# Parameters

We have discussed different classes of variables so far:
- Global/static variables
- Function local or automatic variables
-Dynamic, heap allocated variables

Chap 18

However, one important class of variables is still missing ⇒ parameters

Terminology: Example: Java, C, C++

Function Definition

```
int plus (int a, int b)
{
    return a + b;
}
```

Formal Parameters

Function Body

Function Call

```
    …
int x = plus(1,2);
    …
```

Actual Parameters

Observation: in function definitions formal parameters act as placeholders for the values of actual parameters.

## Reading

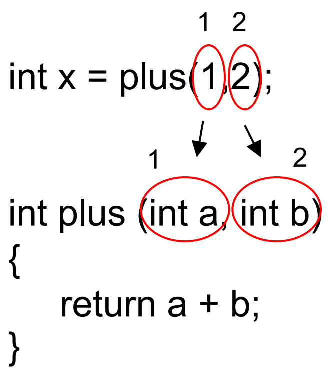
- Chap 18 in MPL

## Two Fundamental Questions

- How is the correspondence between actual and formal parameters established?
- How is the value of an actual parameter transmitted to a formal parameter?

# Correspondence

Most programming languages use <u>positional parameters</u>; the first actual parameter is assigned to the first formal parameter, the second actual parameter is assigned to the second formal parameters, *etc.*



```
                1  2
int x = plus(1,2);

          1           2
int plus (int a, int b)
{
    return a + b;
}
```

## Correspondence

Some languages such as Ada provide keyword parameters.

Example: Ada

```
FUNCTION Divide(Dividend:Float, Divisor:Float) RETURN Float IS
BEGIN
   RETURN Dividend/Divisor;
END

...
Foo = Divide(Divisor => 2.0, Dividend => 4.0);
...
```

2nd formal
parameter
becomes 2.0

1st formal
Parameter
Becomes 4.0

## Parameter Value Transmission

- We look at two different techniques
  - By value
  - By reference

## I. By Value

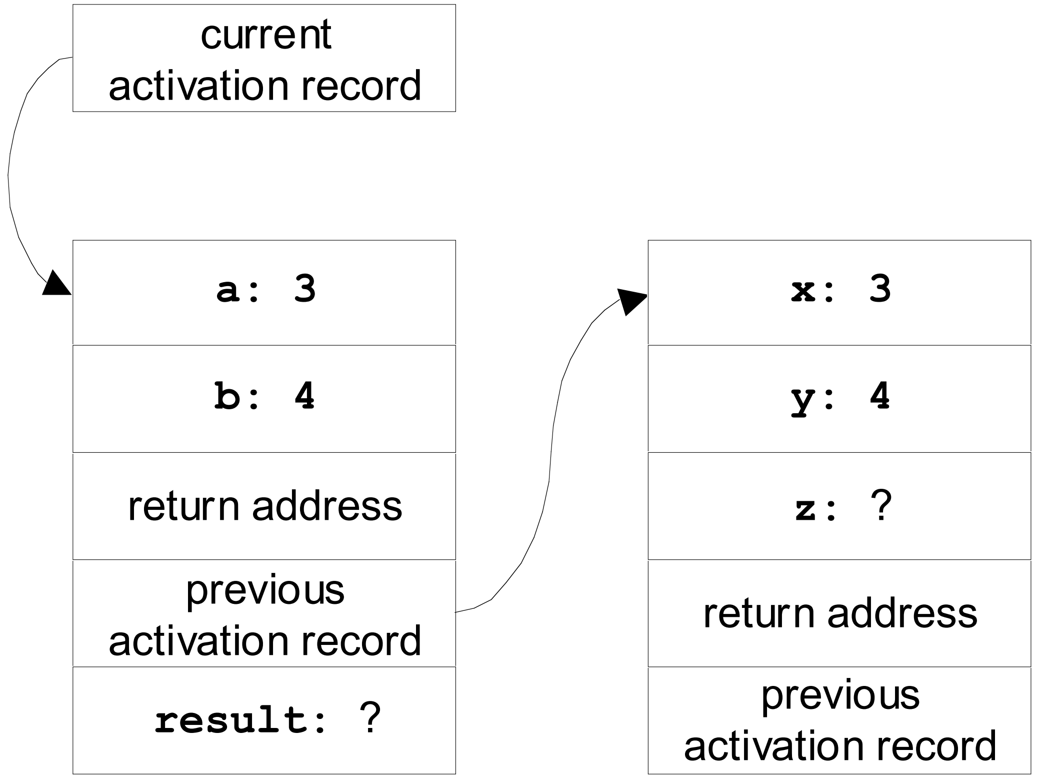
For by-value parameter passing, the formal parameter is just like a local variable in the activation record of the called method, with one important difference: it is initialized using the value of the corresponding actual parameter, before the called method begins executing.

