

Applications of Pointers (1A)

Copyright (c) 2010 - 2018 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

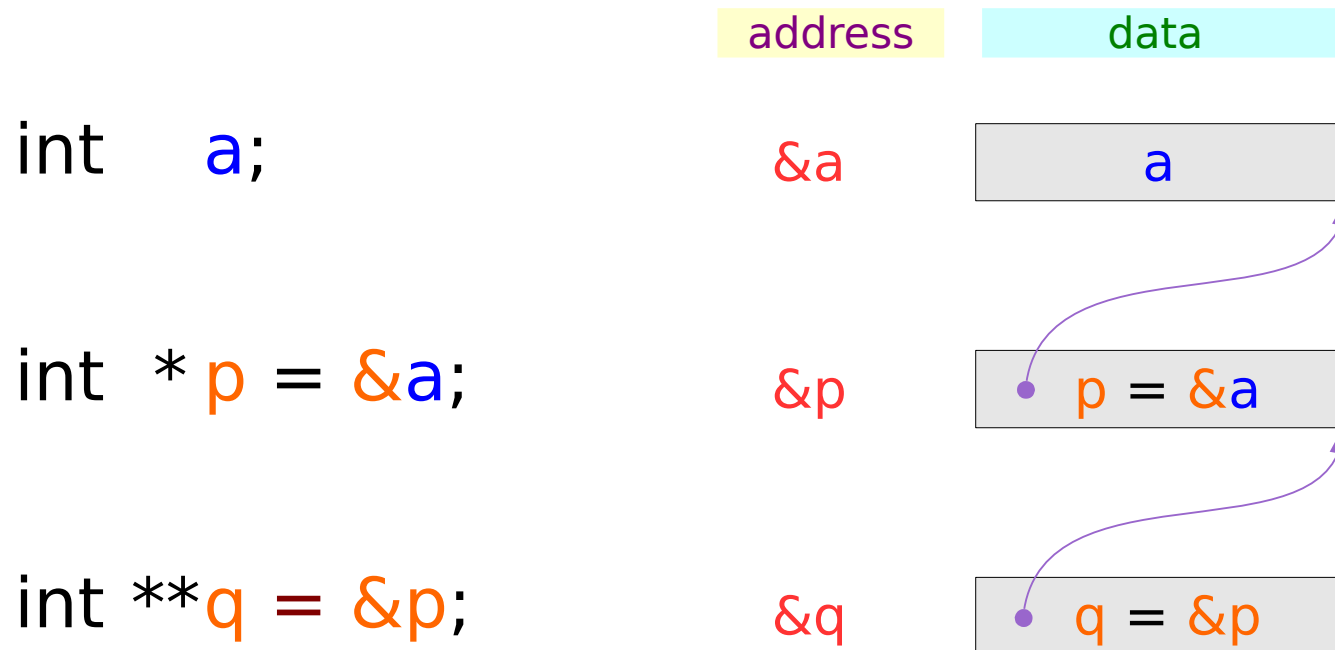
Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

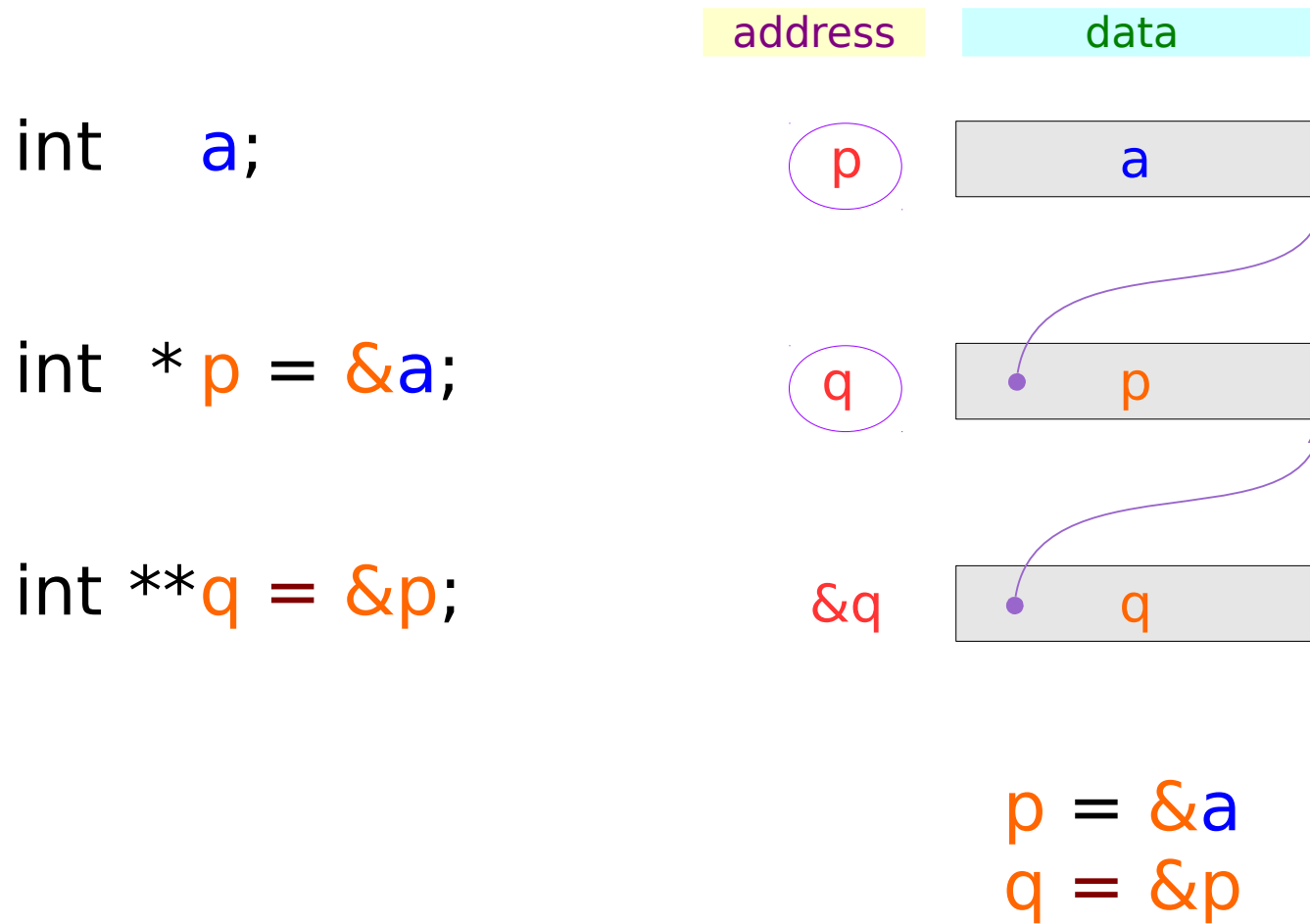
Variables and their addresses

	address	data
<code>int a;</code>	<code>&a</code>	<code>a</code>
<code>int *p;</code>	<code>&p</code>	<code>p</code>
<code>int **q;</code>	<code>&q</code>	<code>q</code>

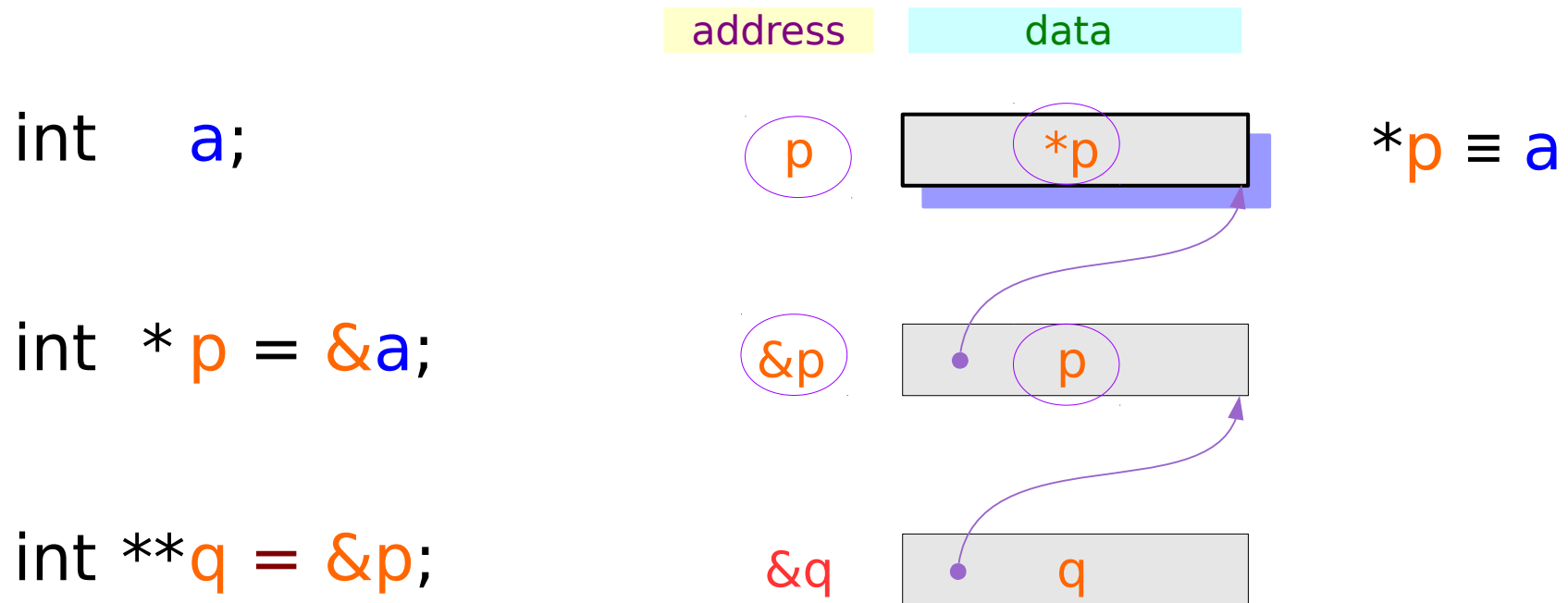
Initialization of Variables



Pointed addresses : p, q



Dereferenced Variables : *p



Dereferenced Variables : *p

```
int a;
```

```
int *p = &a;
```

```
int **q = &p;
```

