Overview (1A)

Young Won Lim
9/25/17

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Calculating the Mean

The mean of 3 numbers

$$m = \frac{a + b + c}{3}$$

$$\frac{40 + 50 + 60}{3} = \frac{150}{3} = 50$$

Integer number

fixed point number

$$\frac{45 + 53 + 63}{3} = \frac{161}{3} = 53.6666666\ldots$$

Real number

floating point number

Calculating a mean in C

```
int     a, b, c;
int     mean;

a = 40;
b = 50;
c = 60;

mean = (a + b +c) / 3;
```

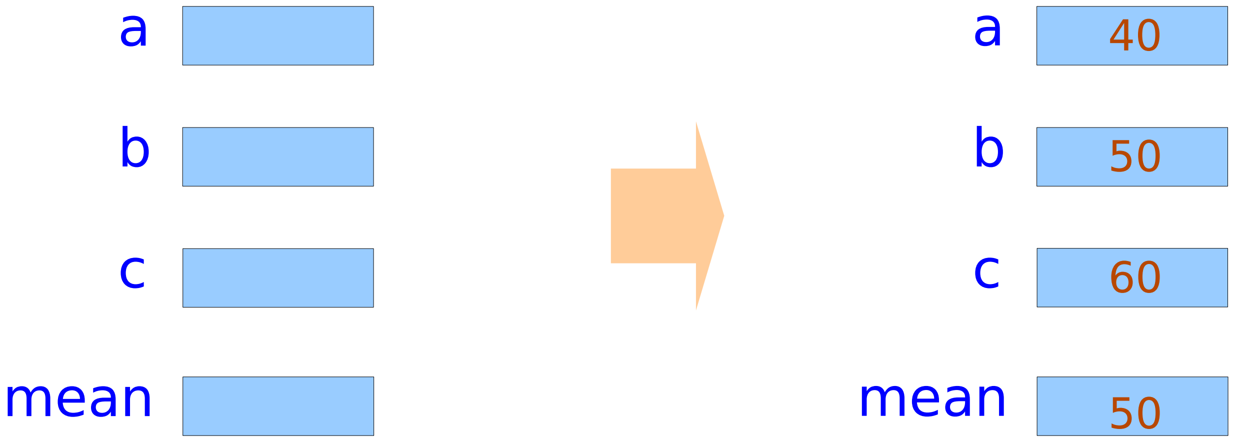
```
int     a, b, c;
float   mean;

a = 45;
b = 53;
c = 63;

mean = (a + b +c) / 3.0;
```

* Variable
* Type
* Assignment
* Operator

Variables – int Type

```
int     a, b, c;
int     mean;
```

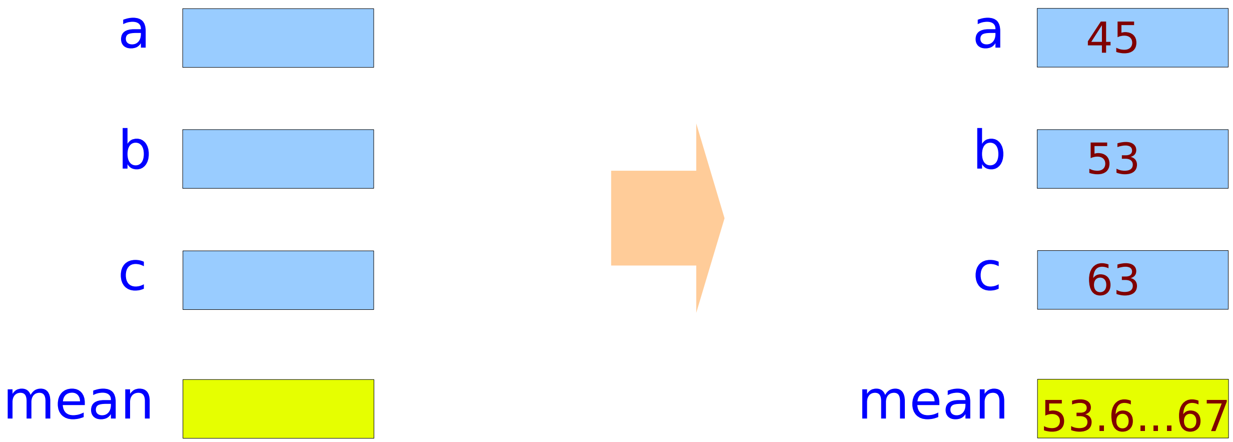
```
a = 40;
b = 50;
c = 60;
mean = (a + b +c) / 3;
```



Variables – float Type

```
int     a, b, c;
float   mean;
```

```
a = 45;
b = 53;
c = 63;

mean = (a + b +c) / 3.0;
```