- Also called ‘copy-in’
- Simplest method
- Widely used
- The only method in Java

# By Value - Example

```
int plus(int a, int b) {
  a += b;
  return a;
}

void f() {
  int x = 3;
  int y = 4;
  int z = plus(x, y);
}
```


current activation record
a: 3
b: 4
return address
previous activation record
result: ?
x: 3
y: 4
z: ?
return address
previous activation record

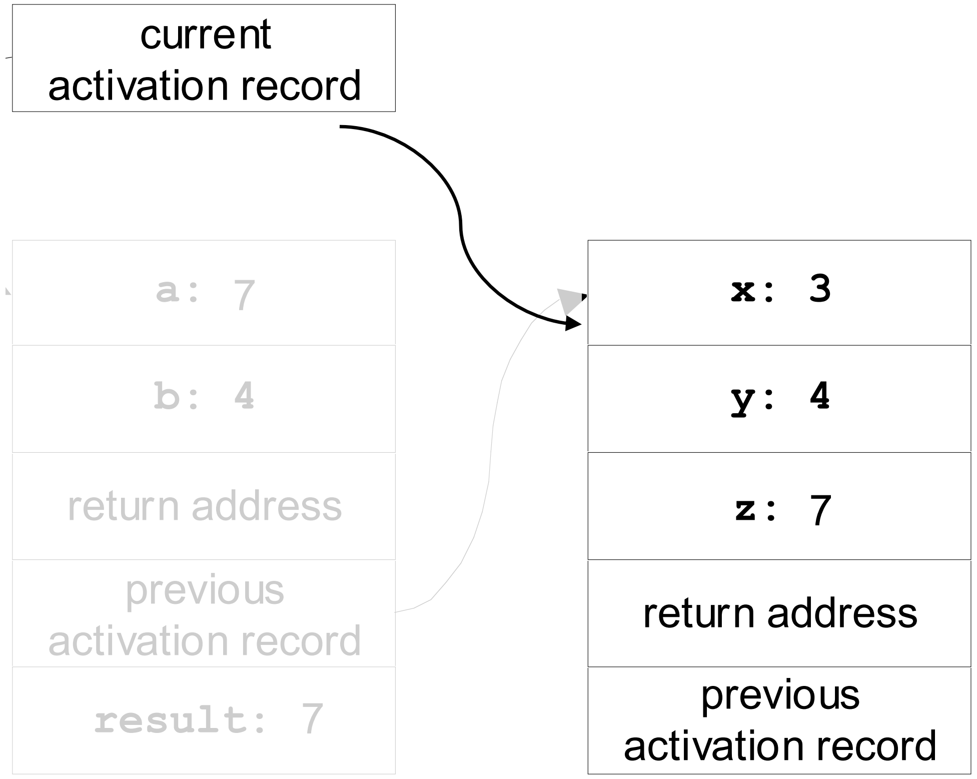
When **`plus`** is starting

# By Value - Example

```
int plus(int a, int b) {
  a += b;
  return a;
}

void f() {
  int x = 3;
  int y = 4;
  int z = plus(x, y);
}
```

After **`plus`** returns



current activation record
a: 7
b: 4
return address
previous activation record
result: 7
x: 3
y: 4
z: 7
return address
previous activation record