Address
Assignment

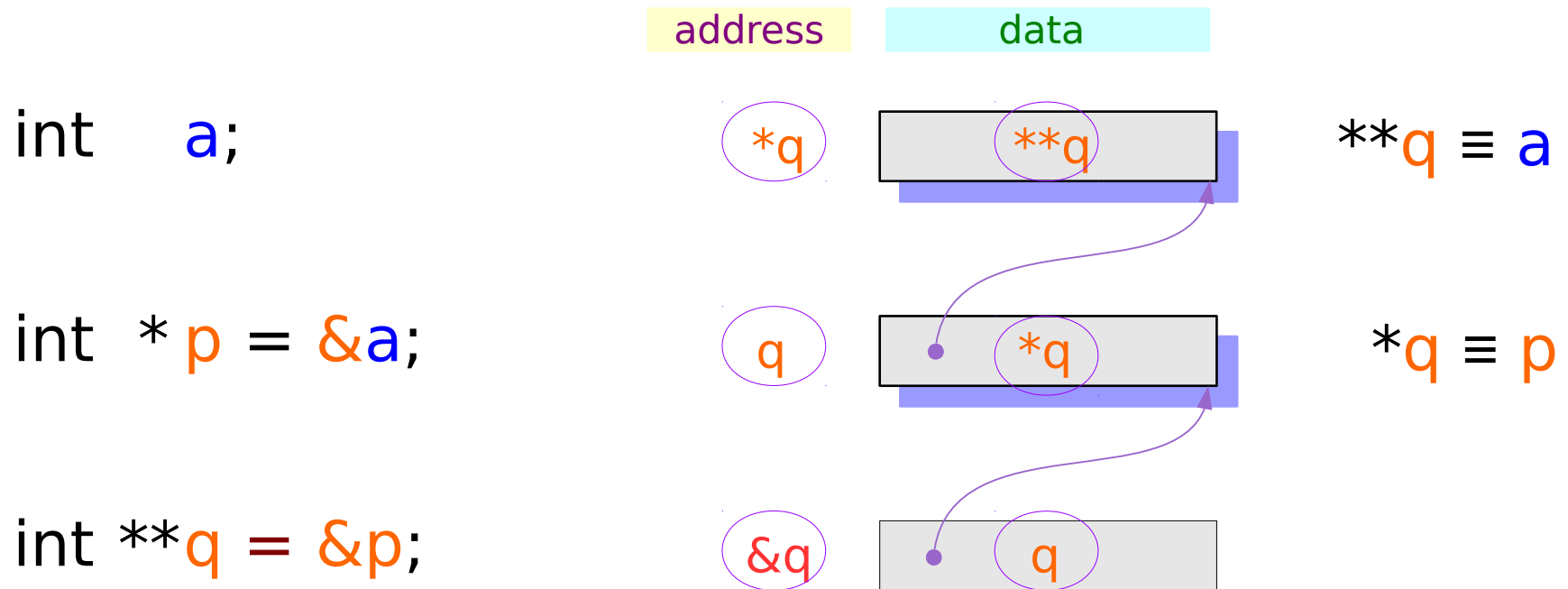
Variables
with the same address

$p = \&a \Rightarrow *p \equiv a$

$p \equiv \&a$
 $*(p) \equiv *(\&a)$
 $*p \equiv a$

Relations after
address assignment

Dereferenced Variables : *q, **q



Dereferenced Variables : *q, **q

```
int a;
```

Address
Assignment

Variables
with the same address

```
int *p = &a;
```

$p = \&a \Rightarrow *p \equiv a$

```
int **q = &p;
```

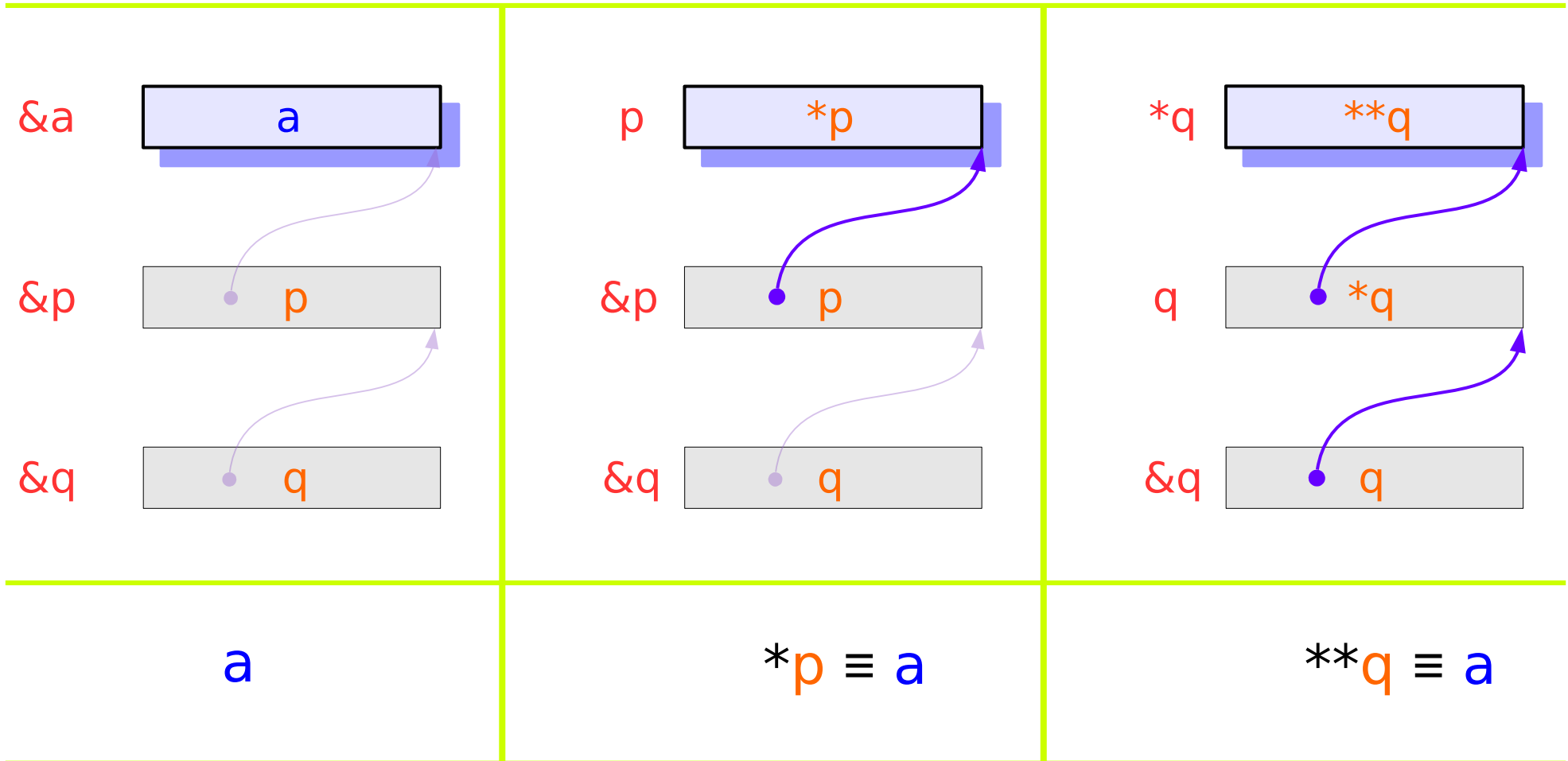
$q = \&p \Rightarrow *q \equiv p$

$\Rightarrow **q \equiv a$

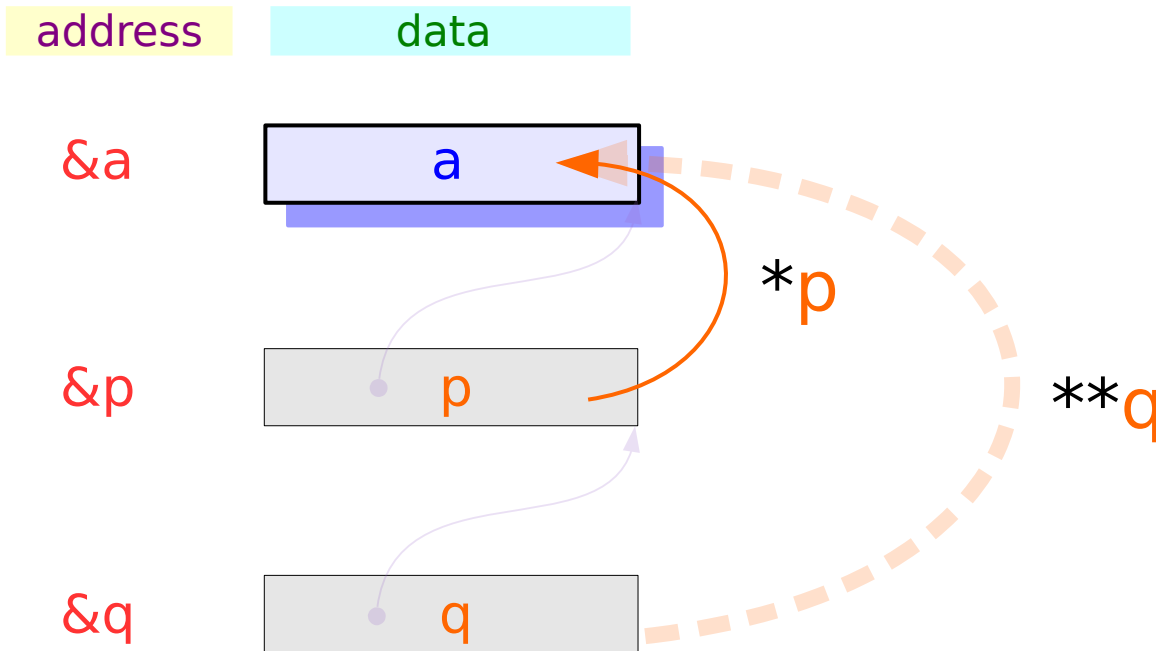
$q \equiv \&p$
 $*(q) \equiv *(\&p)$
 $*q \equiv p$
 $**q \equiv *p$
 $**q \equiv a$

Relations after
address assignment

Two more ways to access **a** : ***p**, ****q**



Two more ways to access a : *p, **q



- 1) Read / Write `a`
- 2) Read / Write `*p`
- 3) Read / Write `**q`

Variables

```
int a;
```

a can hold an integer

address

data

&a

a

```
a = 100;
```

a holds an integer 100

address

data

&a

a ← 100

Pointer Variables

```
int * p;
```

`p` holds an address

```
int * p;
```

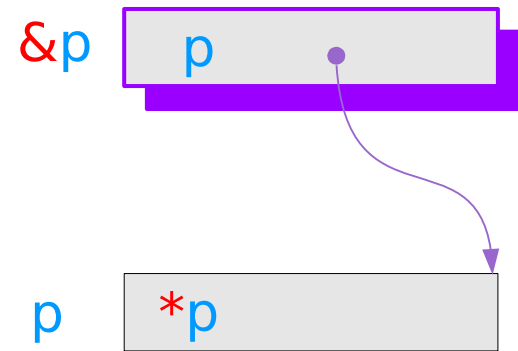
`p` holds an address
of a `int` type data

pointer to int

```
int * p;
```

`*p` holds an integer

int



Pointer to Pointer Variable

```
int ** q;
```

q holds an address

```
int ** q;
```

pointer to
pointer to int

q holds an address of
a pointer to int type
data

```
int * *q;
```

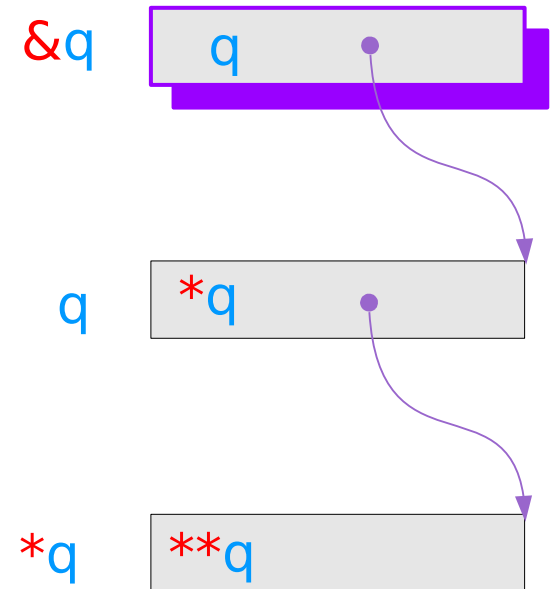
pointer to int

*q holds an address of
a int type variable

```
int **q;
```

int

**q holds an integer

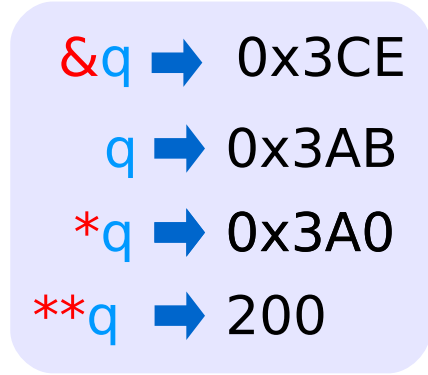
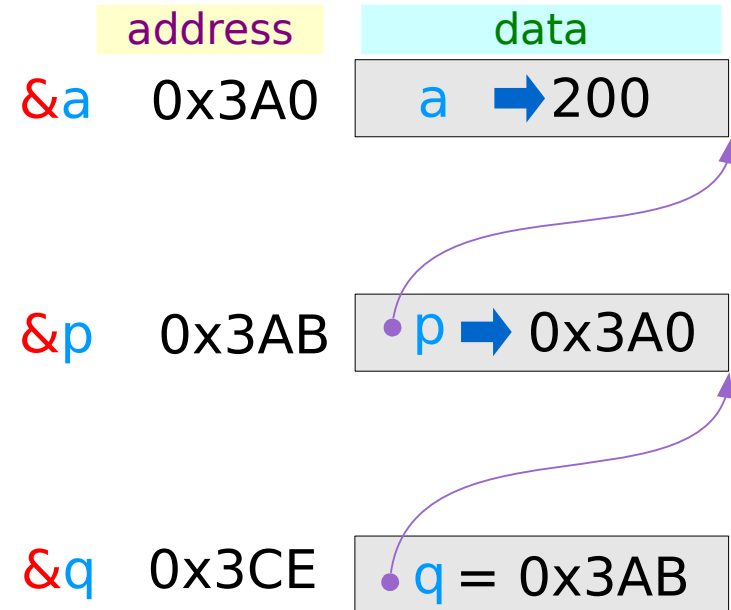