a

b

c

mean

a 45

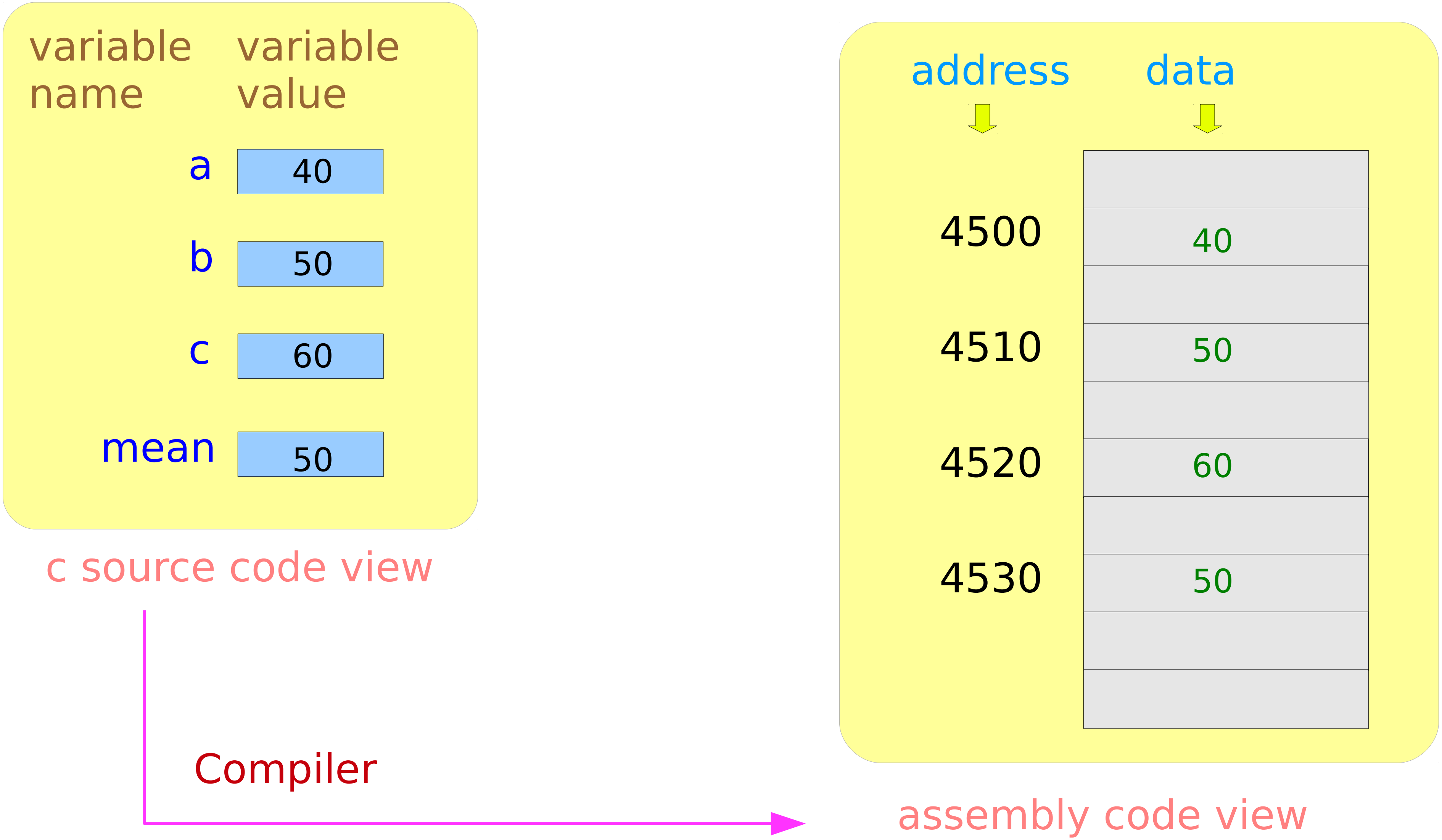
b 53

c 63

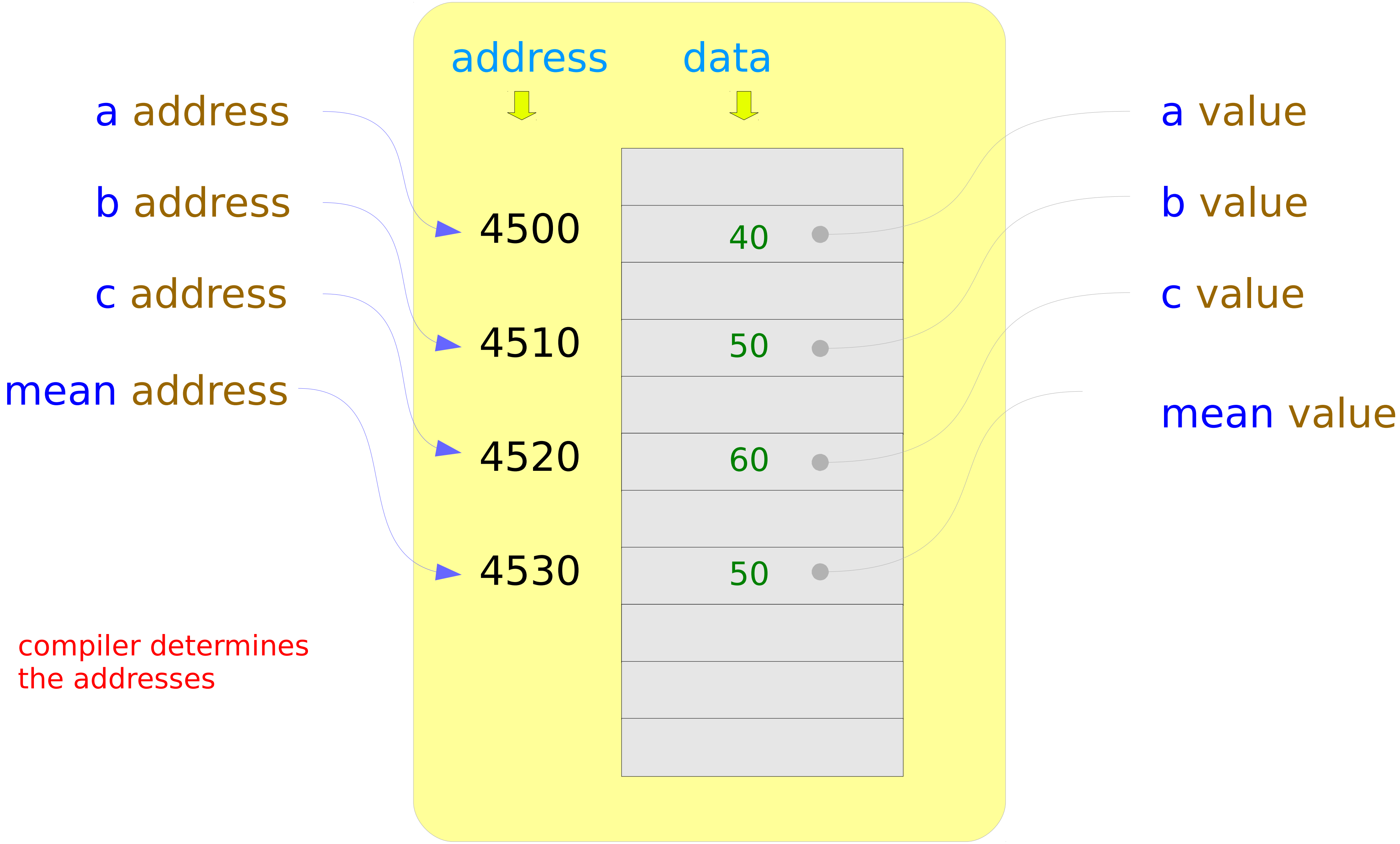
mean 53.6...67

float type

C and assembly code view of variables



Memory : (Address, Data)



compiler determines
the addresses

Getting the addresses of variables

&*variable* → address

Addresses determined by a compiler

&a → address of **a**

&b → address of **b**

&c → address of **c**

&mean → address of **mean**

Example: a variable stored in memory

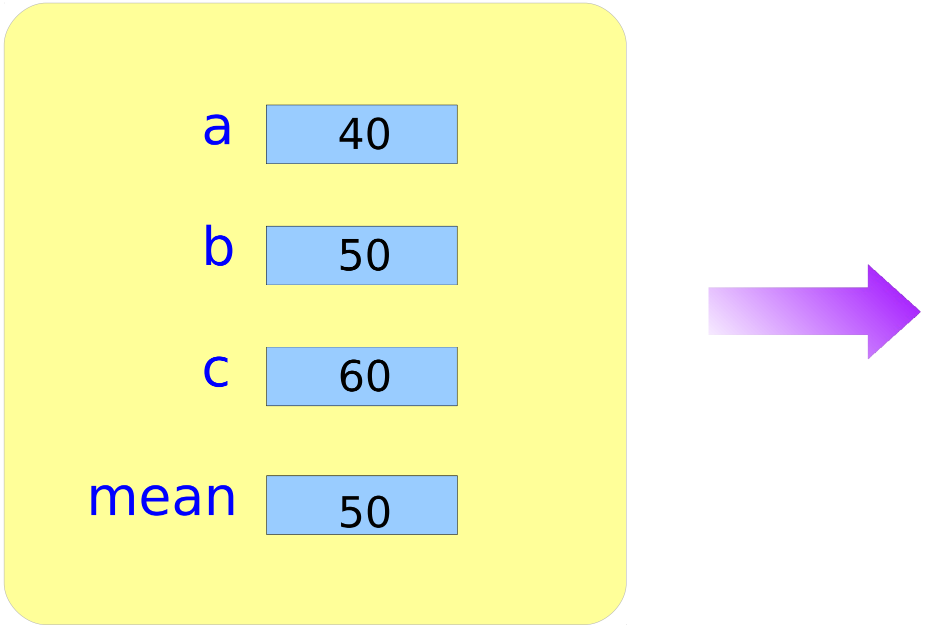
&*variable* → address

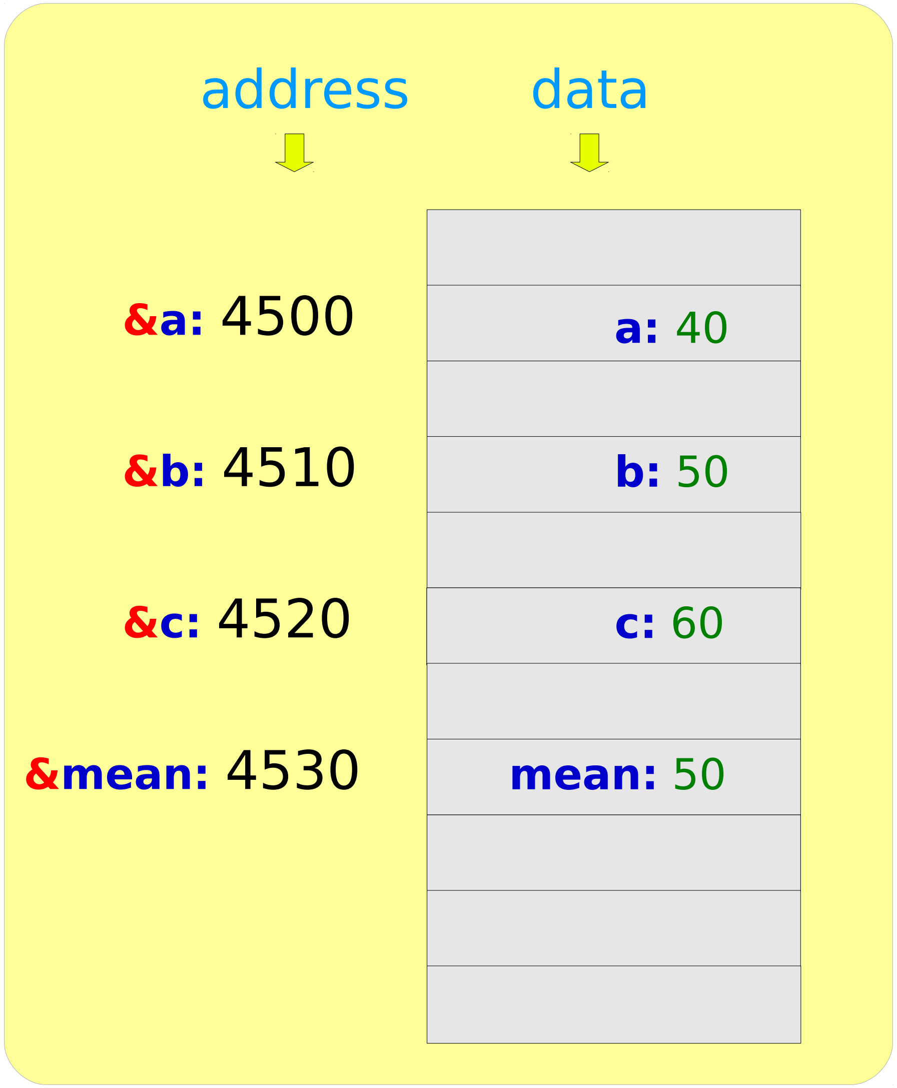
```
int    a;

a = 40;
```

&a	:address (*where*)	4500
a	:data (*what*)	40

Variables and the & operator

```
int     a=40, b=50, c=60;
int     mean=(a+b+c)/3;
```





address
data

&a: 4500
a: 40

&b: 4510
b: 50

&c: 4520
c: 60

&mean: 4530
mean: 50

Graphical representation of address assignment

a = 40;   Value assignment to a

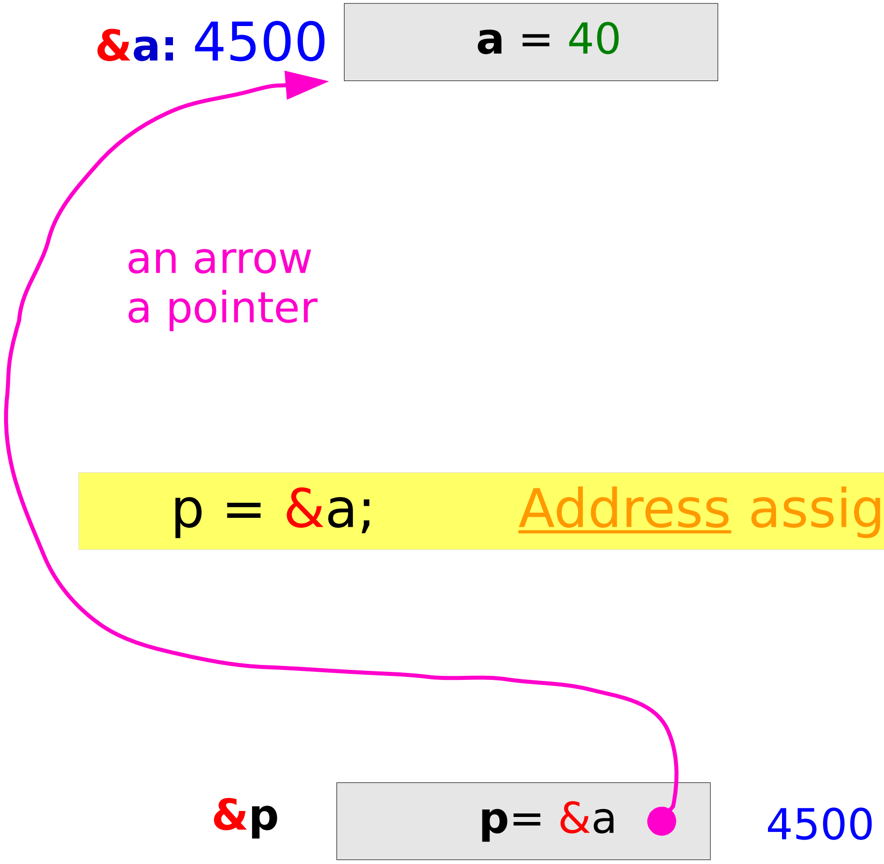
int a ;

integer variable

p = &a;   Address assignment to p

int * p ;

pointer variable



Data and Address Operators

Address operator

&*variable* → address

Data operator

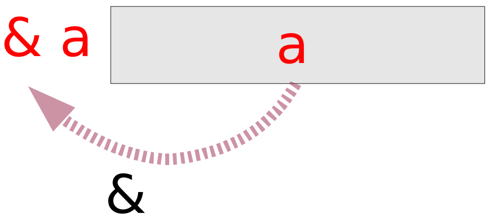
**variable* → data

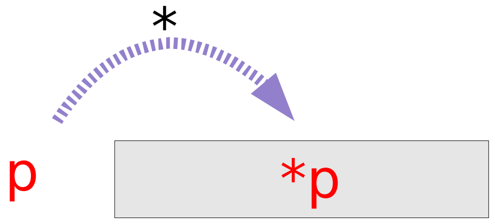
&a	address of **a**	***(&a)**	data at the address of **a**
&b	address of **b**	***(&b)**	data at the address of **b**
&c	address of **c**	***(&c)**	data at the address of **c**
&mean	address of **mean**	***(&mean)**	data at the address of **mean**

