
  c[3] := 8;
  r(m,c[m]);               // call 2
  write m,c[1],c[2],c[3];  // print 2
}
```

m := m + 1
n := n + 2

m := 2 + 1
n := 3 + 2

3 5

m := m + 1
c[m] := c[m] + 2

m := 2 + 1
c[3] := c[3] + 2
10 + 2

3 1 4 10

Pass by Name (Example)

- Example:
  - `call 1: m, n set to:`
  - `print 1:` 3 5
  - `call 2: m, c[m] set to:`
  - `print 2:`

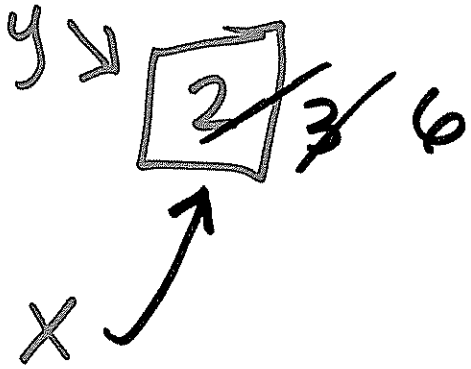
m	c[1]	c[2]	c[3]	c[4]
2	1	4	8	
3	1	4	10	

- Benefit: same as pass by reference
- Problems: Inefficient, requires a *thunk*:
  - essentially a little program is passed that represents the argument
  - evaluates argument in caller's environment

Aliasing

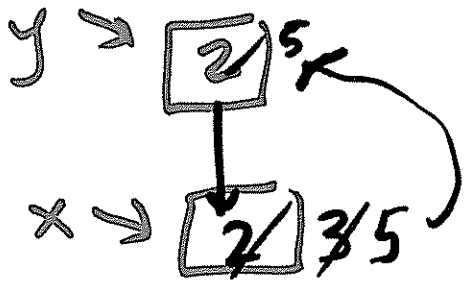
```
{ y : integer ;
  procedure p ( x : integer ) begin
    x := x + 1;
    x := x + y
  end p;
  ...
  y := 2;
  p(y);
  write y
}
```

pass by reference



6

PASS BY VALUE-RESULT (In contrast)

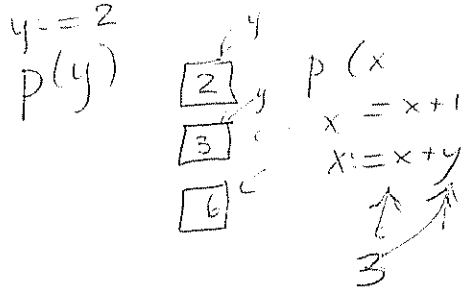


WRITE Y ⇒ 5

Aliasing

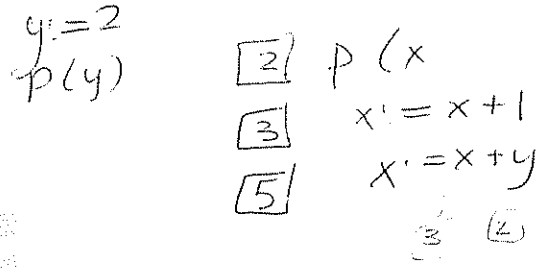
Pass by Reference:

- The identifiers `x` and `y` refer to the same location in call of `p`.
- Result of "`write y`"? 6



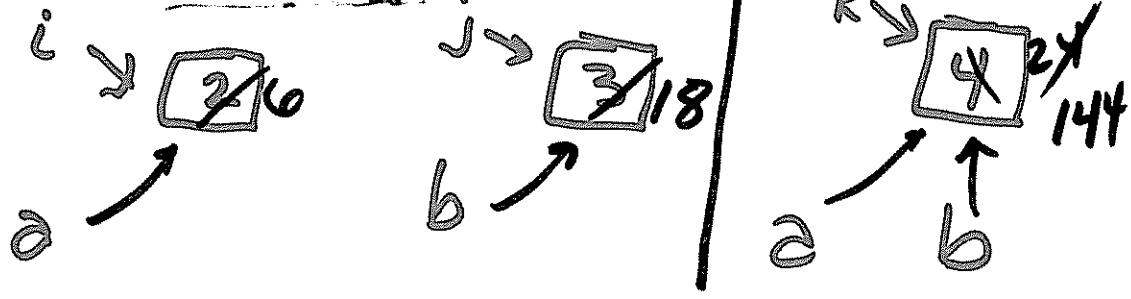
Pass by Value-Result:

- The identifiers `x` and `y` refer to different locations in call of `p`.
- Result of "`write y`"? 5



More Aliasing

```
{ i, j, k : integer ;
  procedure q ( a, b : integer ) begin
    a := i * b;
    b := i * b;
  end q;
  ...
  i := 2; j := 3; k := 4;
  q(i,j);
  q(k,k);
}
```



- First call has global-formal aliases:
  - a and i
  - ~~b and j~~
- Second call has formal-formal alias:
  - a and b