Pointer Variables Examples

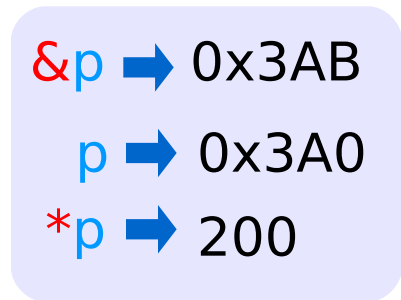
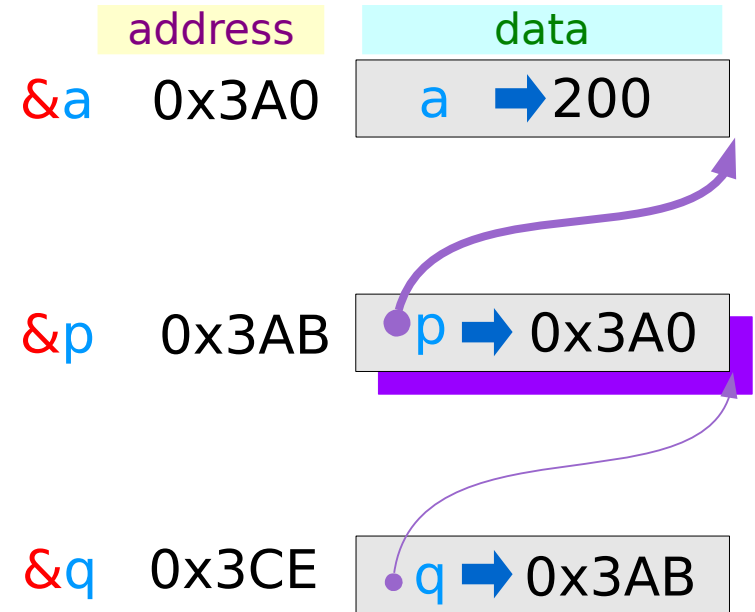
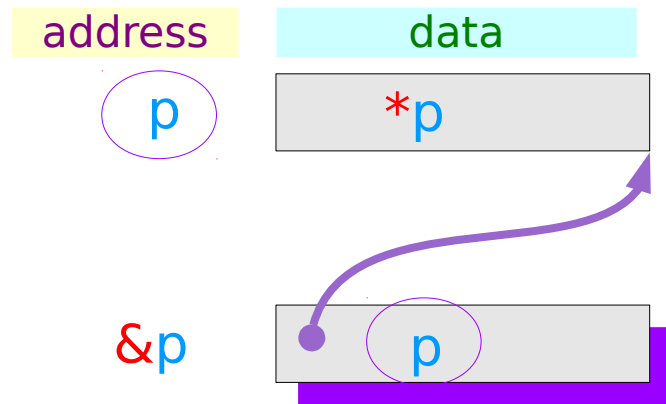
```
int    a;
```

```
int *  p = &a;
```

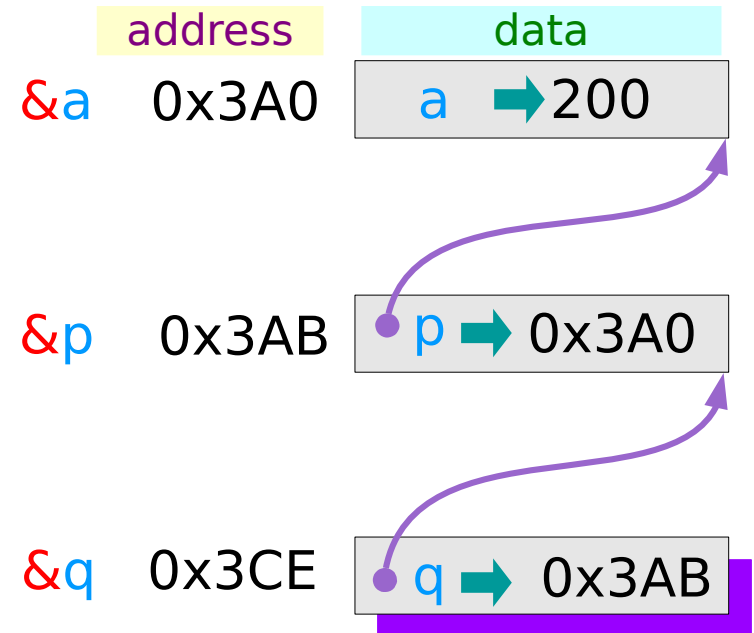
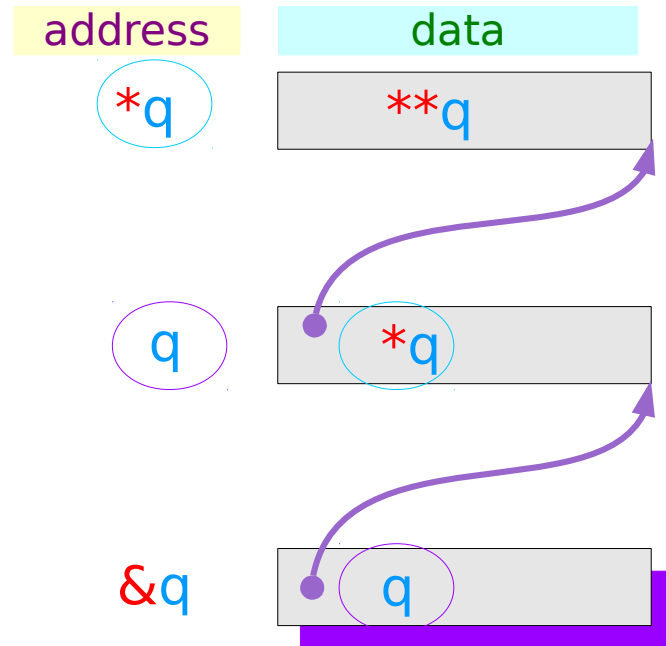
```
int ** q = &p;
```



Pointer Variable **p**

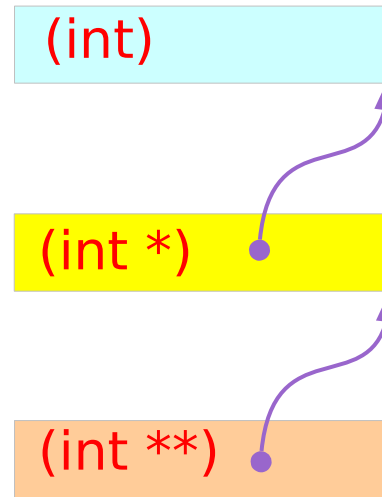
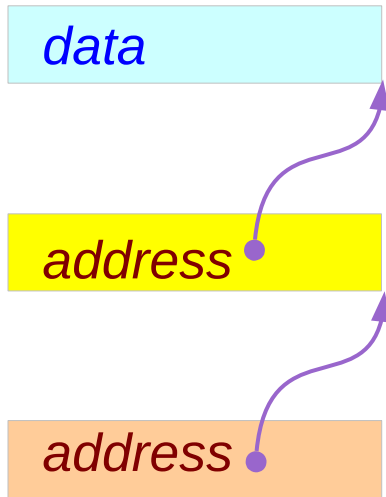


Pointer Variable **q**



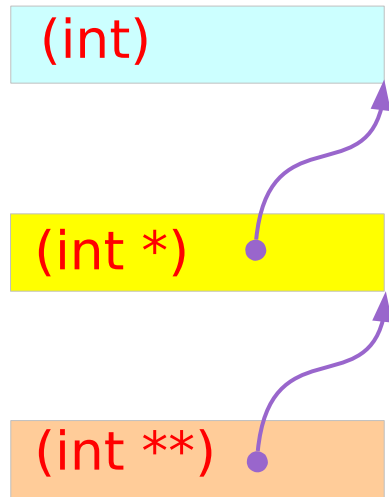
`&q` → 0x3CE
`q` → 0x3AB
`*q` → 0x3A0
`**q` → 200

Interpretation of Pointers - Types

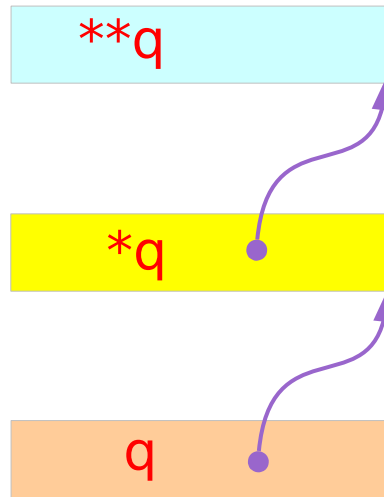


Types

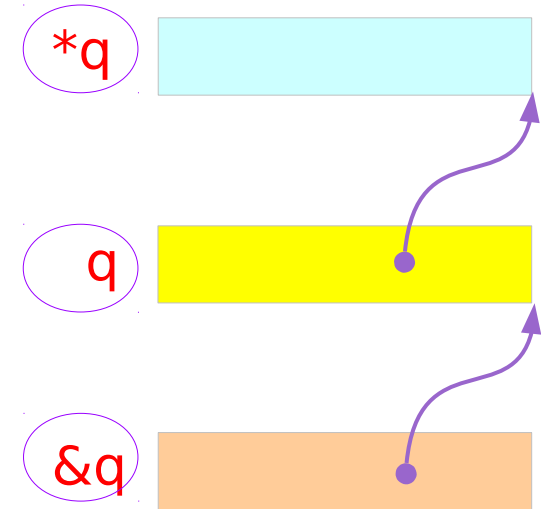
Interpretation of Pointers – Variables and addresses