The & and * operators

The address of a variable :
Address of operator &

the value at an address :
*Dereferencing operator **





&a and *p

&a

↓

a must be a <u>variable</u>

```
int a ;
```

integer variable a

***p**

↓

p’s value must be an <u>address</u>

```
int * p ;
```

pointer variable p

The address &a and the variable *p

&a an address

***p** a variable

a a variable

p an address

```
int a ;
```

integer variable a

```
int * p ;
```

pointer variable p

R/W Accessing &a and *p

int a ;

integer variable a

& a ~~=~~ … ~~(write)~~

… = & a (read)

like a constant

int * p ;

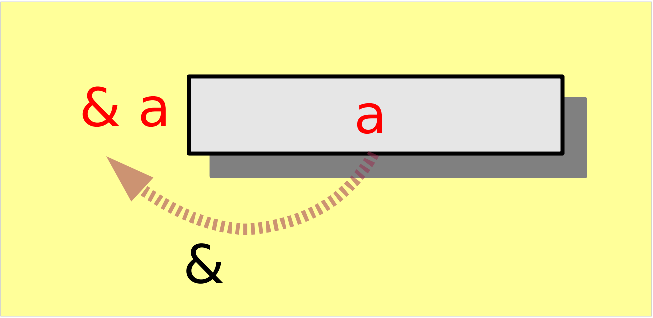
pointer variable p

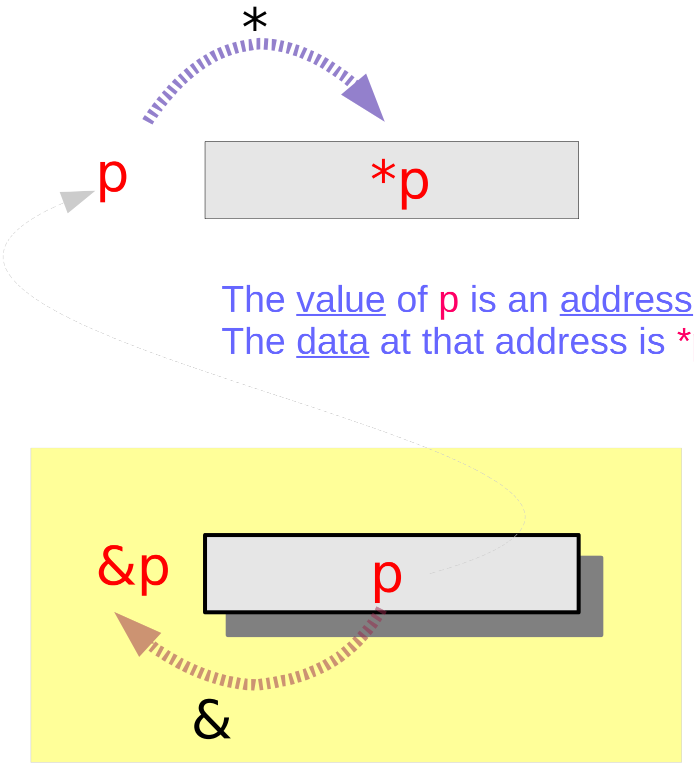
* p = … (write)

… = * p (read)

like a variable

&a and *p examples





The value of p is an address.
The data at that address is *p

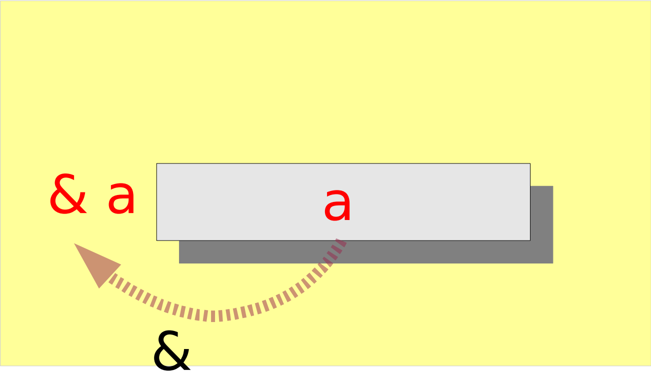
int a ;

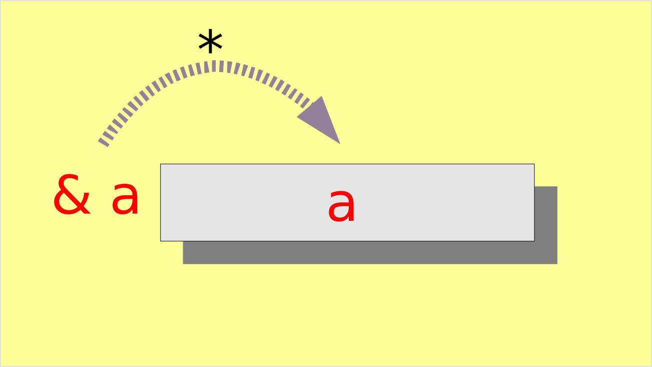
integer variable a

int * p ;

pointer variable p

The & and * operators [int a]





*(& a) = a

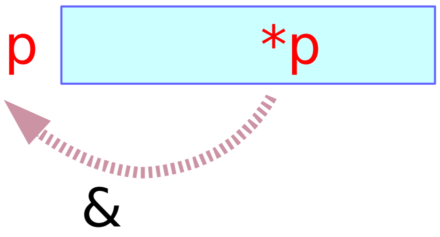
int a ;

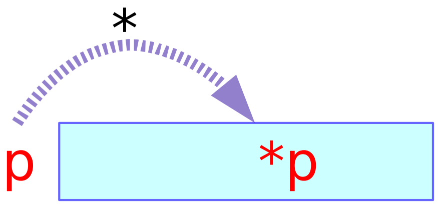
integer variable a

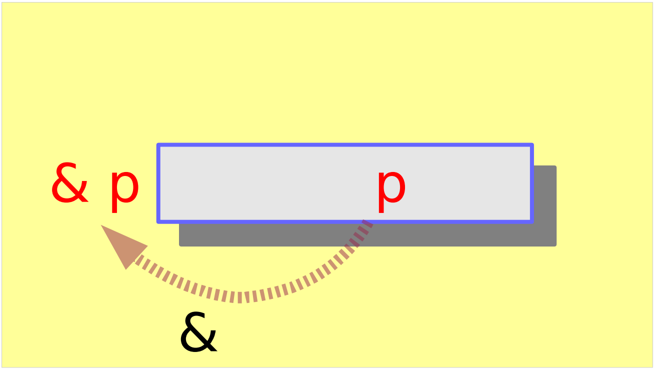
int a ;

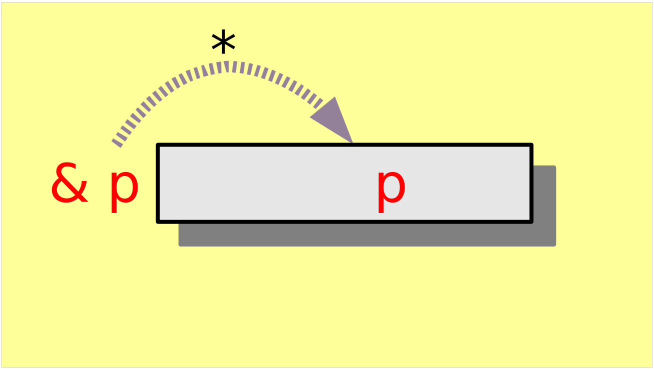
integer variable a

The & and * operators [int *p]









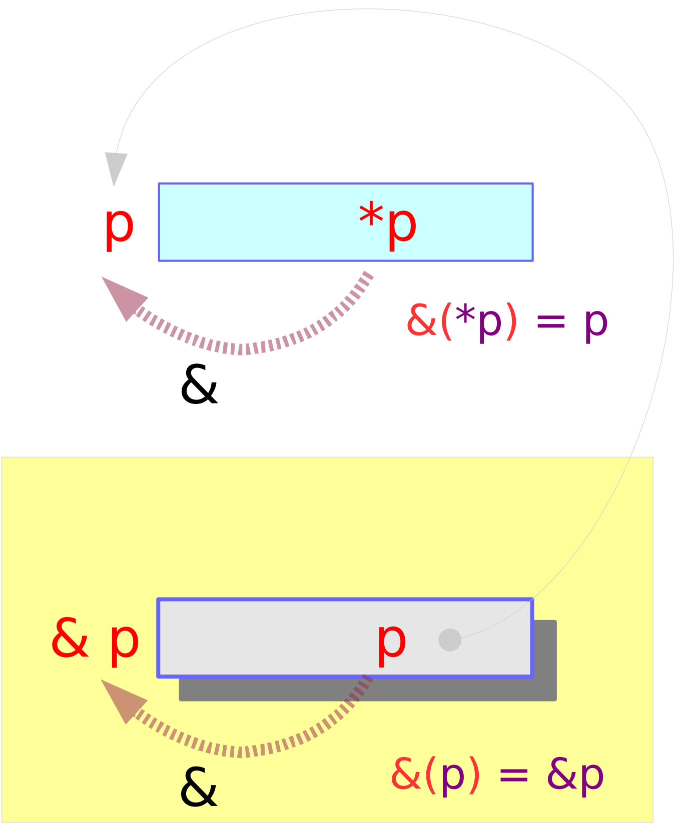
int * p ;