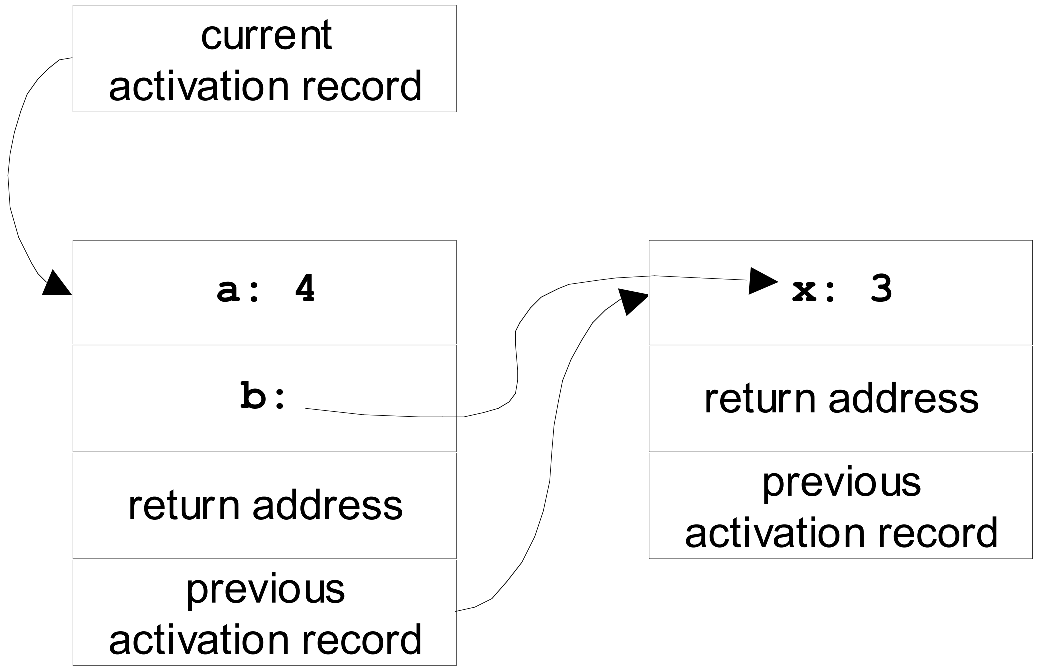
## III. By Reference

For passing parameters by reference, the lvalue of the actual parameter is computed before the called method executes. Inside the called method, that lvalue is used as the lvalue of the corresponding formal parameter. In effect, the formal parameter is an alias for the actual parameter—another name for the same memory location.

- One of the earliest methods: Fortran
- Most efficient for large objects
- Still frequently used; C++ allows you define calls by reference

# By Reference - Example

```
void plus(int a, by-reference int b) {
  b += a;
}
void f() {
  int x = 3;
  plus(4, x);
}
```

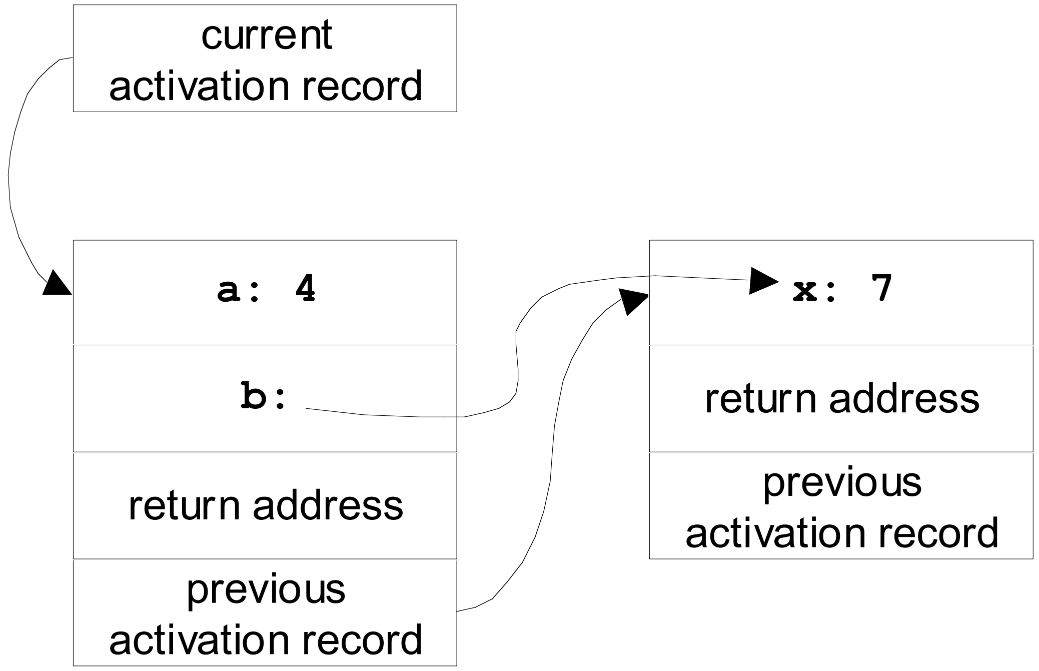


current activation record
a: 4
b:
return address
previous activation record
x: 3
return address
previous activation record

When **plus** is starting

## By Reference - Example

```
void plus(int a, by-reference int b) {
  b += a;
}
void f() {
  int x = 3;
  plus(4, x);
}
```



current activation record
a: 4
b:
return address
previous activation record
x: 7
return address
previous activation record

When **`plus`** has made the assignment

## Assignment

- Assignment #6 -- see BrightSpace