Types



Variables

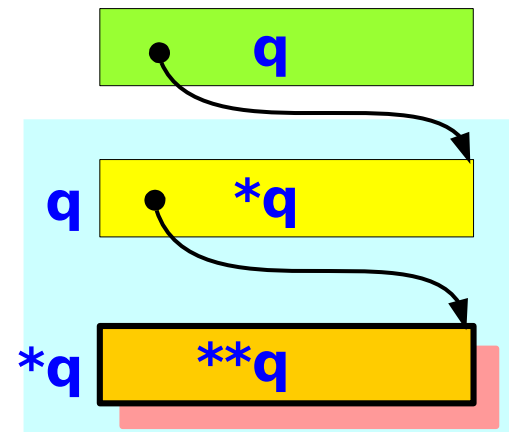
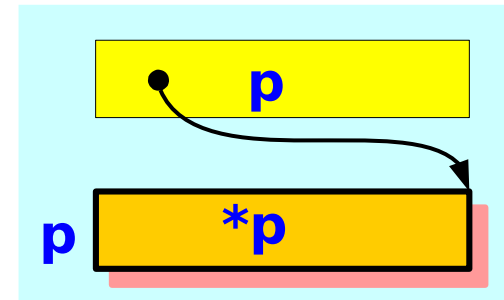


Addresses

Single and Double Pointer Examples (1)

```
int a ;  
int *p ;  
int **q ;
```

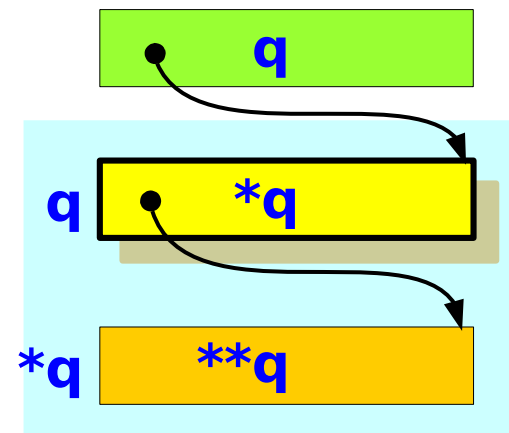
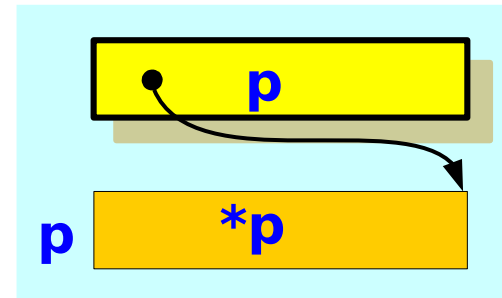
a, *p, and **q:
int variables



Single and Double Pointer Examples (2)

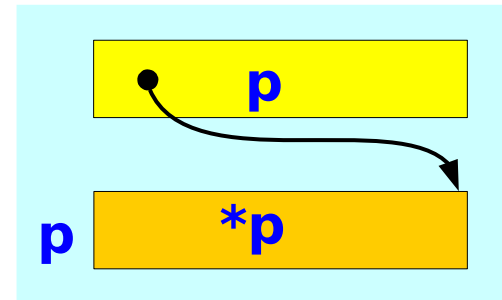
```
int    a ;  
int *  p ;  
int *  *q ;
```

p and ***q** :
int pointer variables

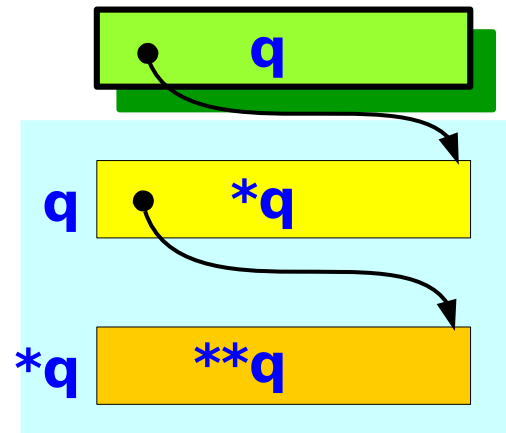


Single and Double Pointer Examples (3)

```
int    a ;  
int    *p ;  
int ** q ;
```

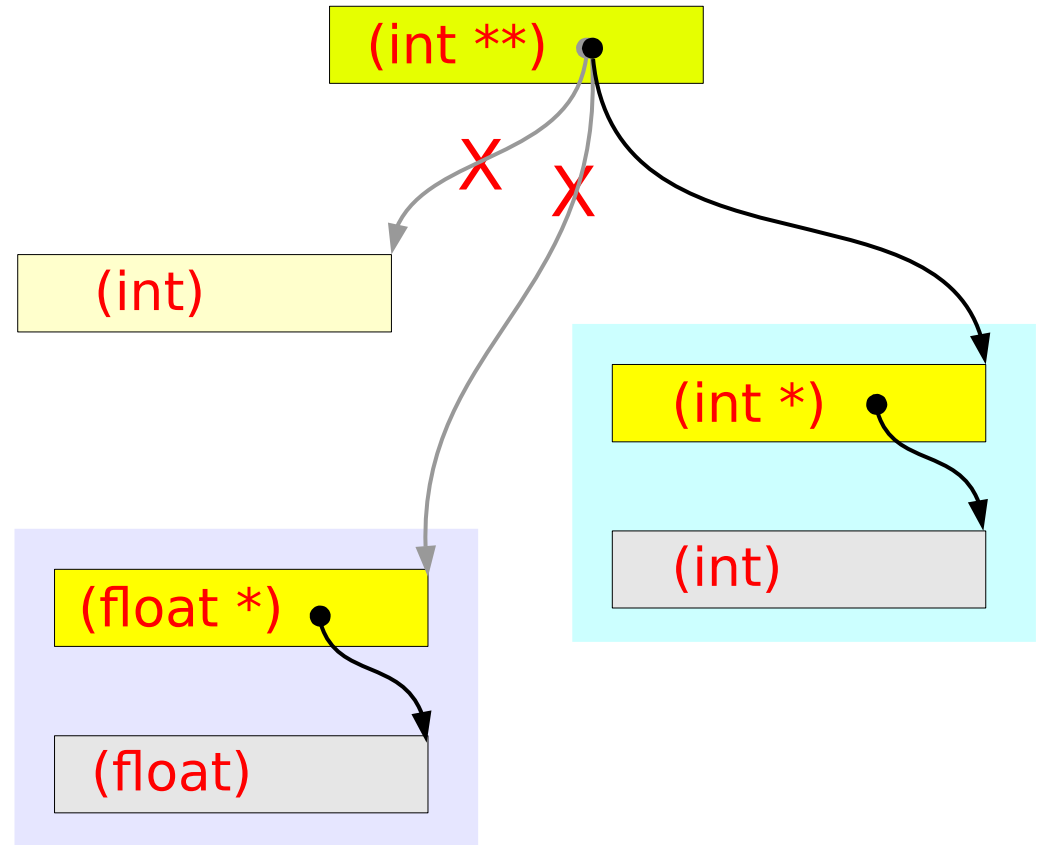


q :
double int pointer variables



Integer Pointer Examples (6)

```
int    a;  
int *  p;  
int ** q;
```




Variable Declarations

```
int a ;
```

&a 

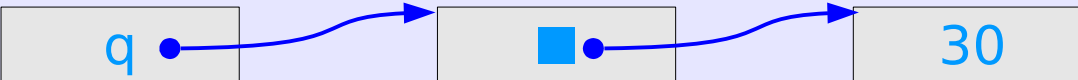
The variable `a` holds an **integer data**

```
int * p ;
```

&p 

The **pointer** variable `p` holds an **address**,
at this address **an integer** is stored

```
int ** q ;
```

&q 

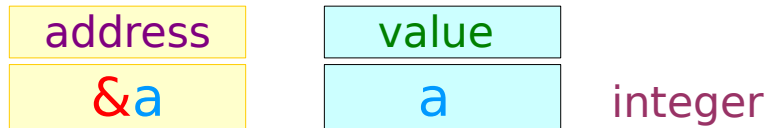
The **pointer** variable `q` holds an **address**,
where **another address** is stored,
where an **integer data** is stored

Access Data Via Pointer Variables (1)

`int a ;`

`&a` `a = 100`

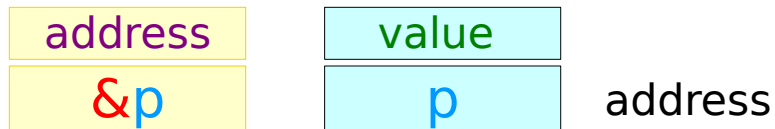
Direct Access



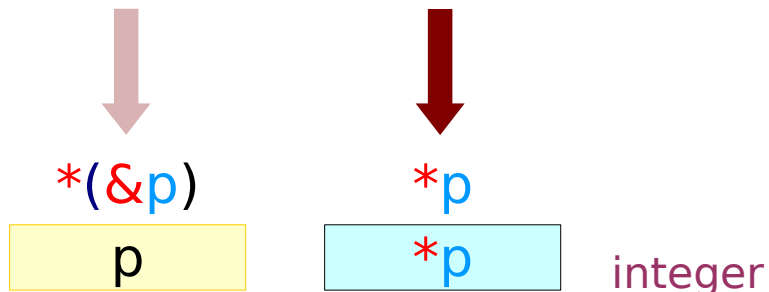
`int * p ;`

`&p` `p` `*p = 200`

Indirect Access

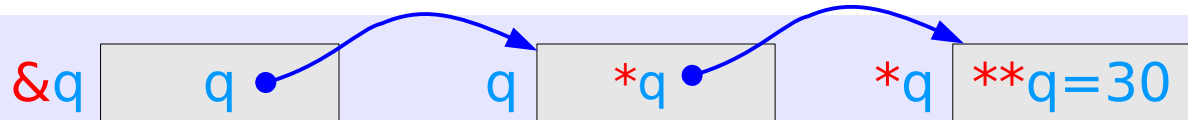


Dereference Operator `*`
the content of the pointed location

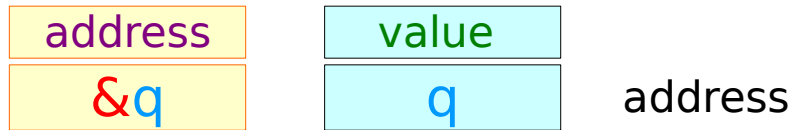


Access Data Via Pointer Variables (2)

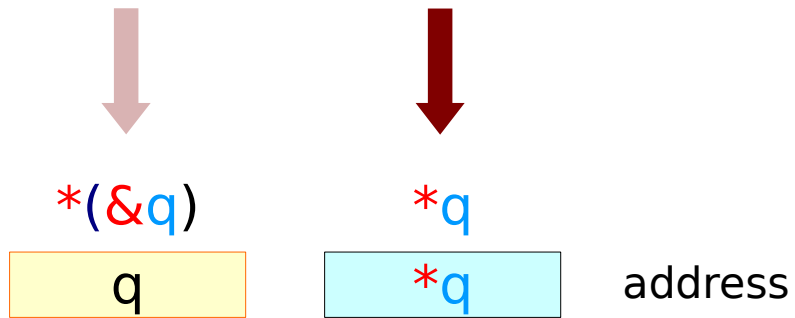
`int ** q ;`



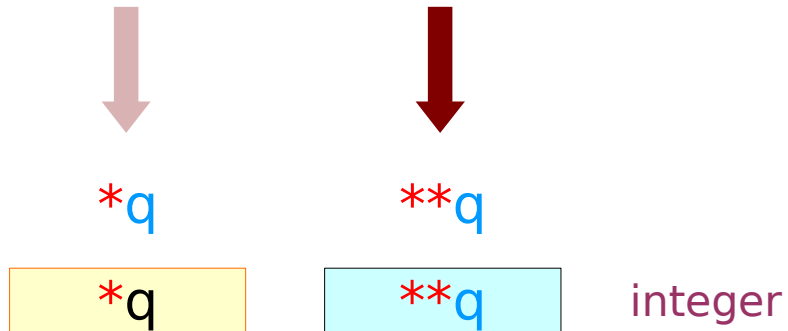
Double Indirect Access



Dereference Operator *
the content of the pointed location



Dereference Operator *
the content of the pointed location



Access Data Via Pointer Variables (3)