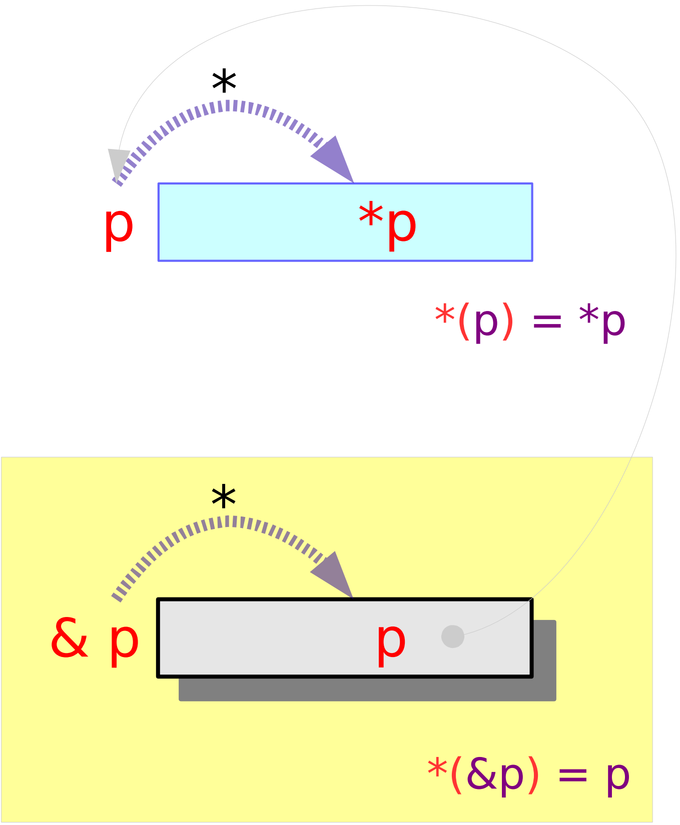
pointer variable p

int * p ;

pointer variable p

The & and * operators   [int *p]





value of p (→ *an address*)   value of p (→ *an address*)

The & and * operators   [int *p]

The address of a variable :
Address of operator &

&(p) = &p

&(*p) = p

```
int * p ;
```

pointer variable p

the value at an address :
*Dereferencing operator **

*(&p) = p

*(p) = *p

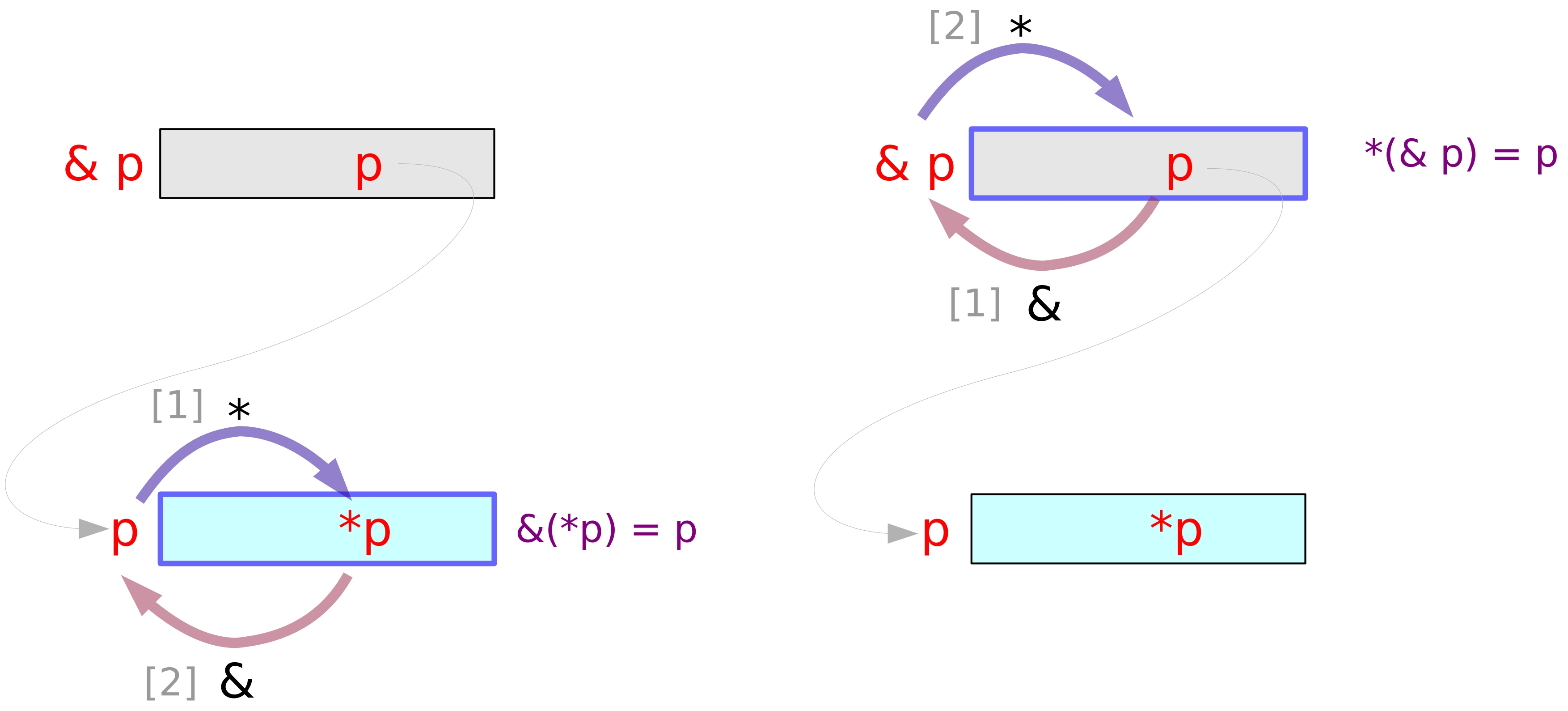
```
int * p ;
```

pointer variable p

The & and * cancel each other

$\cancel{\&}(\cancel{*}p) = p$

$\cancel{*}(\cancel{\&}\ p) = p$



Pointer Type Declaration

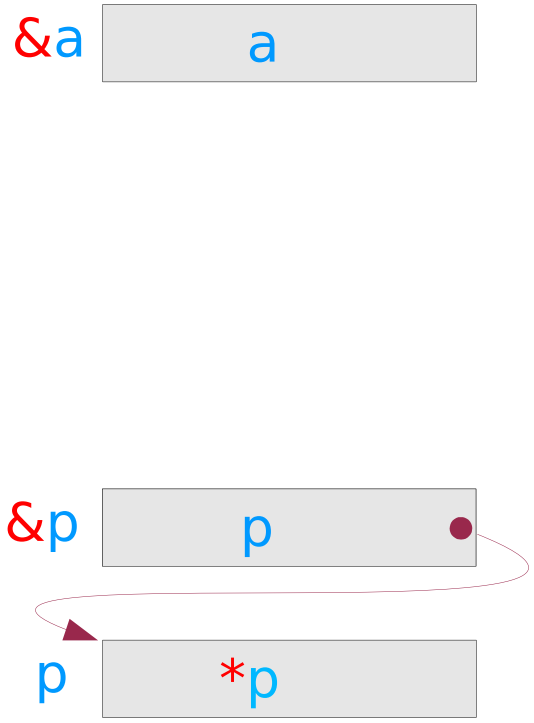
```
int     a;
```

a holds integer *value*

```
int     *p;
```

p holds *address*

*p holds integer *value*



value of p
(→ *an address*)

Address assignment to a pointer variable

```
int *p;
```

```
p = &a;
```

Address assignment

where: the address of p

what: the value of p
⬅&a (address of a)

The value of p is the address &a

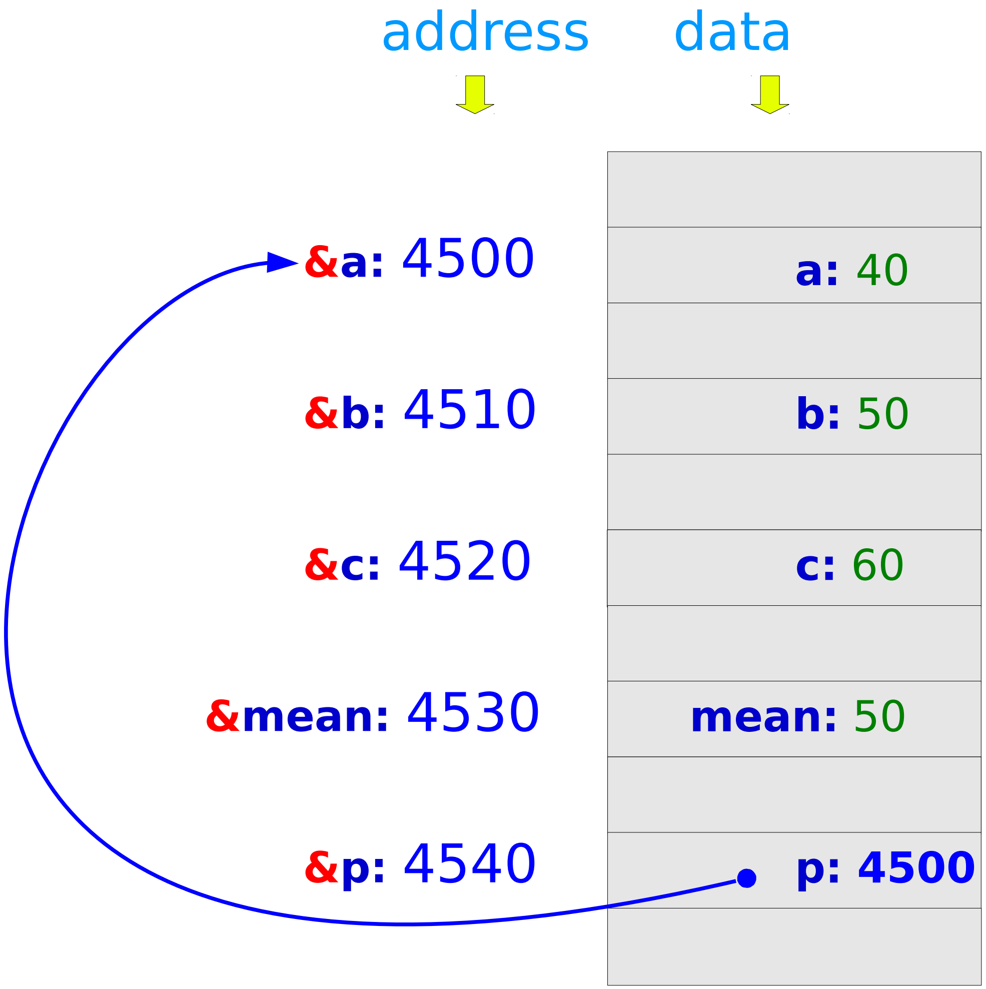
“p points to &a”
“p points to where the variable a is stored “

Address assignment example

```
int *p;
```

```
p = &a;
```

Address assignment



Value assignment to a pointer variable

```
int *p;
```

```
p = &a;
```

```
*p = 55;
```