`int a ;`

`&a`

`a =100`

Direct Access

address

`&a`

value

`a`

integer

`int * p ;`

`&p`

`p`

`p`

`*p=200`

Indirect Access

address

`&p`

`p`

value

`p`

address

`*p`

integer

Dereference Operator `*`

the content of the pointed location

`int ** q ;`

`&q`

`q`

`q`

`*q`

`*q`

`*q`

`**q=30`

Double Indirect Access

address

`&q`

`q`

`*q`

value

`q`

`*q`

`**q`

address

address

integer

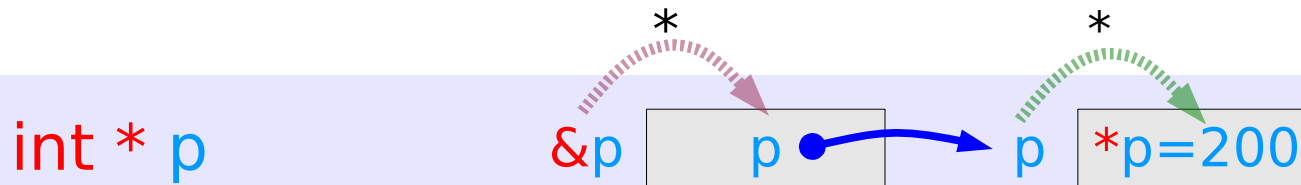
Dereference Operator `*`

the content of the pointed location

Access Data Via Pointer Variables (4)

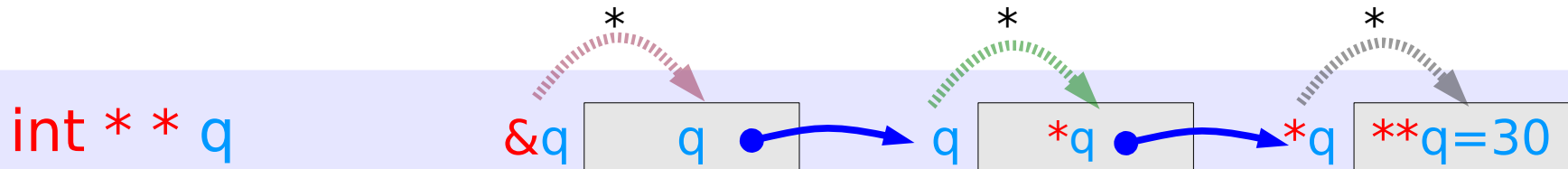


$$*(\&a) = a$$



$$*(\&p) = p$$

$$*(p) = *p$$



$$*(\&q) = q$$

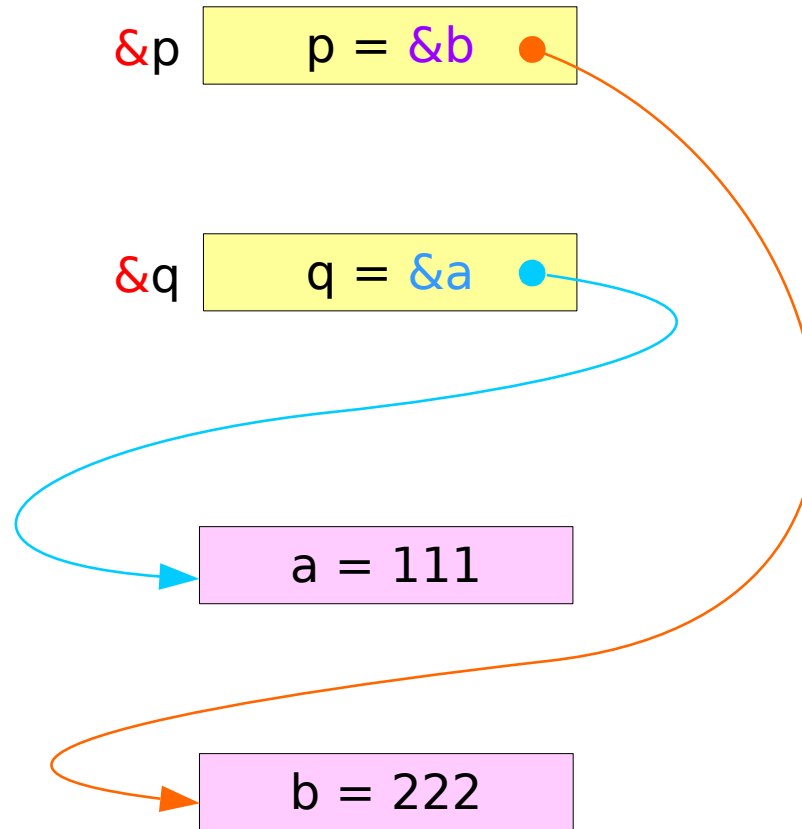
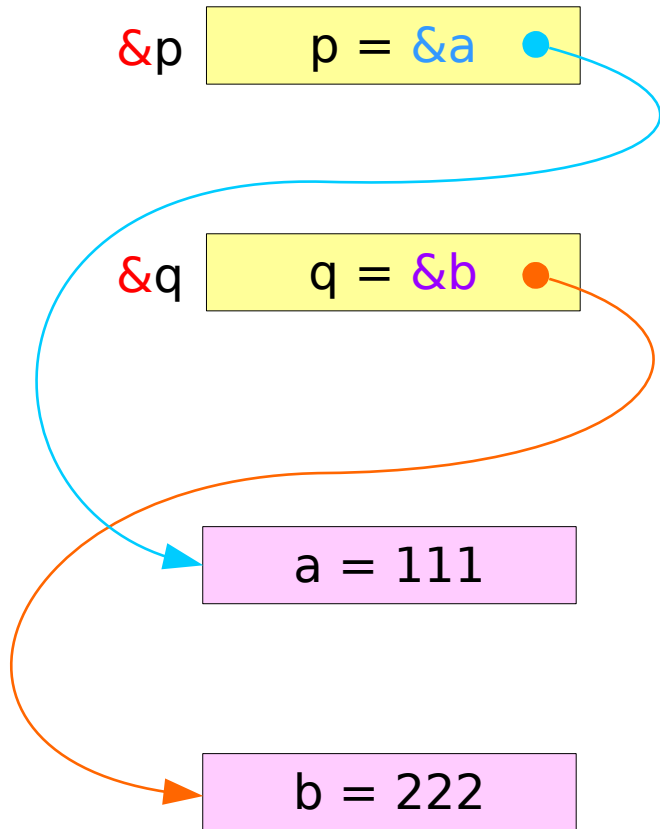
$$*(q) = *q$$

$$**(*q) = **q$$

Swapping pointers

- pass by reference
- double pointers

Swapping integer pointers



Swapping integer pointers



```
int *p, *q;
```

```
swap_pointers( &p, &q );
```

function call

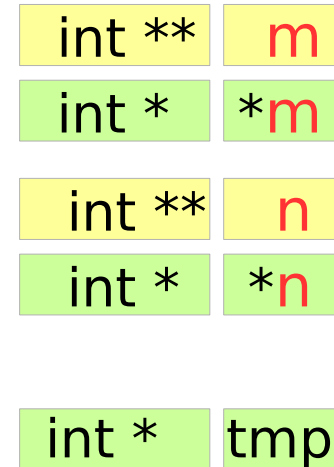
```
swap_pointers( int **, int ** );
```

function prototype

Pass by integer pointer reference

```
void swap_pointers (int **m, int **n)
{
    int* tmp;

    tmp = *m;
    *m = *n;
    *n = tmp;
}
```



```
int a, b;
int *p, *q;    p=&a, q=&b;
...
swap_pointers( &p, &q );
```

Array of Pointers

Array of Pointers (1)

```
int    a [4];
```

```
int *  b [4];
```

Array name **a** holds the starting address

int **a** **[4]**

No. of elements = 4

Type of each element

Array name **b** holds the starting address

int * **b** **[4]**

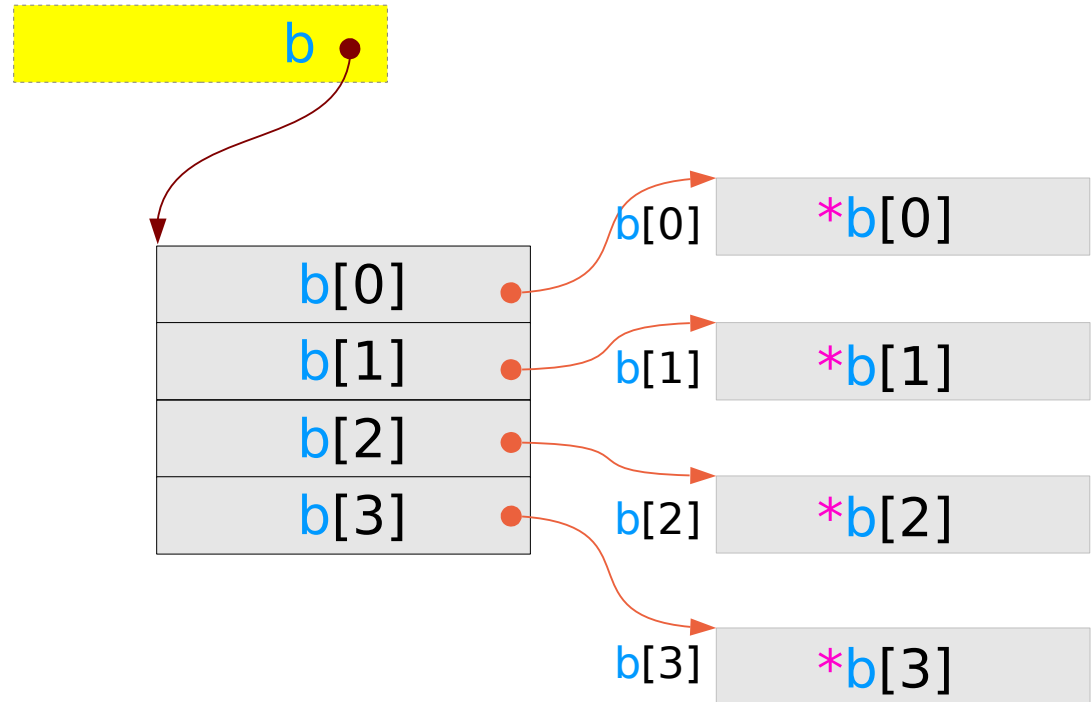
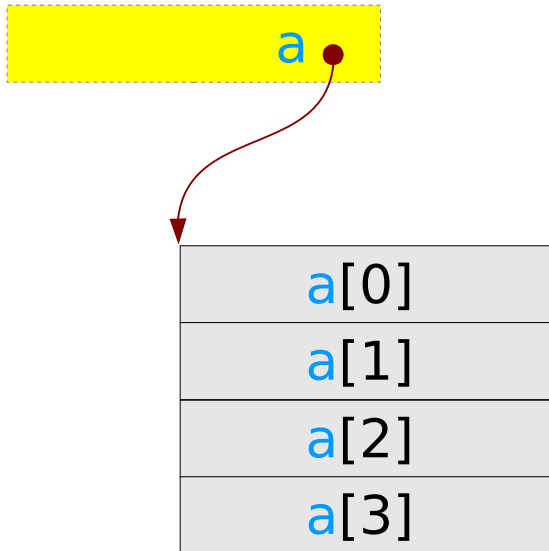
No. of elements = 4