Value assignment

where: the location where the value of p points to
(value of p → address)

what: the data at p
⬅ 55 (an integer value)

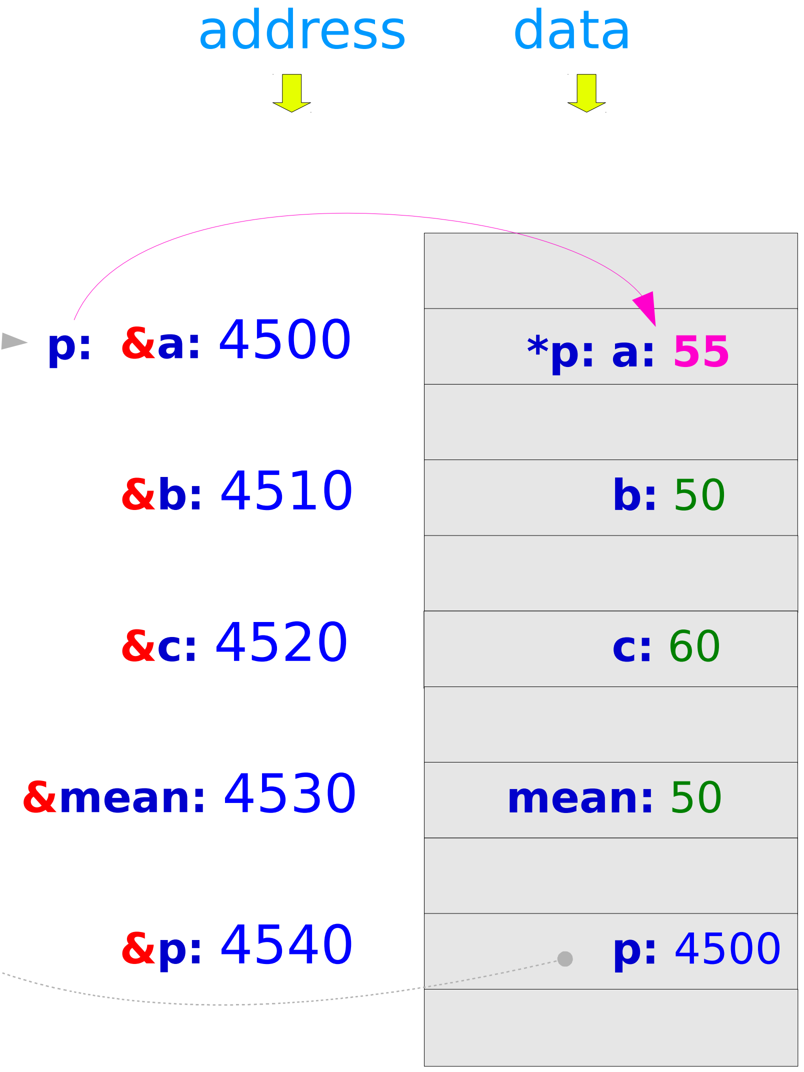
Value assignment to a pointer variable

```
int *p;
```

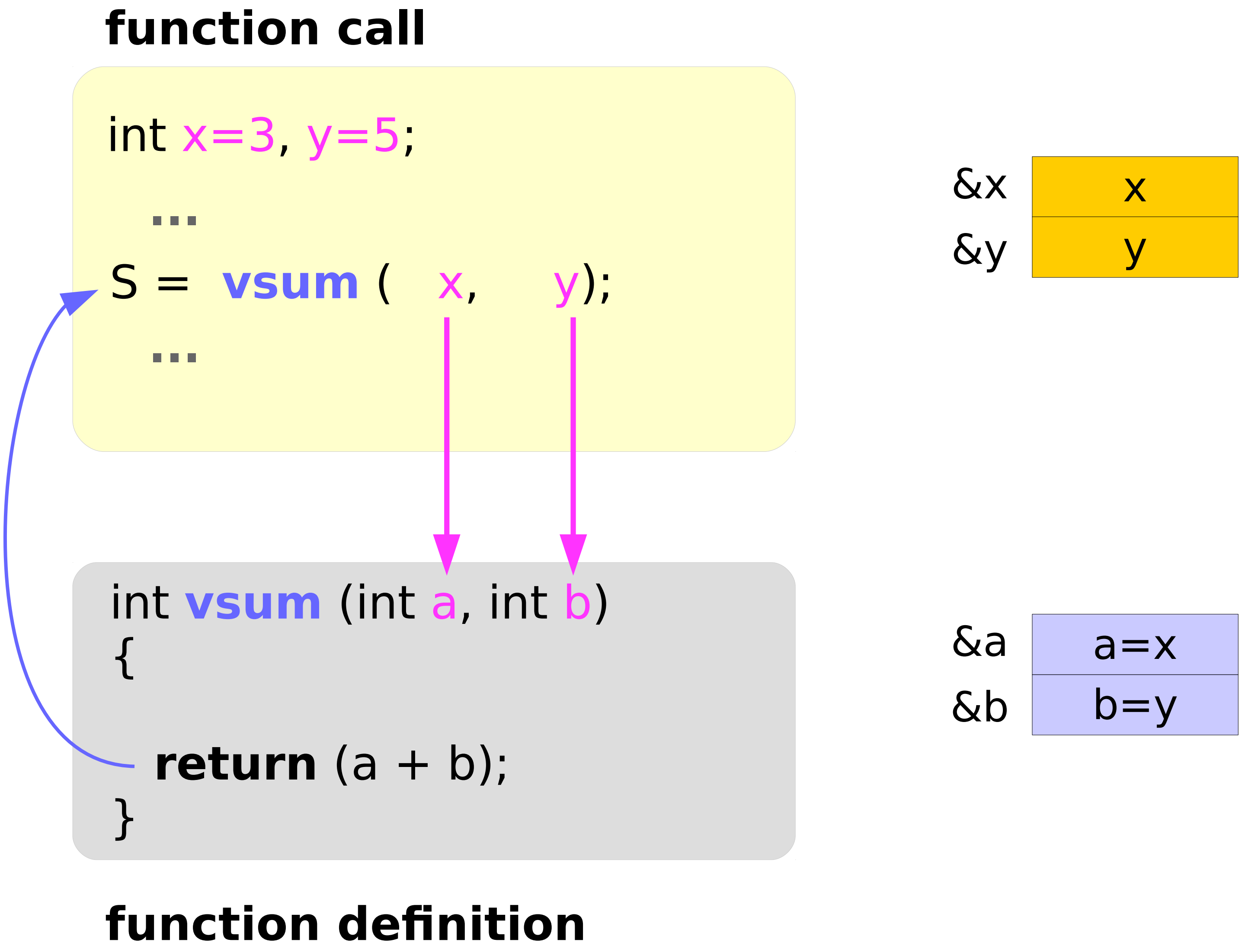
```
p = &a;
```

```
*p = 55;
```

Value assignment



Function examples – passing values



function call

```
int x=3, y=5;

  ...
S =  vsum (   x,     y);
  ...
```

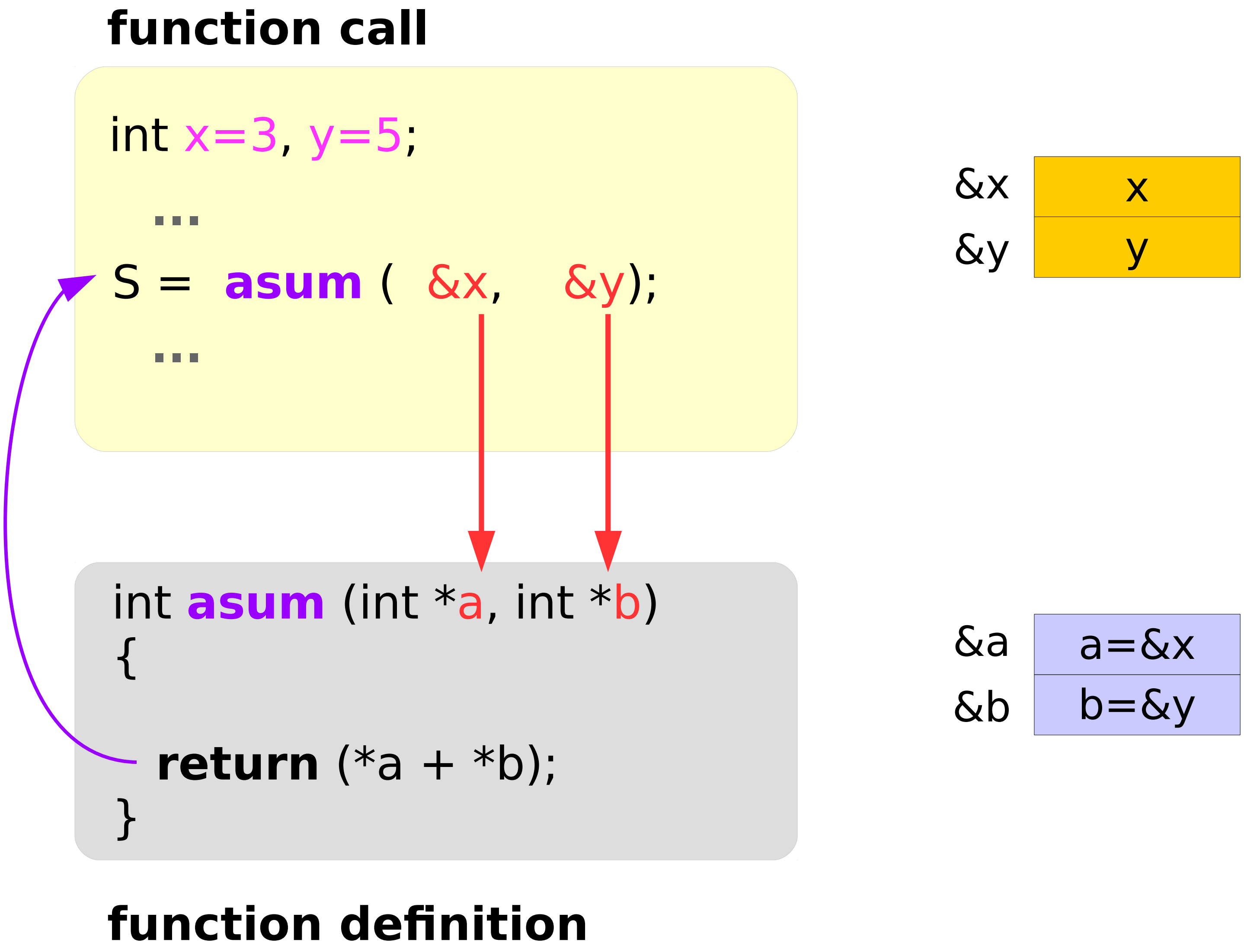
```
int vsum (int a, int b)
{

  return (a + b);
}
```