Type of each element

Array of Pointers (2)

```
int    a [4];
```

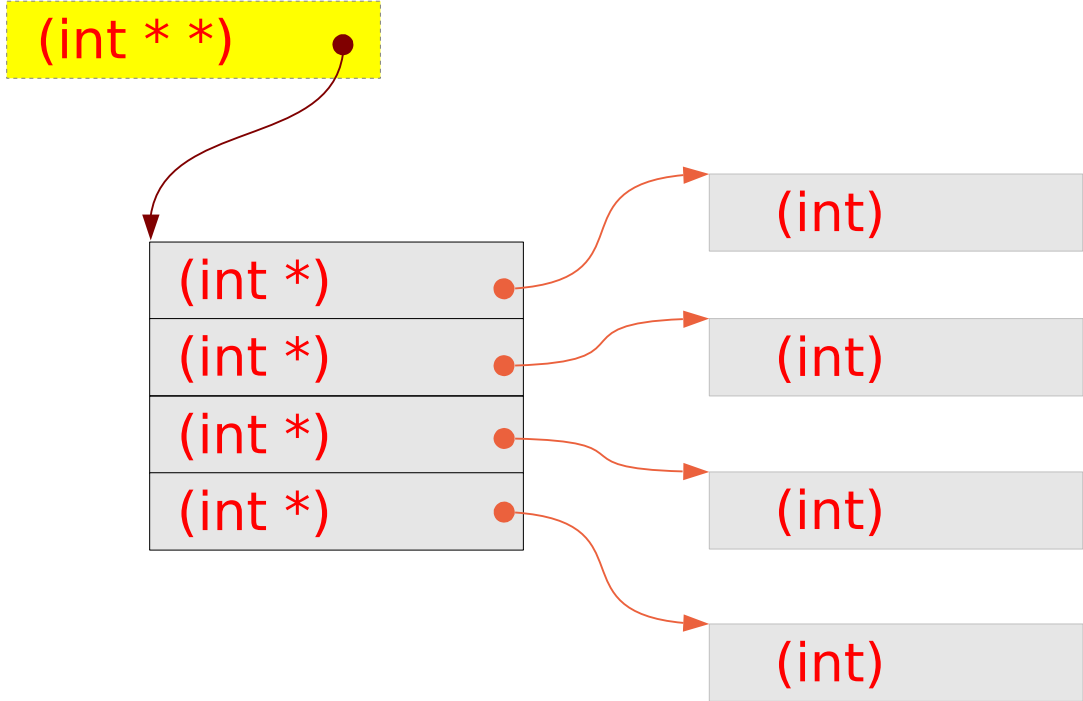
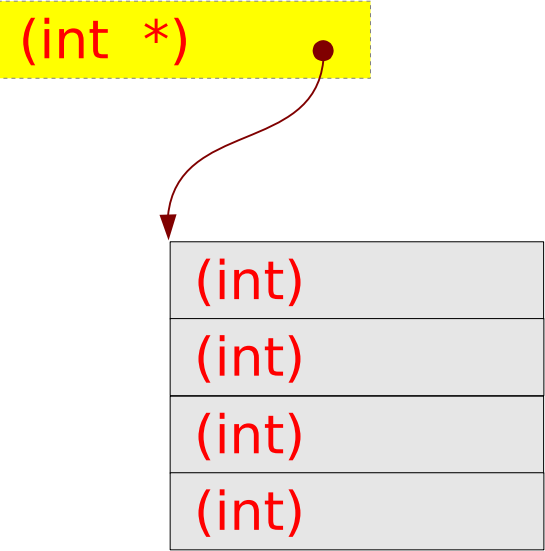
```
int *  b [4];
```



Array of Pointers (3)

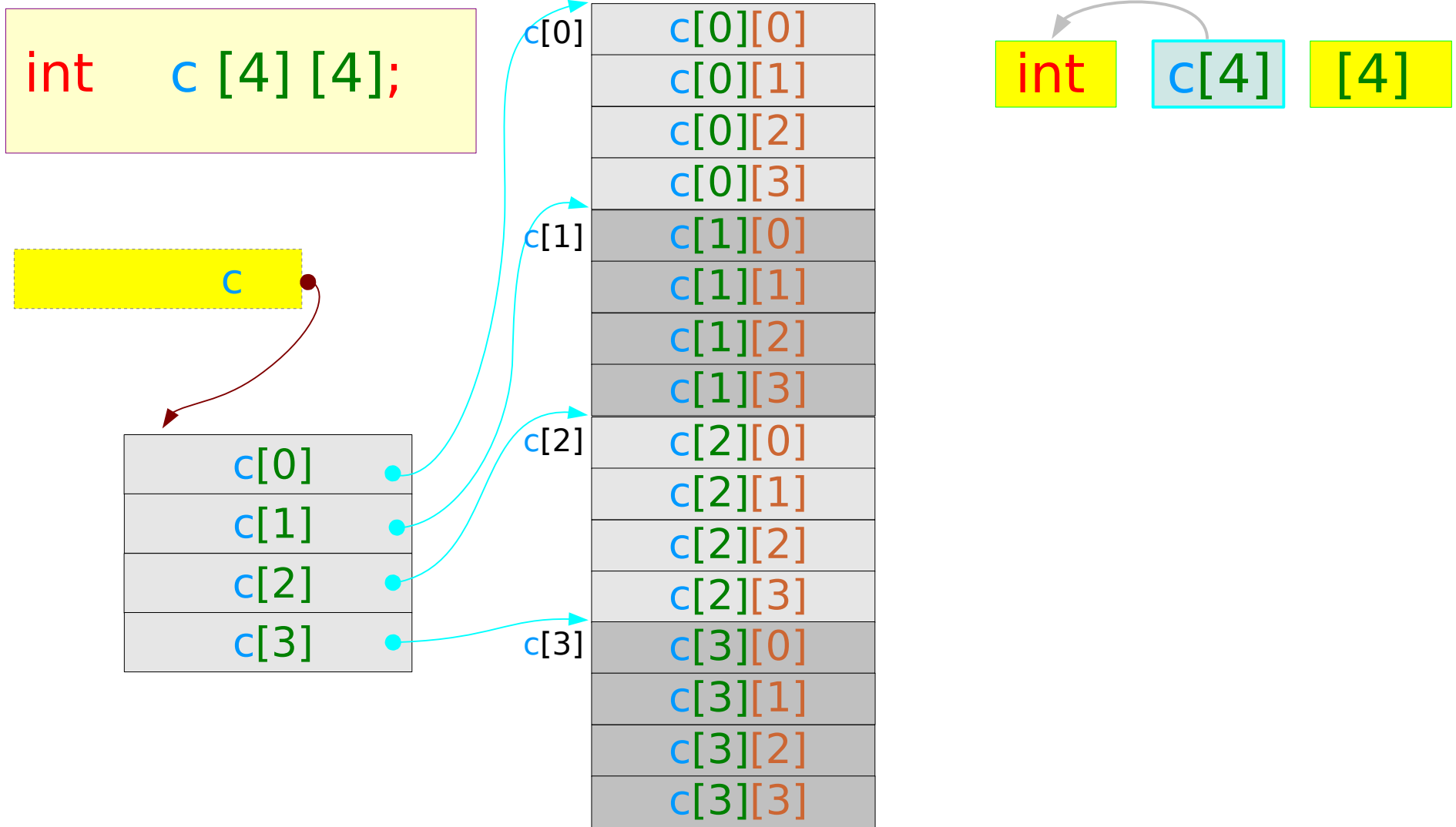
```
int a[4];
```

```
int * b[4];
```

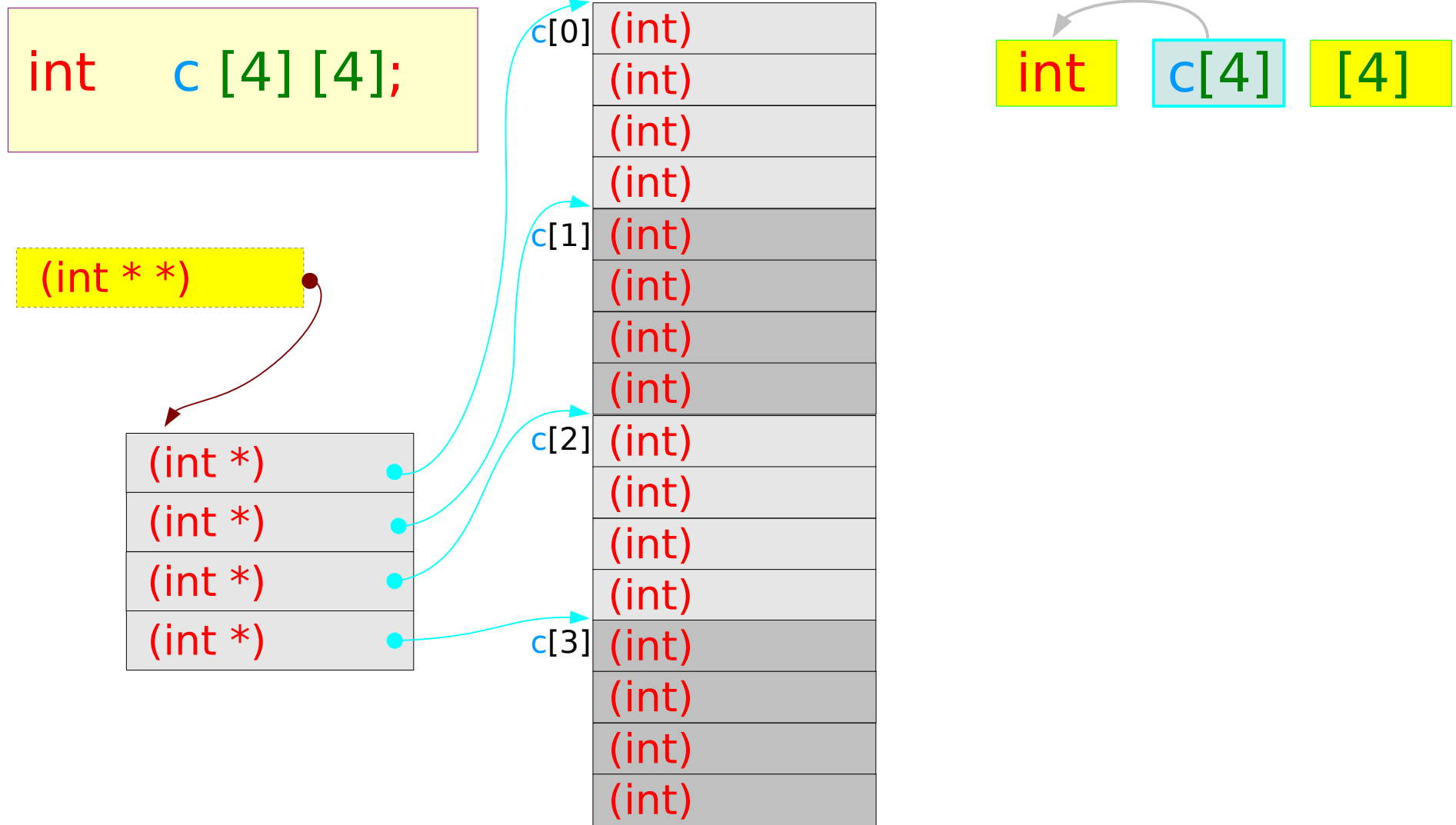


2-d Arrays

A 2-D Array



A 2-D Array



A 2-D Array via a double pointer

```
int c [4] [4];
```

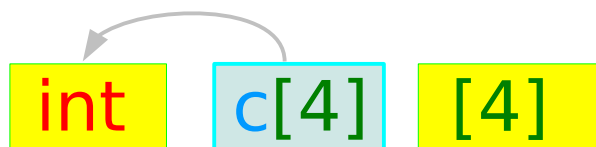
$(c [i]) [j] \rightarrow$

$(* (c+i)) [j] \rightarrow$

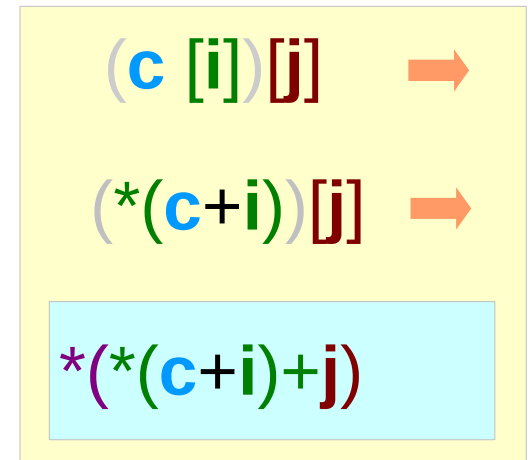
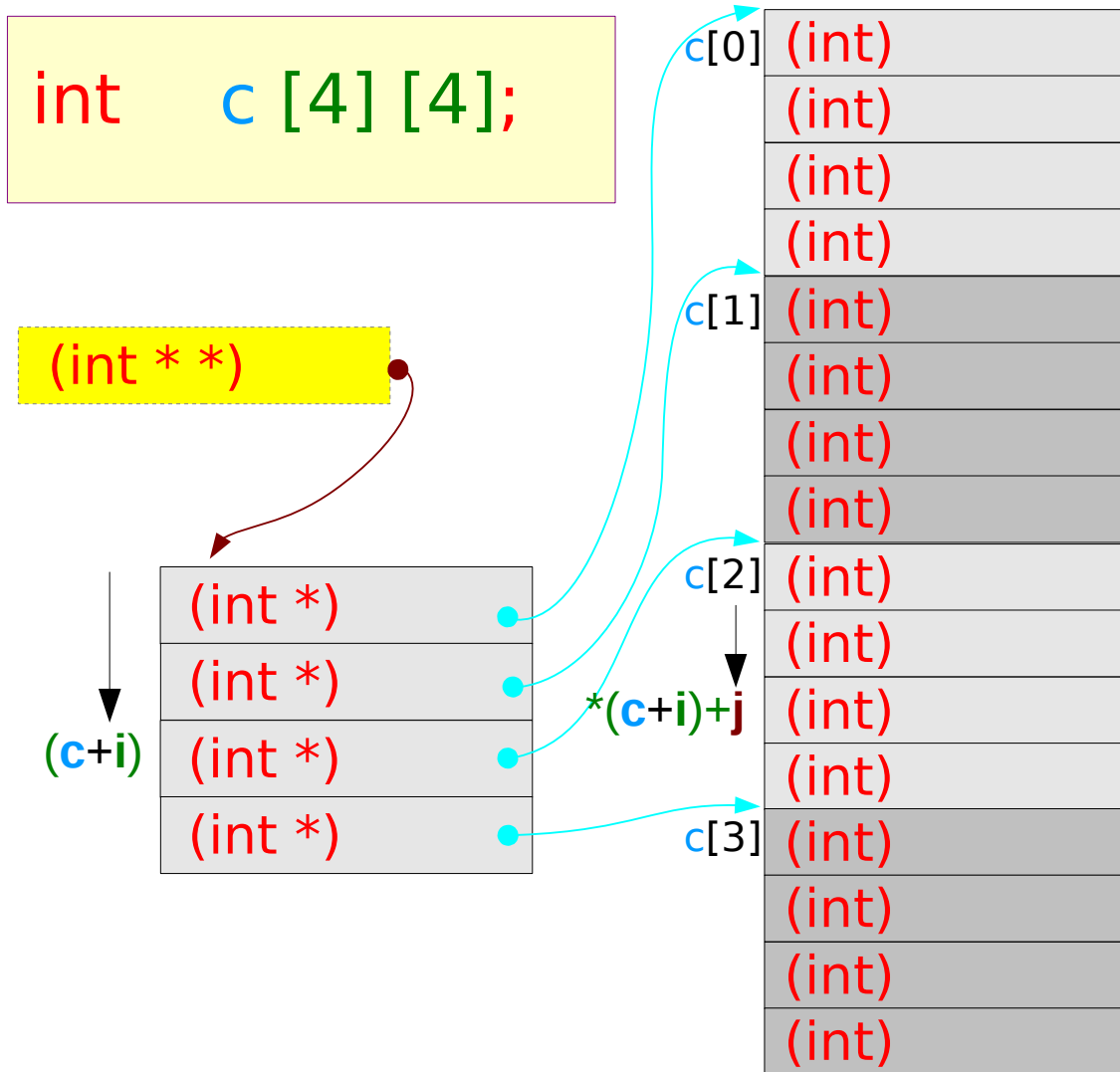
$* (* (c+i) + j)$

$(c [i]) = (* (c+i))$

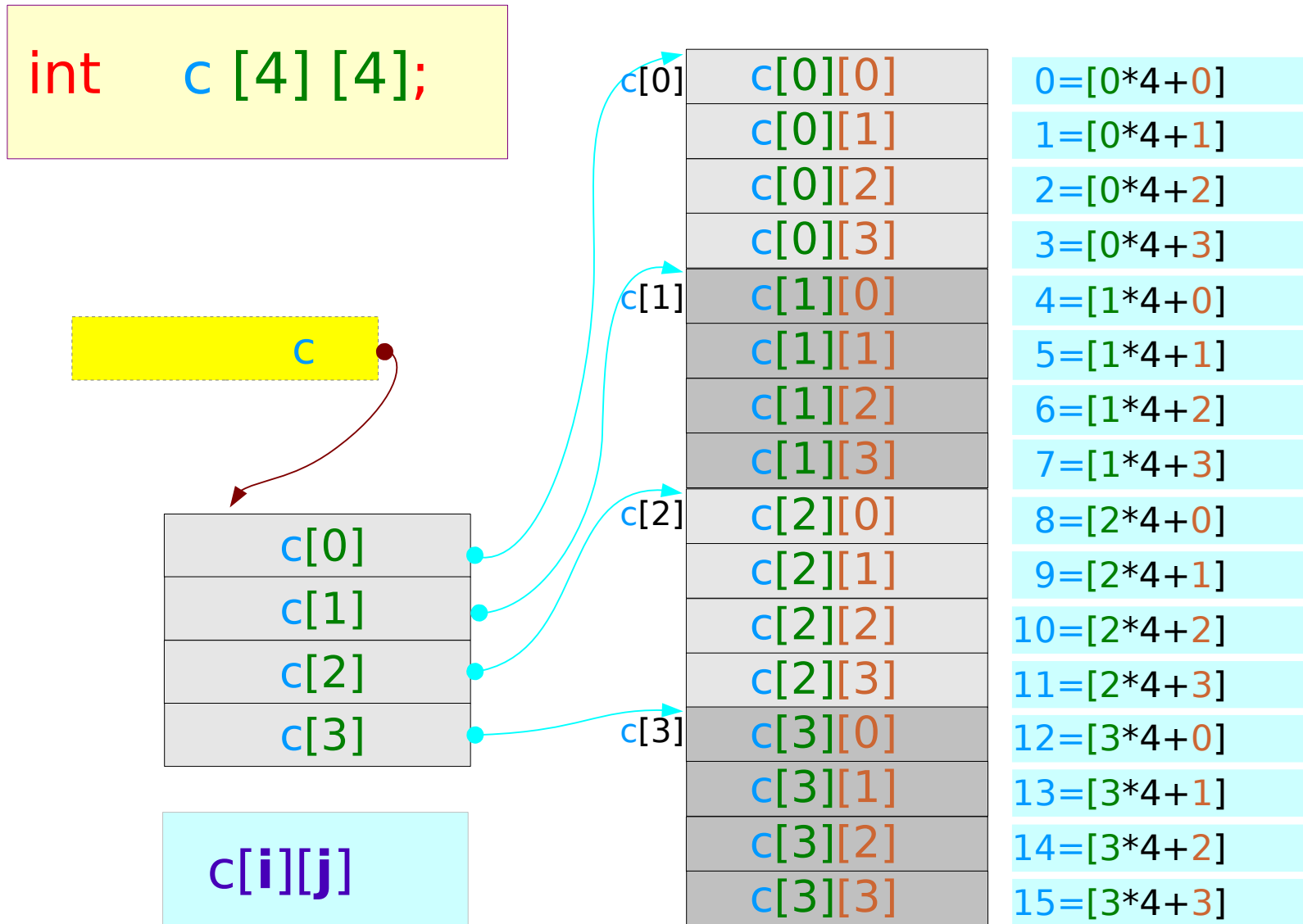
$(_) [j] = * (_ + j)$



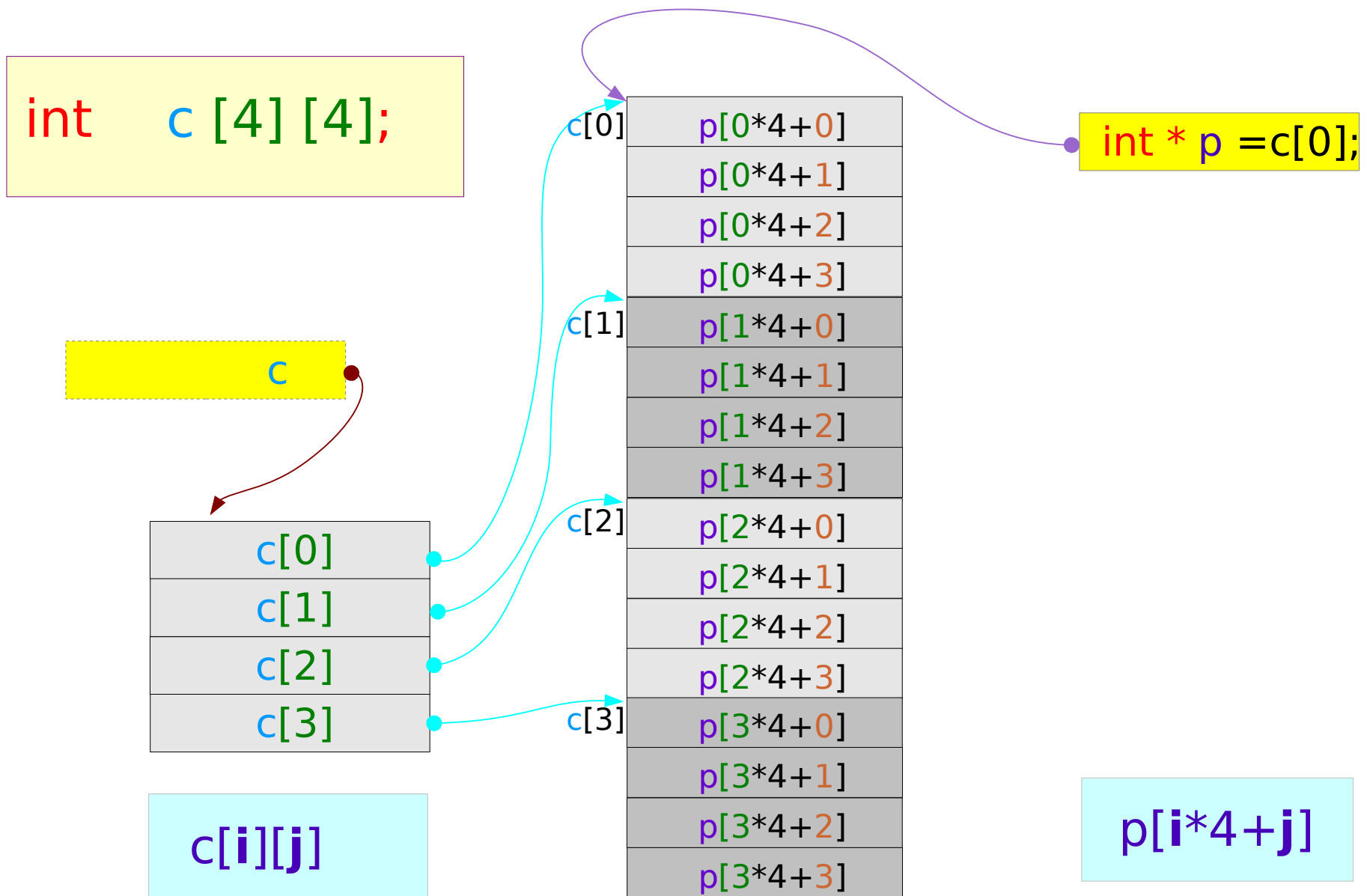
A 2-D Array



A 2-D array via a single pointer



A 2-D array via a single pointer



2-D Array Dynamic Memory Allocation (1)