function definition

&x	x
&y	y

&a	a=x
&b	b=y

Function examples – passing addresses



function call

```
int x=3, y=5;
  ...
S =  asum (  &x,    &y);
  ...
```

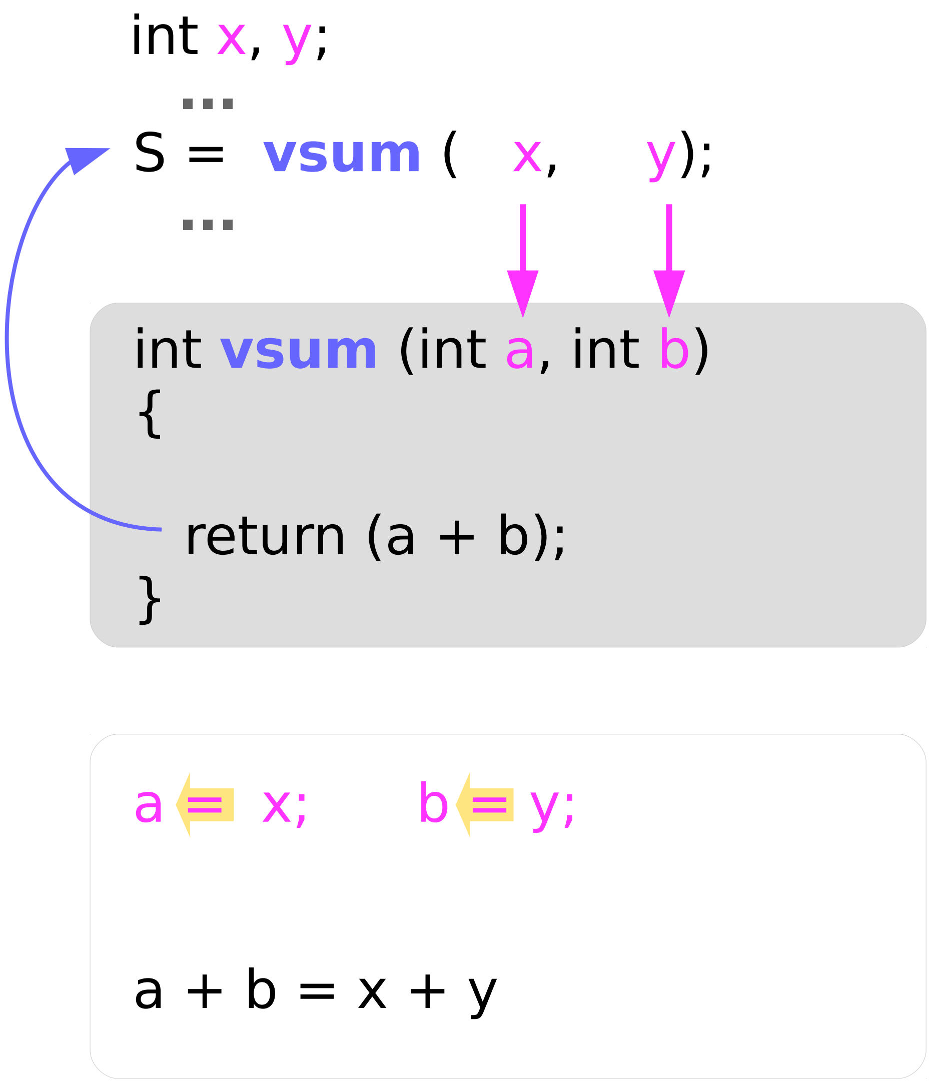
```
int asum (int *a, int *b)
{

   return (*a + *b);
}
```

function definition

&x	x
&y	y

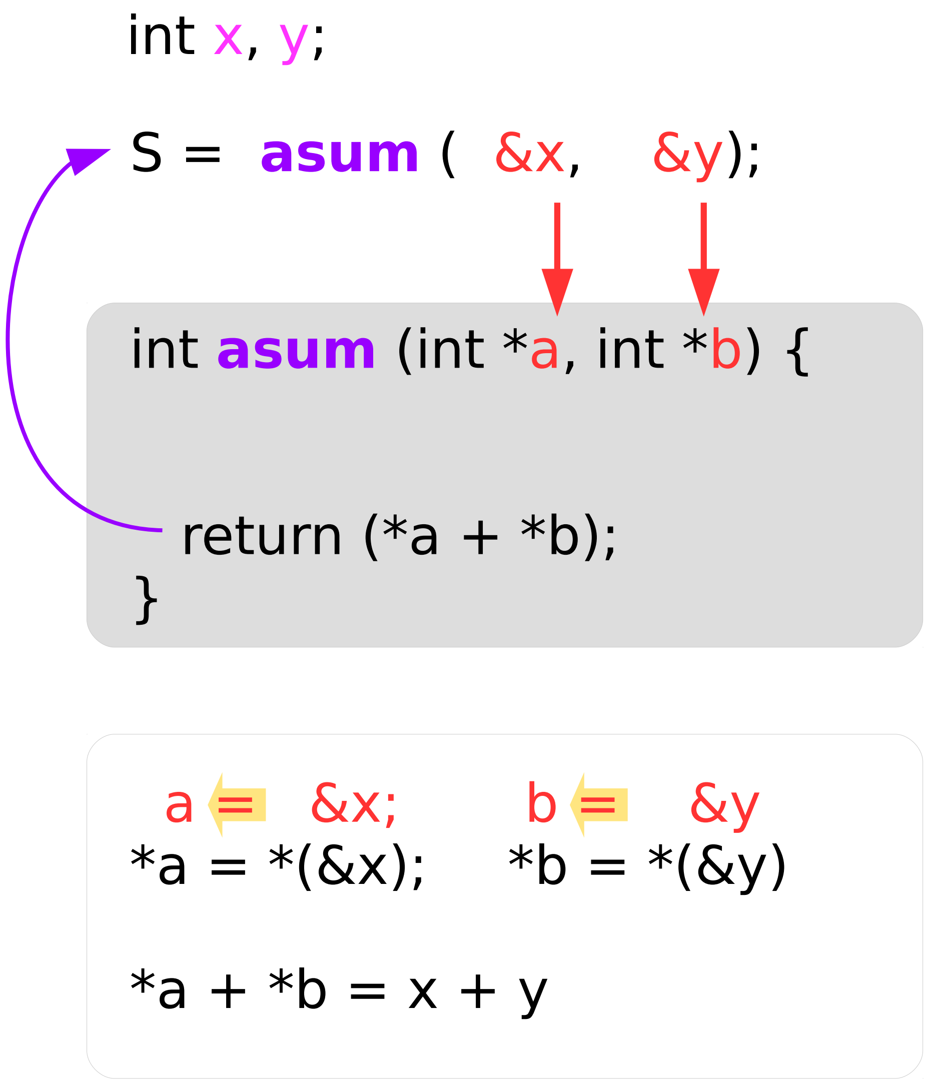
&a	a=&x
&b	b=&y

Comparison



```
int x, y;
  ...
S =  vsum (   x,      y);
  ...

int vsum (int a, int b)
{

  return (a + b);
}
```

```
a ⇐  x;       b ⇐ y;


a + b = x + y
```



```
int x, y;

S =  asum (  &x,     &y);

int asum (int *a, int *b) {


  return (*a + *b);
}
```

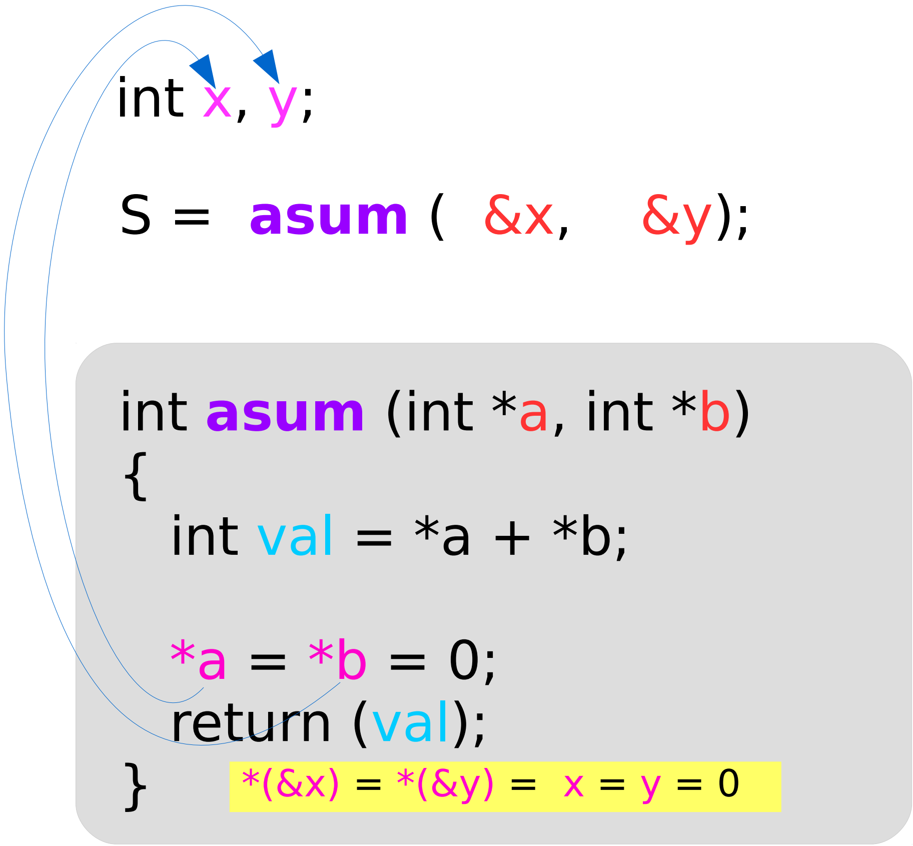
```
  a ⇐   &x;        b ⇐    &y
*a = *(&x);     *b = *(&y)


*a + *b = x + y
```

1-way vs. 2-way

```
int x, y;

S =  vsum (   x,      y);
```

```
int vsum (int a, int b)
{
   int val = a + b;

   a = b = 0;
   return (val);
}
```

```
a =  x;       b = y;

a = 0
b = 0
```



```
int x, y;

S =  asum (  &x,     &y);
```

```
int asum (int *a, int *b)
{
   int val = *a + *b;

   *a = *b = 0;
   return (val);
}
```

*(&x) = *(&y) = x = y = 0

```
a =   &x;        b =    &y

x = *(&x) = *a = 0
y = *(&y) = *b = 0
```

Differences

```
int x, y;

S =  vsum (   x,      y);
```

```
int vsum (int a, int b)
{
   int val = a + b;

   a = b = 0;
   return (val);
}
```

x, y : no change

```
int x, y;

S =  asum (  &x,     &y);
```

```
int asum (int *a, int *b) {

   int val = *a + *b;

   *a = *b = 0;
   return (val);
}
```

*(&x) = *(&y) = x = y = 0

x, y : changed to zeros

printf() : the built-in function

Expected Output

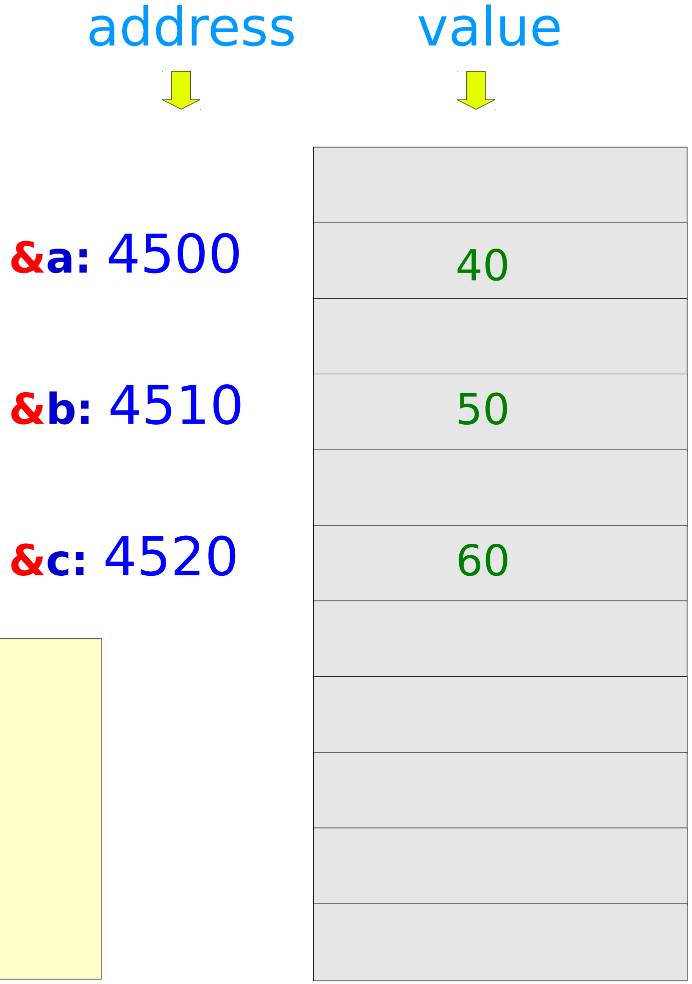
```
The mean of three numbers
a = 40
b = 50
c = 60
mean(40, 50, 60) => 50
```

```
printf(“The mean of three numbers \n”);
printf(“a = %d \n”, a);
printf(“b = %d \n”, b);
printf(“c = %d \n”, c);
printf(“mean (%d, %d, %d) => %d \n”, a, b, c, mean);
```

scanf() : another built-in function

Input Example

```
Enter three numbers!
a = 40≡
b = 50≡
c = 60≡
```



```
printf(“Enter three numbers! \n”);
printf(“a = ”); scanf(“ %d”, &a);
printf(“b = ”); scanf(“ %d”, &b);
printf(“c = ”); scanf(“ %d”, &c);
```

The Main Function (1)

```
main (void)
{
        int         a, b, c;
        int         mean;

        a = 40;
        b = 50;
        c = 60;

        mean = (a + b + c) / 3;

        printf(“The mean of three numbers \n”);
        printf(“a = %d \n”, a);
        printf(“b = %d \n”, b);
        printf(“c = %d \n”, c);
        printf(“mean (%d, %d, %d) => %d \n”, a, b, c, mean);

}
```

The Main Function (2)

```
main (void)
{
    int      a, b, c;
    int      mean;

    printf(“Enter three numbers! \n”);
    printf(“a = ”); scanf(“ %d”, &a);
    printf(“b = ”); scanf(“ %d”, &b);
    printf(“c = ”); scanf(“ %d”, &c);

    mean = (a + b + c) / 3;

    printf(“The mean of three numbers \n”);
    printf(“a = %d \n b = %d \n c = %d \n”, a, b, c);
    printf(“mean (%d, %d, %d) => %d \n”,
                    a,    b,    c,      mean  );

}
```

The Main Function (3)

```
main (void)
{
     int       a, b, c;
     int       mean;

     get_numbers( ? );



     compute_mean( ? );


     print_numbers( ? );



}
```

Function compute_mean()

```
int compute_mean (int x, int y, int z) ;
```

function prototype

```
main (void)
{
     int        mean;

     mean = compute_mean(40, 50, 60);

}          int values are copied
```

* Call by Value

```
int compute_mean (int x, int y, int z)
{
     int        avg;

     avg = (x + y + z) / 3;

     return( avg );

}
```

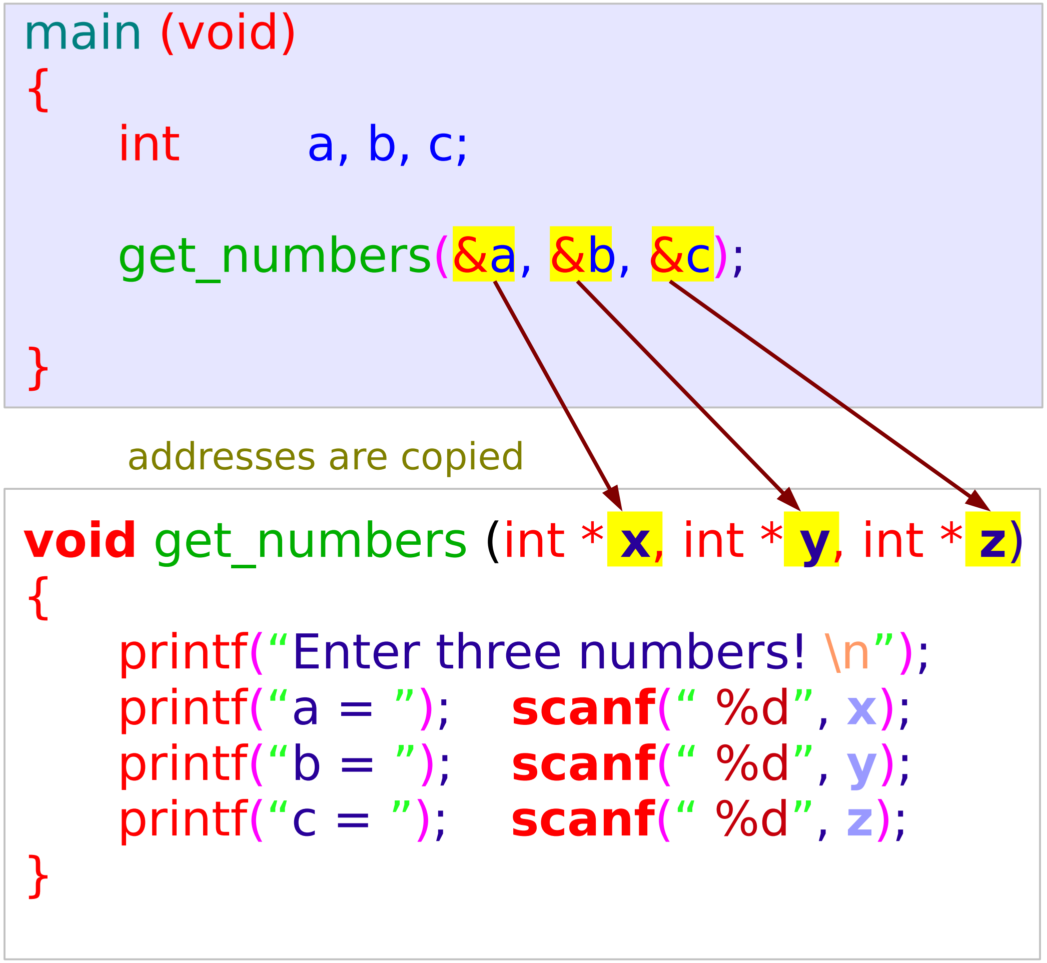
* Local Variable

* Return Value

Function get_numbers()

```
void get_numbers (int *x, int *y, int *z) ;
```

function prototype



```
main (void)
{
      int         a, b, c;

      get_numbers(&a, &b, &c);

}
```

addresses are copied

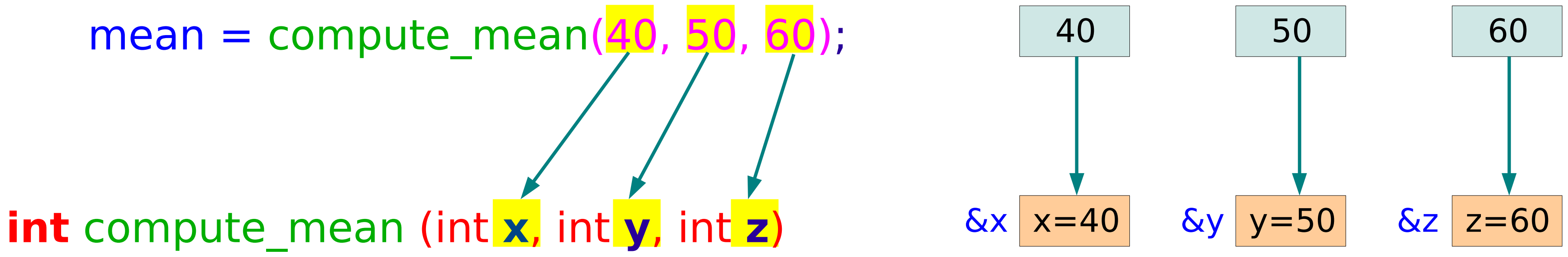
```
void get_numbers (int * x, int * y, int * z)
{
      printf("Enter three numbers! \n");
      printf("a = ");    scanf(" %d", x);
      printf("b = ");    scanf(" %d", y);
      printf("c = ");    scanf(" %d", z);
}
```