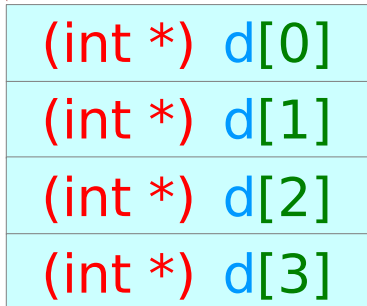
```
int ** d ;
```

```
d = (int **) malloc (4 * size of (int *));
```

```
for (i=0; i<4; ++i)
```

```
    d[i] = (int *) malloc(4 * sizeof(int));
```

(int **) d •



2-D Array Dynamic Memory Allocation (2)

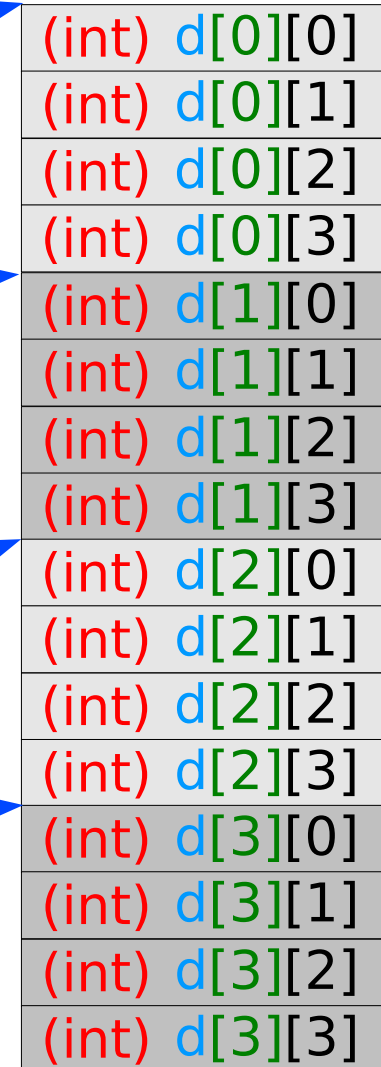
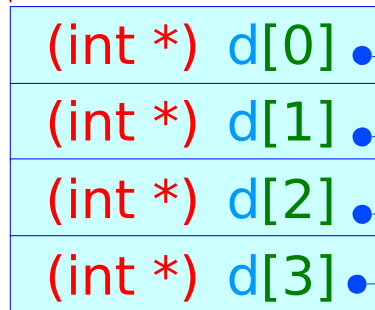
```
int ** d ;
```

```
d = (int **) malloc (4 * size of (int *));
```

```
for (i=0; i<4; ++i)
```

```
    d[i] = (int *) malloc(4 * sizeof(int));
```

&d (int **) d •



Pointer to Arrays

Pointer to array (1)

```
int    a [4];
```

```
(int []) a •
```

(int) a[0]
(int) a[1]
(int) a[2]
(int) a[3]

```
int    a    [4]  
      ↑  
int    (*p) [4]
```

pointer to the array of 4 elements

```
{ int m;      an integer variable  
  int *n;    a pointer variable  
  int func(int a, int b);  a prototype  
  int (*fp)(int a, int b); a function's type  
  int *fp(int a, int b);  function pointer
```

Pointer to array (2)

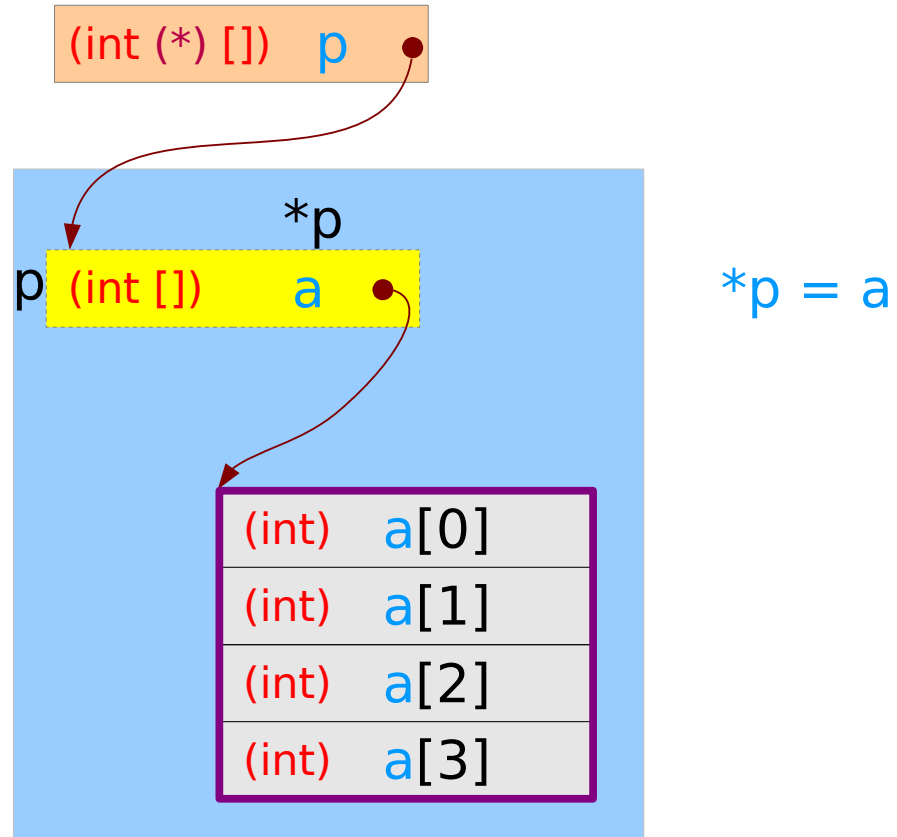
```
int (*p) [4] ;
```

↕

```
int a [4]
```

```
(*p) = a  
↓  
&(*p) = &a  
↓  
p = &a
```

sizeof(p) = 4 bytes
sizeof(*p) = 16 bytes



an array with 4 integer elements

Pointer to array (3)

```
int (*p) [4] ;
```

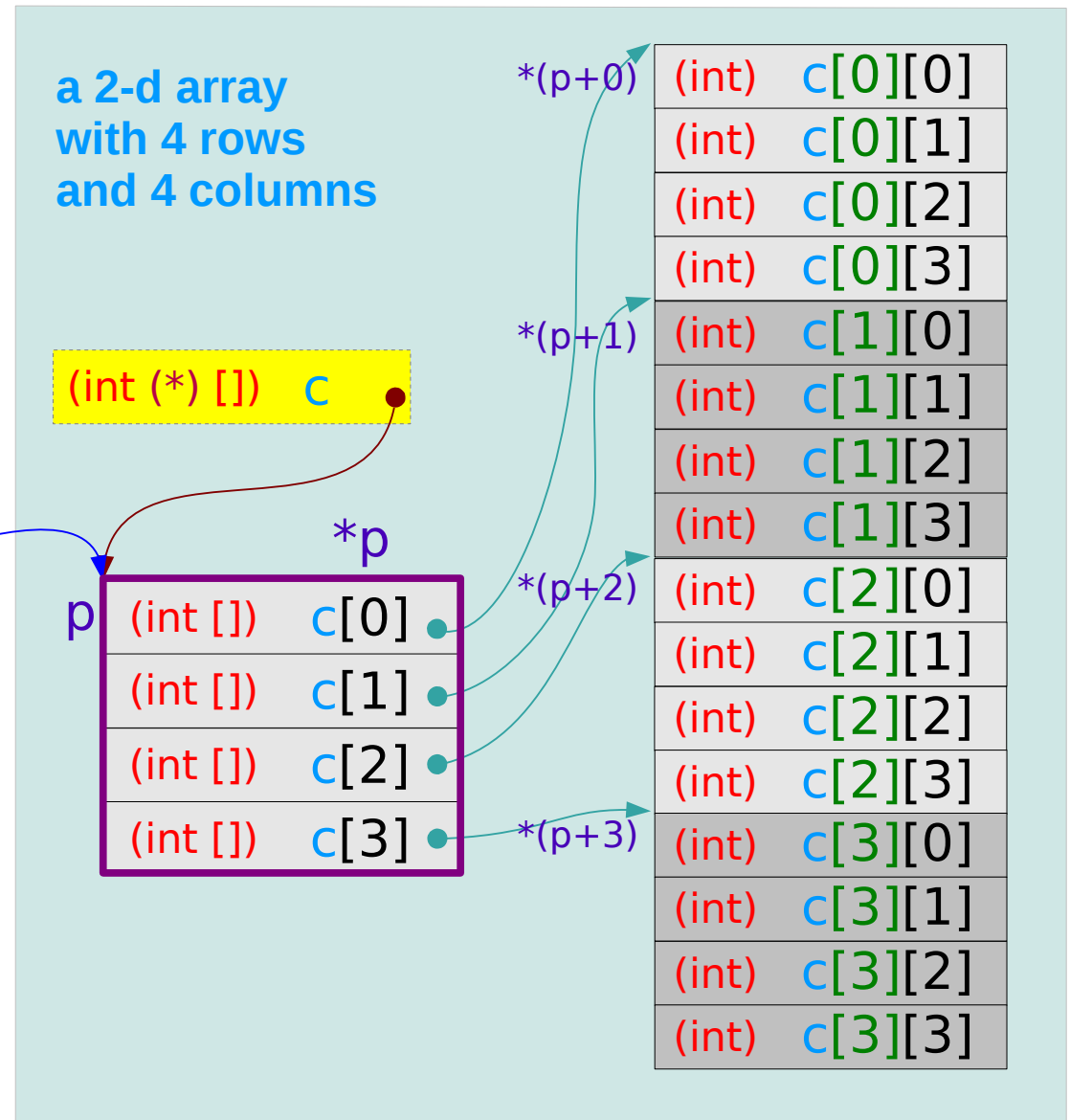


```
int c[4] [4]
```

```
&p (int (*) []) p
```

$p = c$

```
(*p) [i][j];
```



Pointer to array (4)

```
int c [4][4];  
int (*p) [4];
```

```
p = c;
```

```
func(p, ... );
```

```
void func(int (*x)[4], ... )  
{
```

```
    x[r][c] =
```

```
}
```

```
void func(int x[ ][4], ... )  
{
```

```
    x[r][c] =
```

```
}
```

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