* Call by Reference

* No Return Value

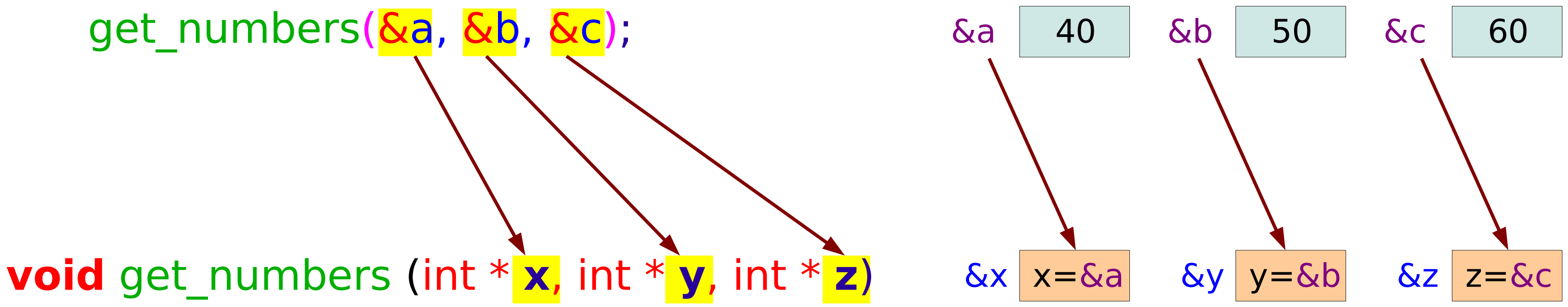
Call by Value & Call by Reference

* Call by Value



```
mean = compute_mean(40, 50, 60);

int compute_mean (int x, int y, int z)
```

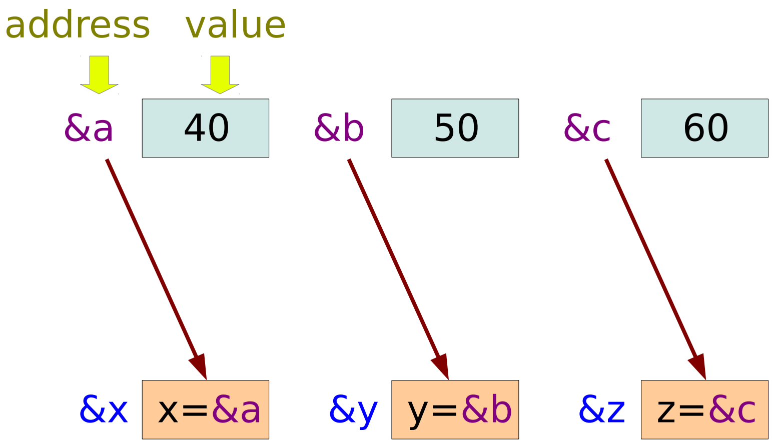
* Call by Reference



```
get_numbers(&a, &b, &c);

void get_numbers (int * x, int * y, int * z)
```

Change the caller's variables

* Call by Reference



Caller's Variables

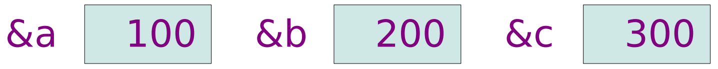
Now, values of a, b, c are changed
in the get_numbers() function

```
*x= 100;
*y= 200;
*z= 300;
```

→

```
a= 100;
b= 200;
c= 300;
```

The callee can change the
values of the caller's variables



Function print_numbers()

```
void print_numbers (int x, int y, int z, int avg)
{
     printf(“The mean of three numbers \n”);
     printf(“a = %d \n b = %d \n c = %d \n”, x, y, z);
     printf(“mean (%d, %d, %d) => %d \n”,  x, y, z, avg);
}
```

* Call by Value

* No Return Value

```
main (void)
{
     int        a, b, c;
     int        mean;

     print_numbers(a, b, c, mean);

}
```

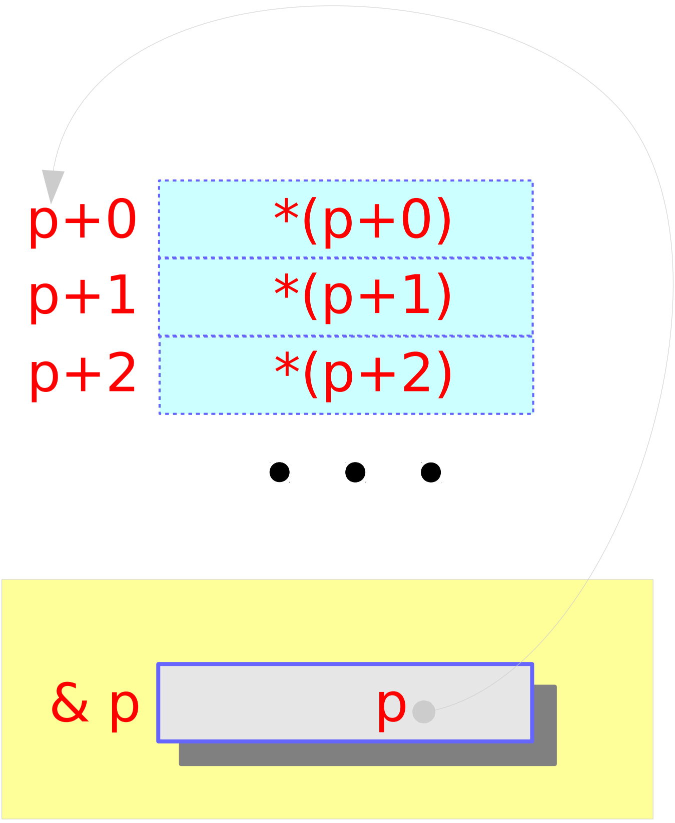
The Main Function (4)

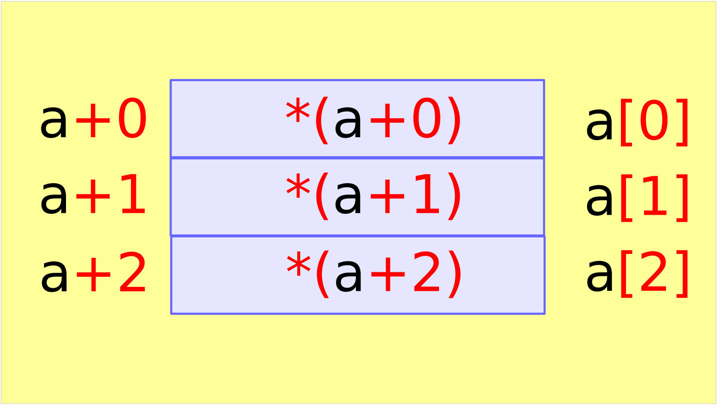
```
void  get_numbers      (int *x, int *y, int *z);
int   compute_mean     (int x, int y, int z);
void  print_numbers    (int x, int y, int z, int avg);
```

Prototypes

```
main (void)
{
     int        a, b, c;
     int        mean;

     get_numbers(&a, &b, &c);

     mean = compute_mean(a, b, c);

     print_numbers(a, b, c, mean);
}
```

Pointers and Arrays





```
int * p ;
```

pointer variable p

```
int  a[3] ;
```

array name a

The Main Parameters (1)

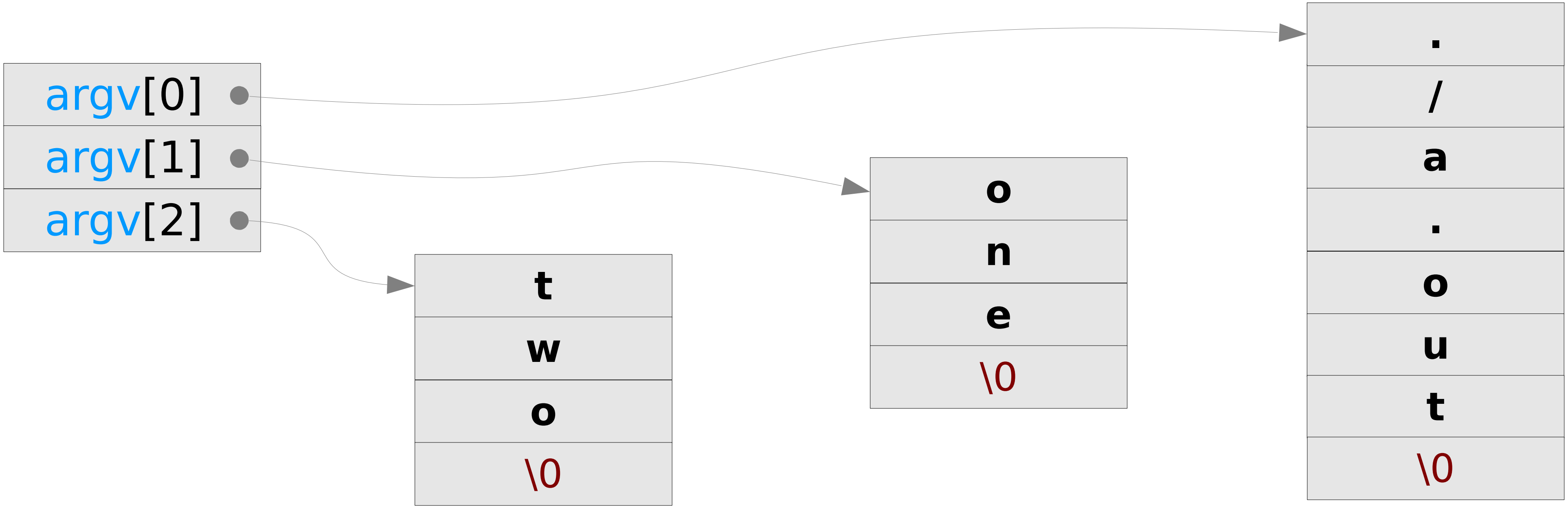
```
#include <stdio.h>

int main(int argc, char *argv[])
{

   int i;

   printf("argc= %d \n", argc);

   for (i=0; i<argc; ++i) {
     printf("argv[%d] = %s \n", i, argv[i]);
   }

}
```

The Main Parameters (2)

```
young@usys ~ $ ./a.out
argc= 1
argv[0] = ./a.out

young@usys ~ $ ./a.out  one  two  three
argc= 4
argv[0] = ./a.out
argv[1] = one
argv[2] = two
argv[3] = three

young@usys ~ $ ./a.out  one  two  three  four
argc= 5
argv[0] = ./a.out
argv[1] = one
argv[2] = two
argv[3] = three
argv[4] = four
```

The Main Parameters (3)

```
young@usys ~ $ ./a.out  one  two
argc= 3
argv[0] = ./a.out
argv[1] = one
argv[2] = two
```



References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun