Applications of Pointers (1A)

Young Won Lim
3/10/18

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.

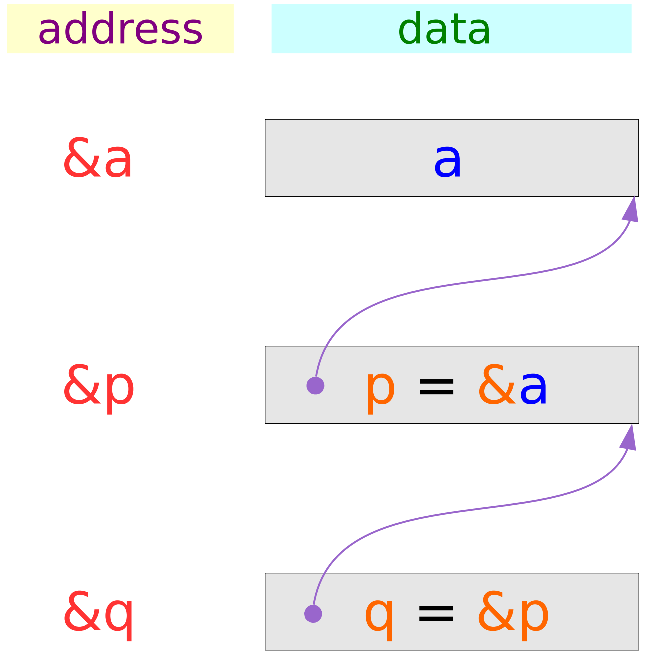
This document was produced by using LibreOffice.

Variables and their addresses

	address	data
`int    a;`	&a	a
`int  * p;`	&p	p
`int **q;`	&q	q

Initialization of Variables

```
int     a;

int   * p = &a;

int **q = &p;
```



address

data

&a

a

&p

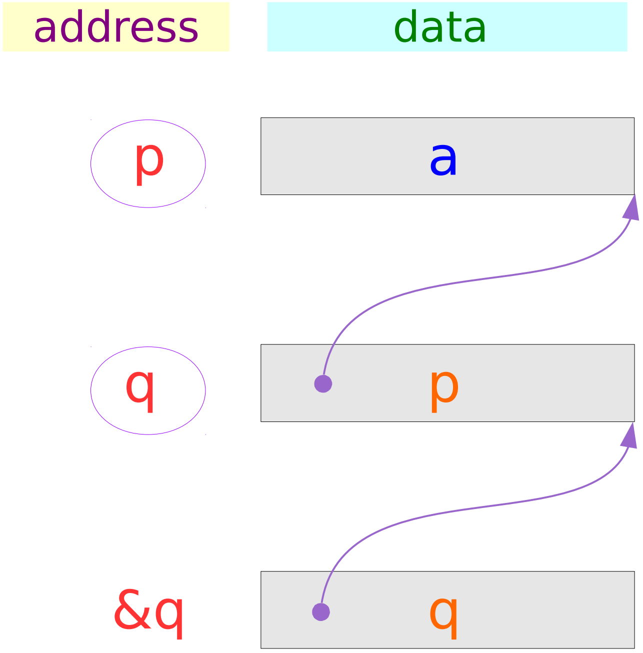
p = &a

&q

q = &p

Pointed addresses : p, q

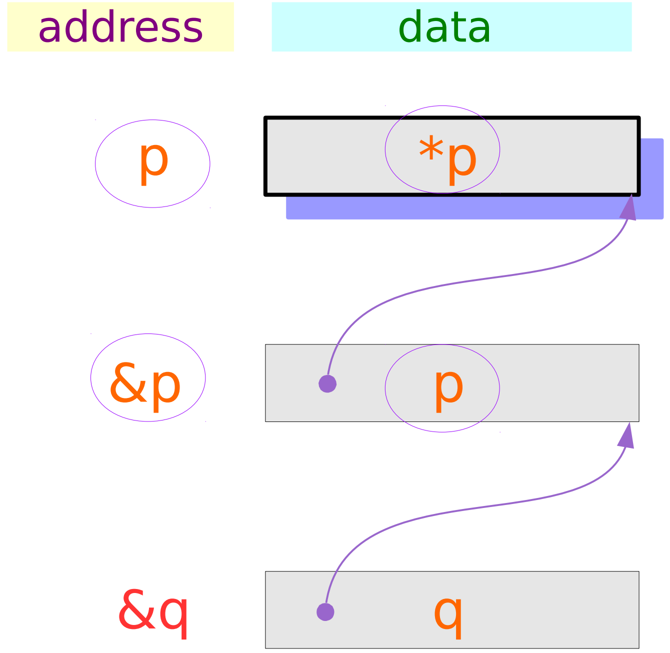
```
int    a;

int  * p = &a;

int **q = &p;
```



p = &a
q = &p

Dereferenced Variables : *p

```
int     a;

int  * p = &a;

int **q = &p;
```



$*p \equiv a$

Dereferenced Variables : *p

```
int    a;

int  * p = &a;

int **q = &p;
```

Address
Assignment

Variables
with the same address

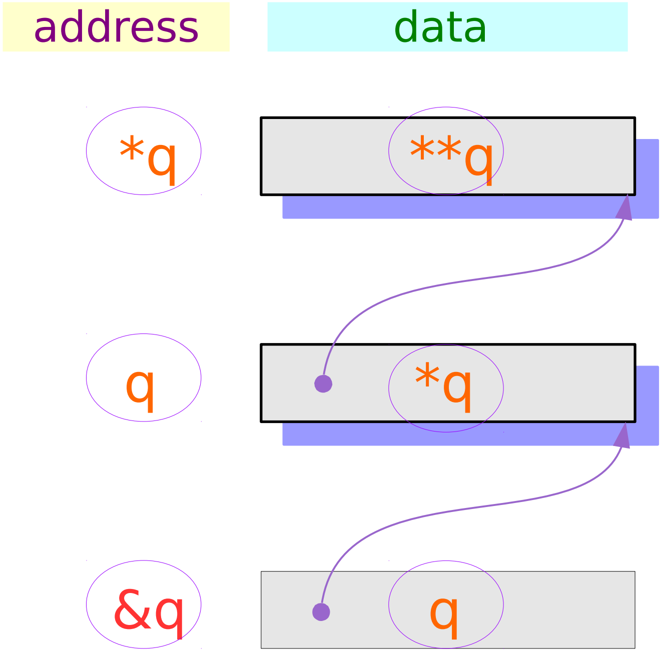
p = &a ⇨ *p ≡ a

```
  p  ≡    &a
*(p) ≡ *(&a)
* p  ≡     a
```

Relations after
address assignment

Dereferenced Variables : *q, **q

```
int     a;

int  * p = &a;

int **q = &p;
```



**q ≡ a

*q ≡ p

Dereferenced Variables : *q, **q

```
int    a;

int  * p = &a;

int **q = &p;
```

Address Assignment

Variables with the same address

p = &a ⇒ *p ≡ a

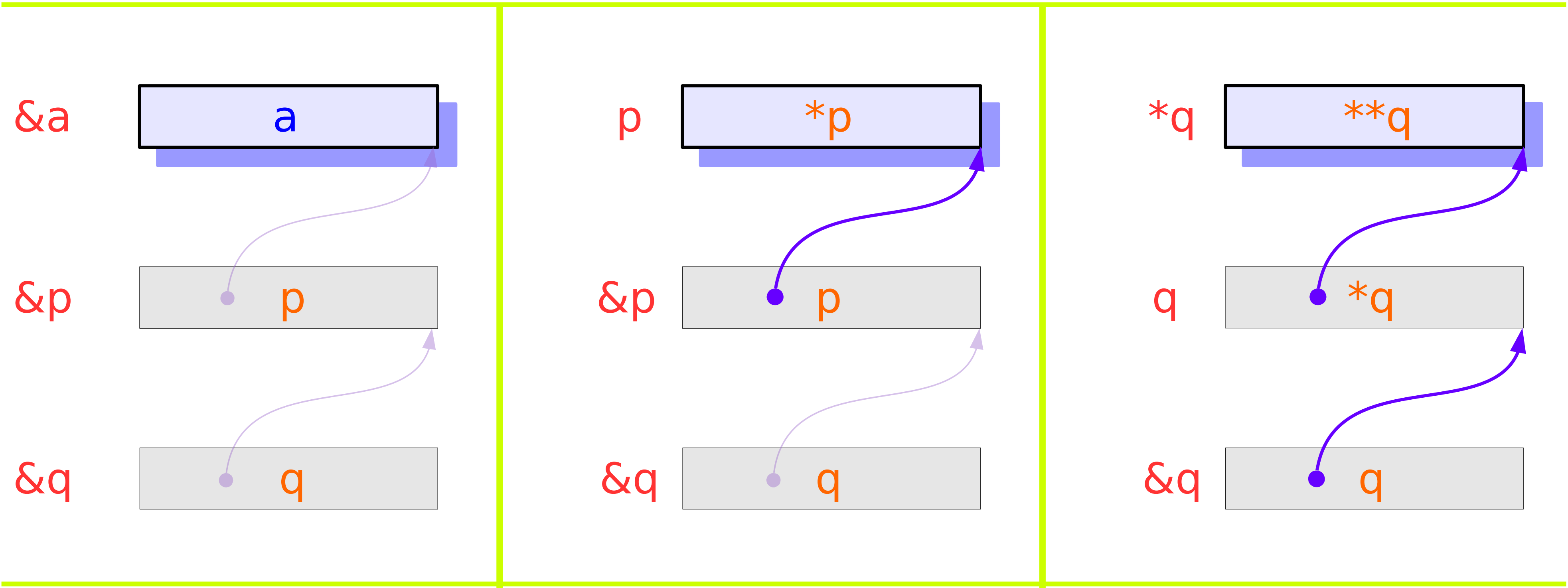
q = &p ⇒ *q ≡ p

⇒ **q ≡ a

```
  q   ≡    &p
*(q) ≡ *(&p)
* q   ≡      p
**q  ≡     *p
**q  ≡      a
```

Relations after address assignment

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

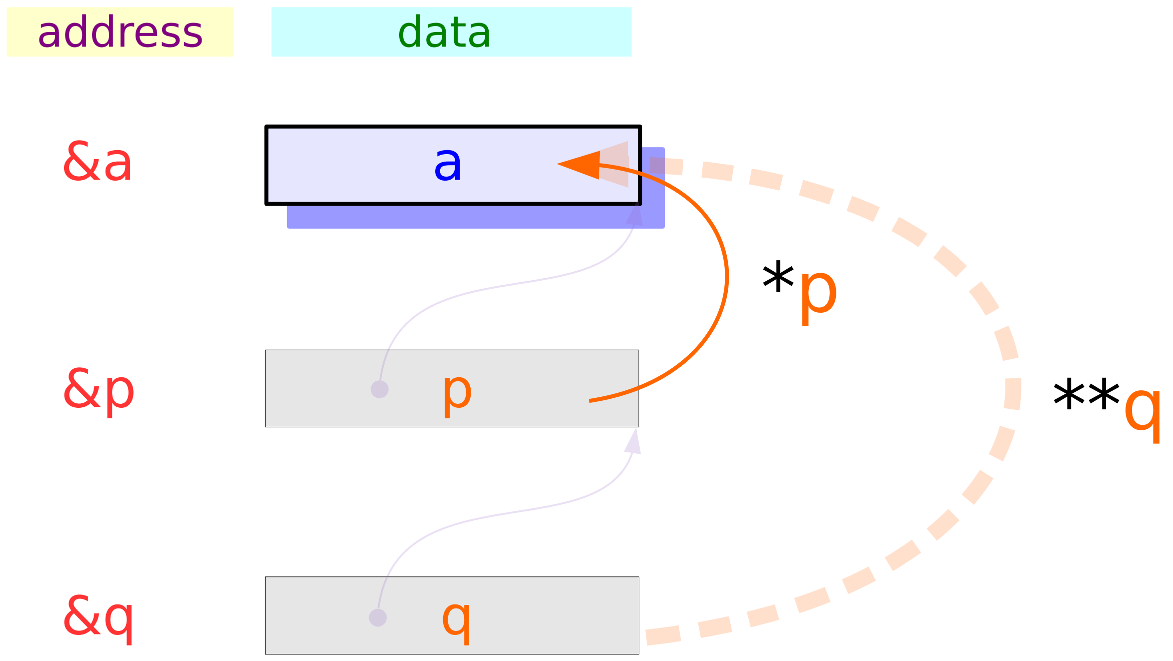


a

*p ≡ a

**q ≡ a

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;
```

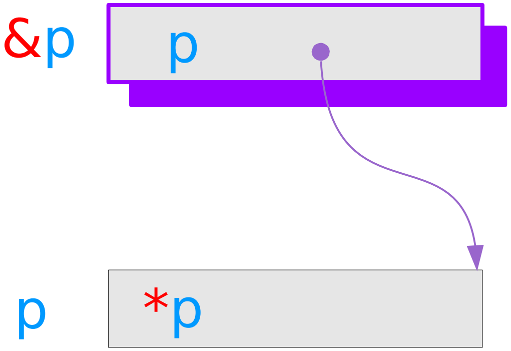
p holds an *address*

$\underbrace{\texttt{int *}}_{\textit{pointer to int}}\ \underbrace{\texttt{p;}}$

p holds an *address* of a int type data

$\underbrace{\texttt{int}}_{\textit{int}}\ \underbrace{\texttt{* p;}}$

*p holds an *integer*



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;

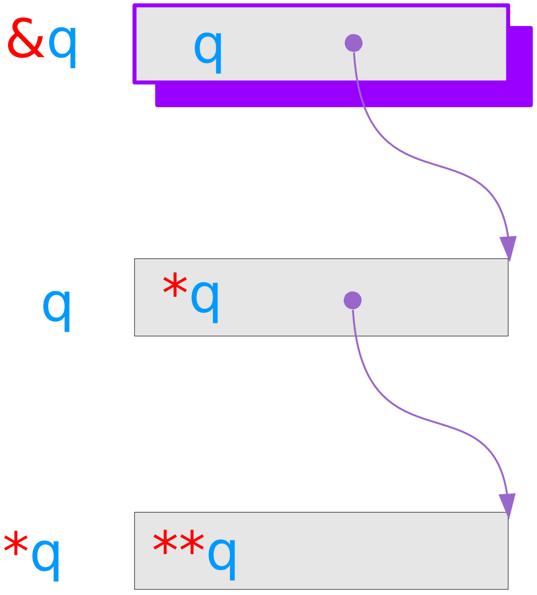
pointer to int

*q holds an *address* of a int type variable

int **q;

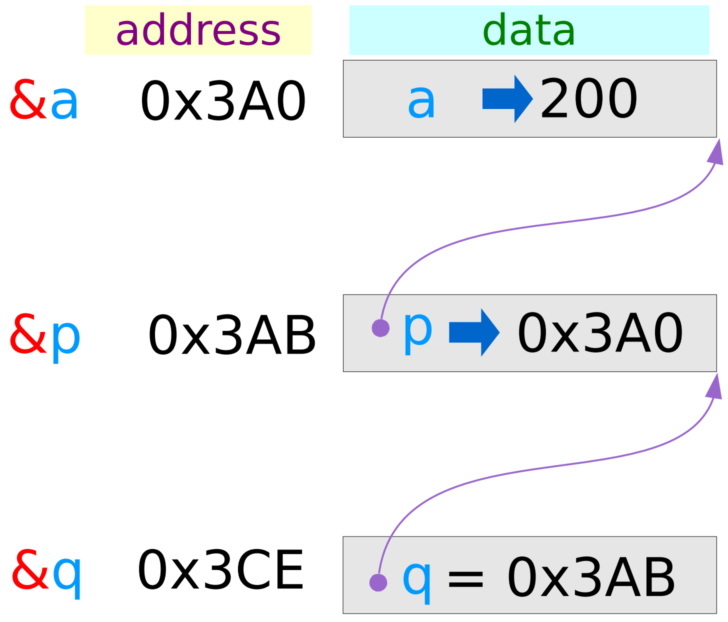
int

**q holds an *integer*



Pointer Variables Examples

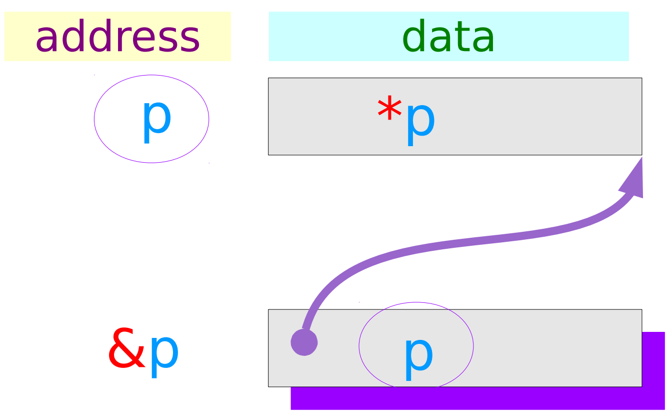
```
int      a;

int *    p = & a;

int **   q = & p;
```



	address	data
&a	0x3A0	a → 200
&p	0x3AB	p → 0x3A0
&q	0x3CE	q = 0x3AB

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

Pointer Variable **p**



address

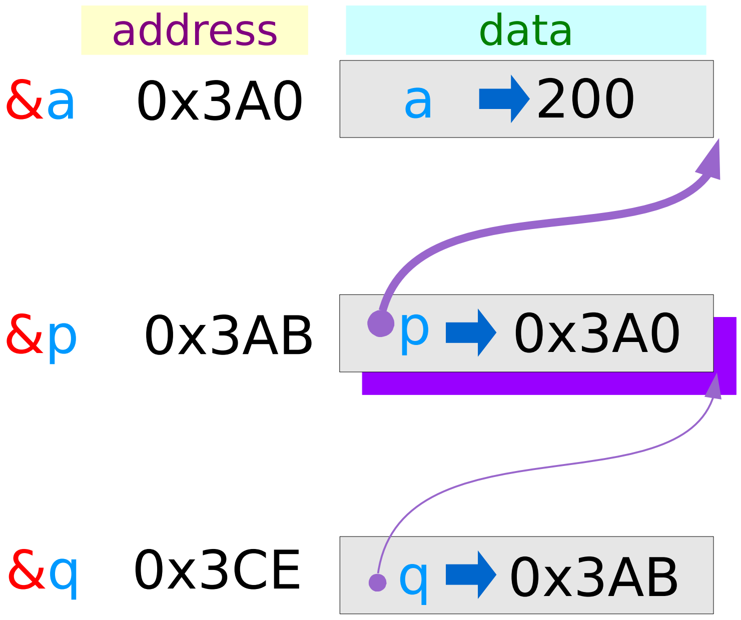
data

p

*p

&p

p



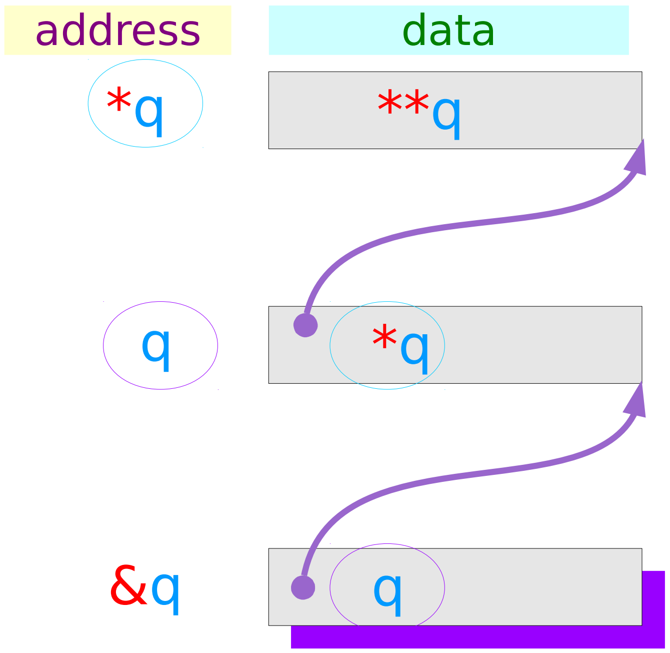
	address	data
&a	0x3A0	a → 200
&p	0x3AB	p → 0x3A0
&q	0x3CE	q → 0x3AB

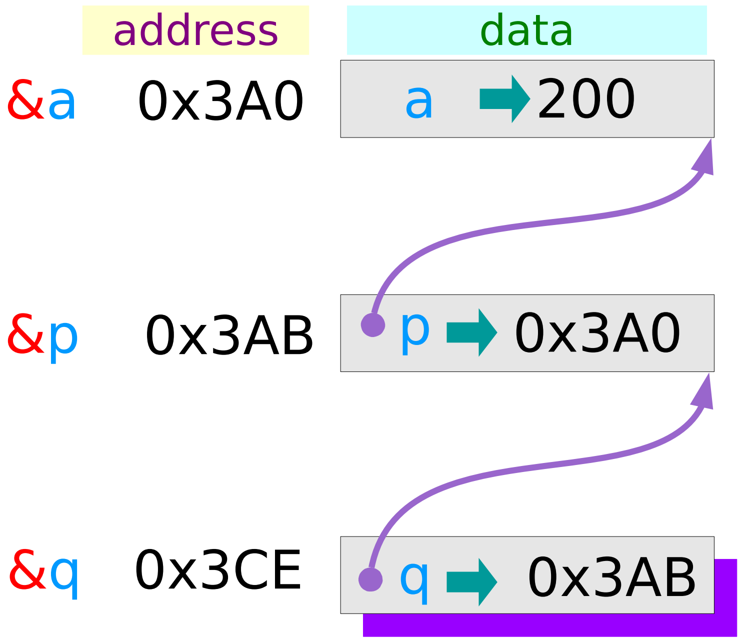
&p → 0x3AB

p → 0x3A0

*p → 200

Pointer Variable **q**





address		data
&a	0x3A0	a → 200
&p	0x3AB	p → 0x3A0
&q	0x3CE	q → 0x3AB

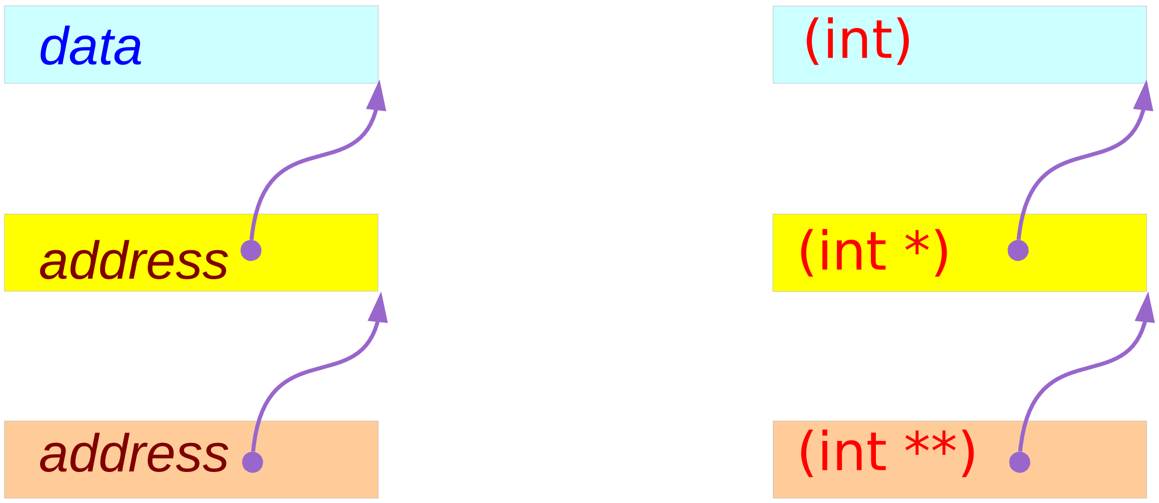
&q → 0x3CE

q → 0x3AB

*q → 0x3A0

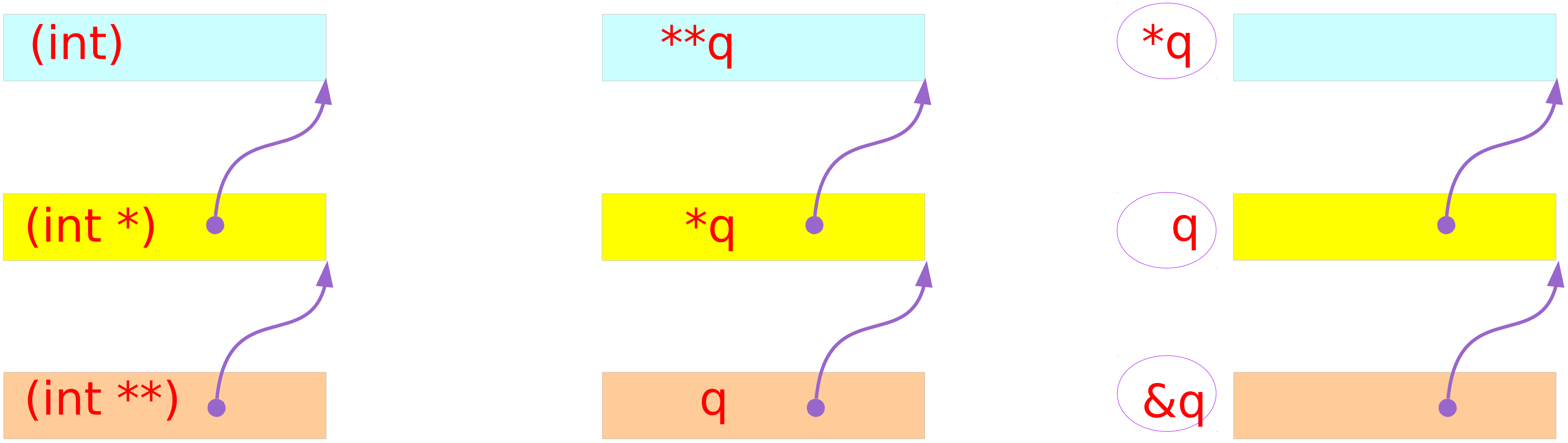
**q → 200

Interpretation of Pointers – Types



Types

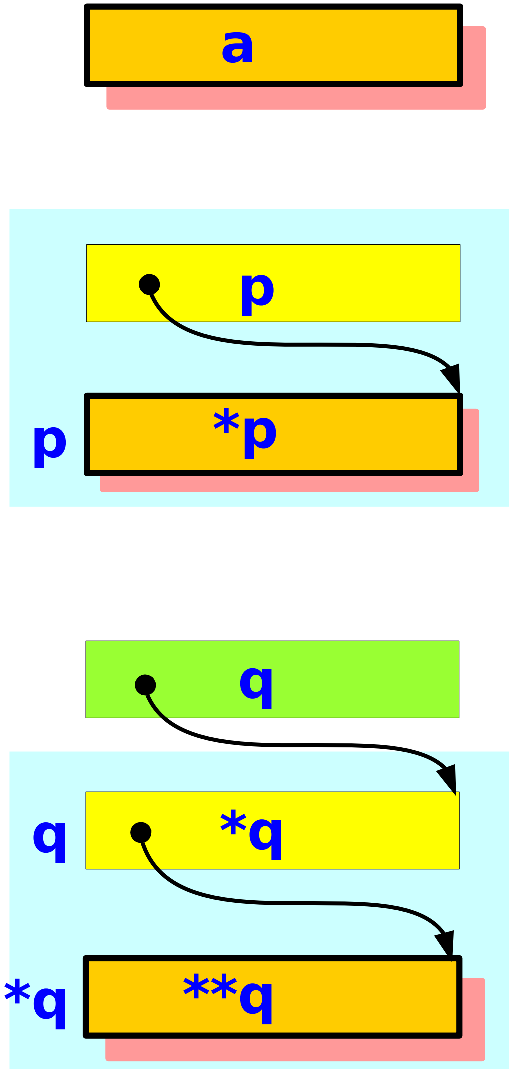
Interpretation of Pointers – Variables and addresses



Types **Variables** **Addresses**

Single and Double Pointer Examples (1)

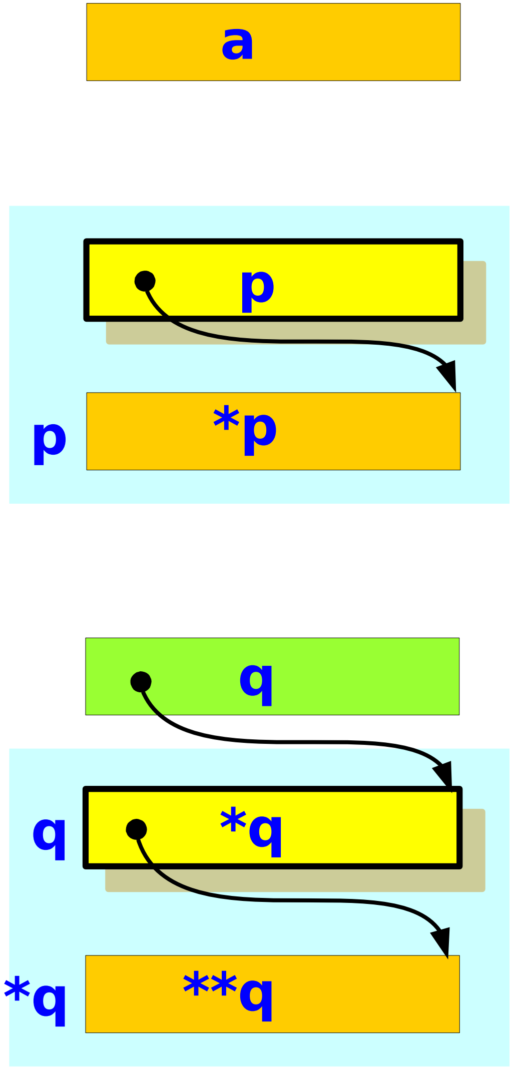
```
int    a ;
int   *p ;
int  **q ;
```



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

Single and Double Pointer Examples (2)

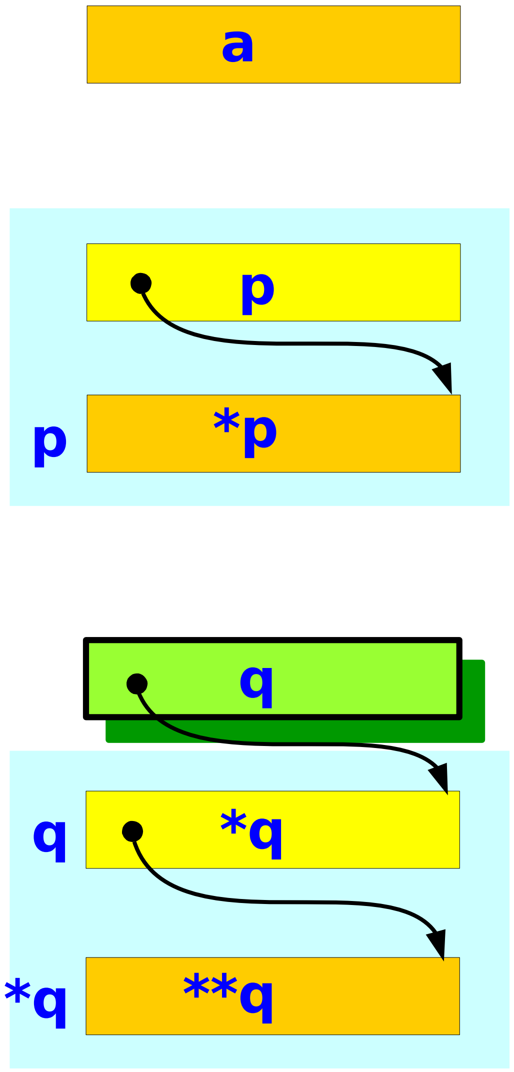
```
int      a  ;
int *    p  ;
int *   *q  ;
```



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

Single and Double Pointer Examples (3)

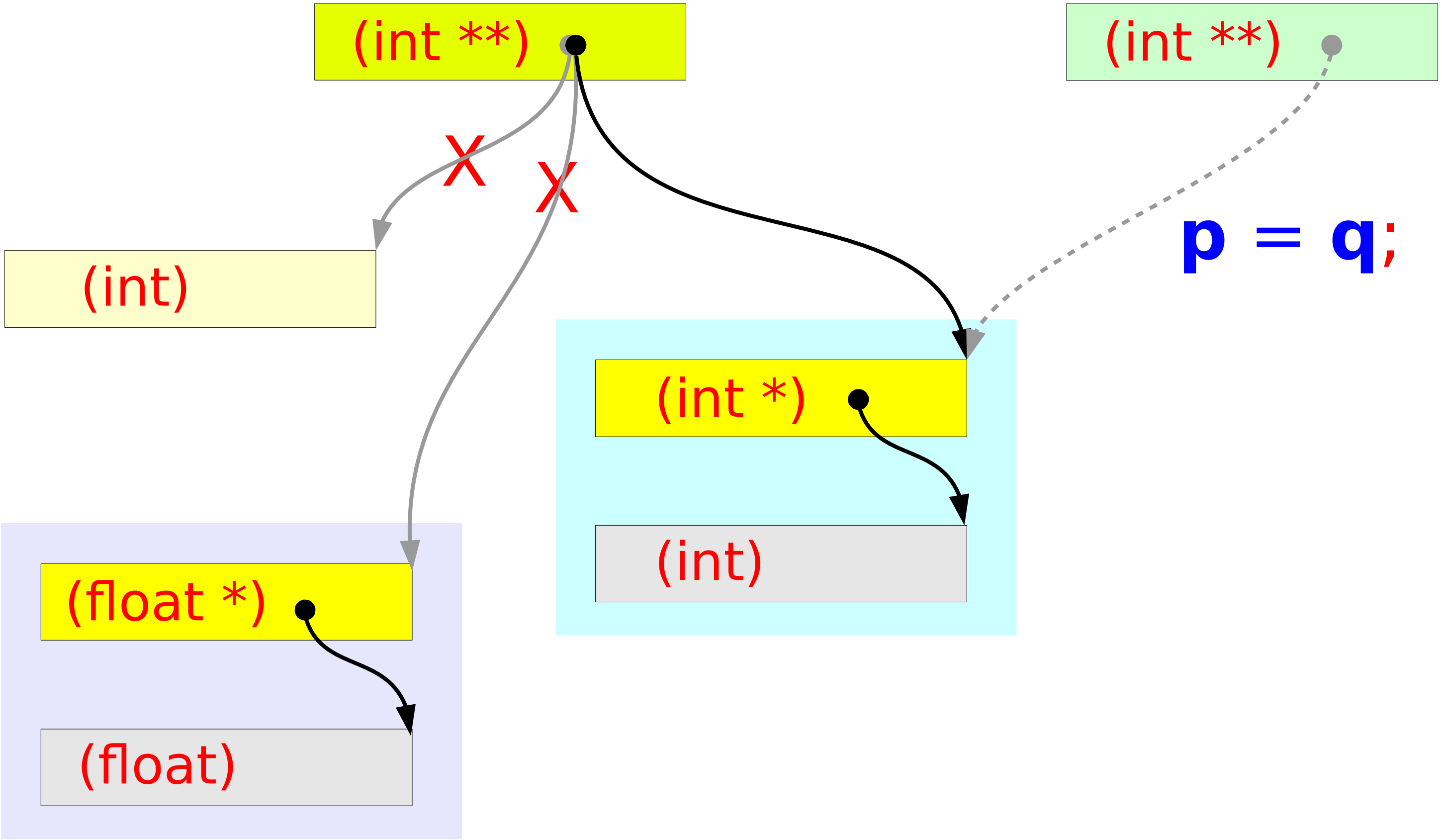
```
int      a  ;
int     *p  ;
int **  q   ;
```



q :
double int pointer variables

Values of double pointer variables

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



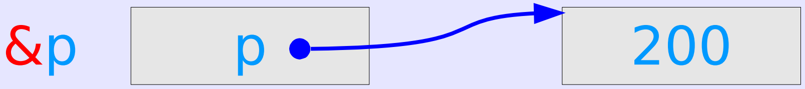
Variable Declarations

```
int a ;
```

&a a =100

The variable a holds an integer data

```
int * p ;
```



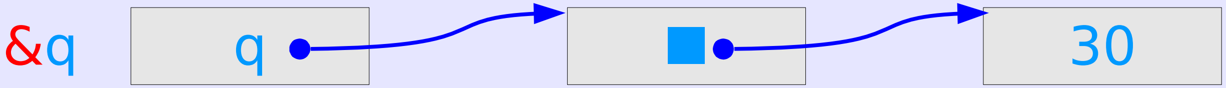
&p

p

200

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

```
int * * q ;
```



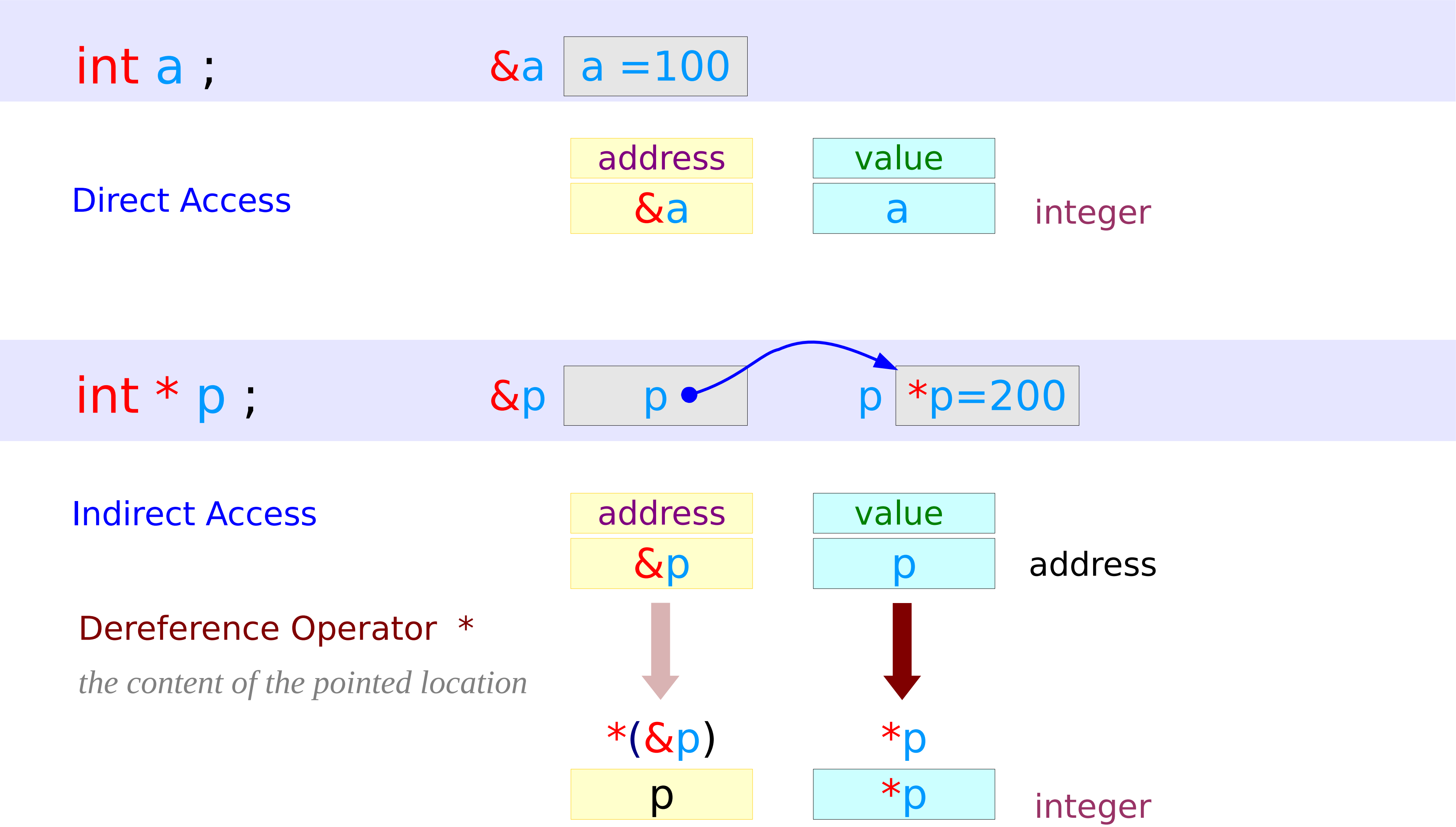
&q

q

30

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

address	value	
&a	a	integer

int * p ; &p p p *p=200

Indirect Access

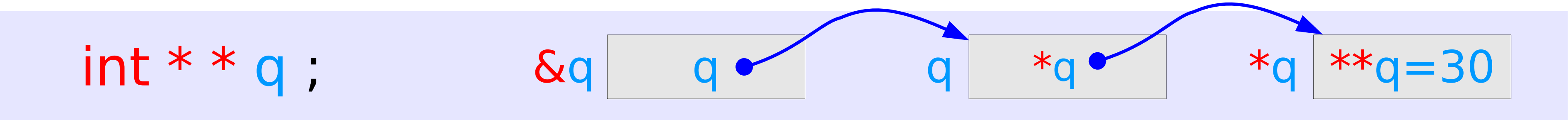
address	value	
&p	p	address

Dereference Operator *

the content of the pointed location

*(&p)	*p	
p	*p	integer

Access Data Via Pointer Variables (2)



int * * q ;

&q q

q *q

*q **q=30

Double Indirect Access

address	value	
&q	q	address

Dereference Operator *

the content of the pointed location

*(&q)	*q	
q	*q	address

Dereference Operator *

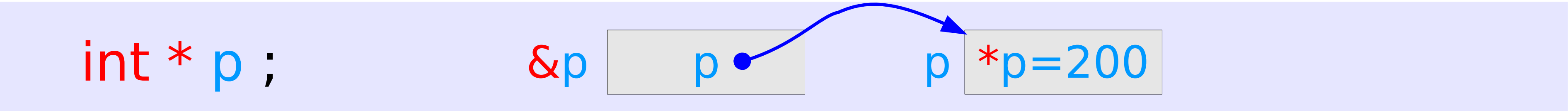
the content of the pointed location

*q	**q	
*q	**q	integer

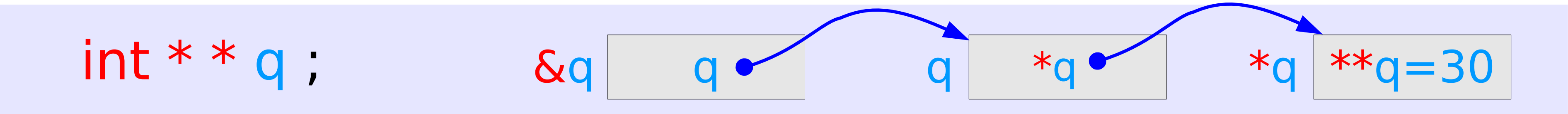
Access Data Via Pointer Variables (3)

int a ;   &a   a =100

	address	value	
Direct Access	&a	a	integer

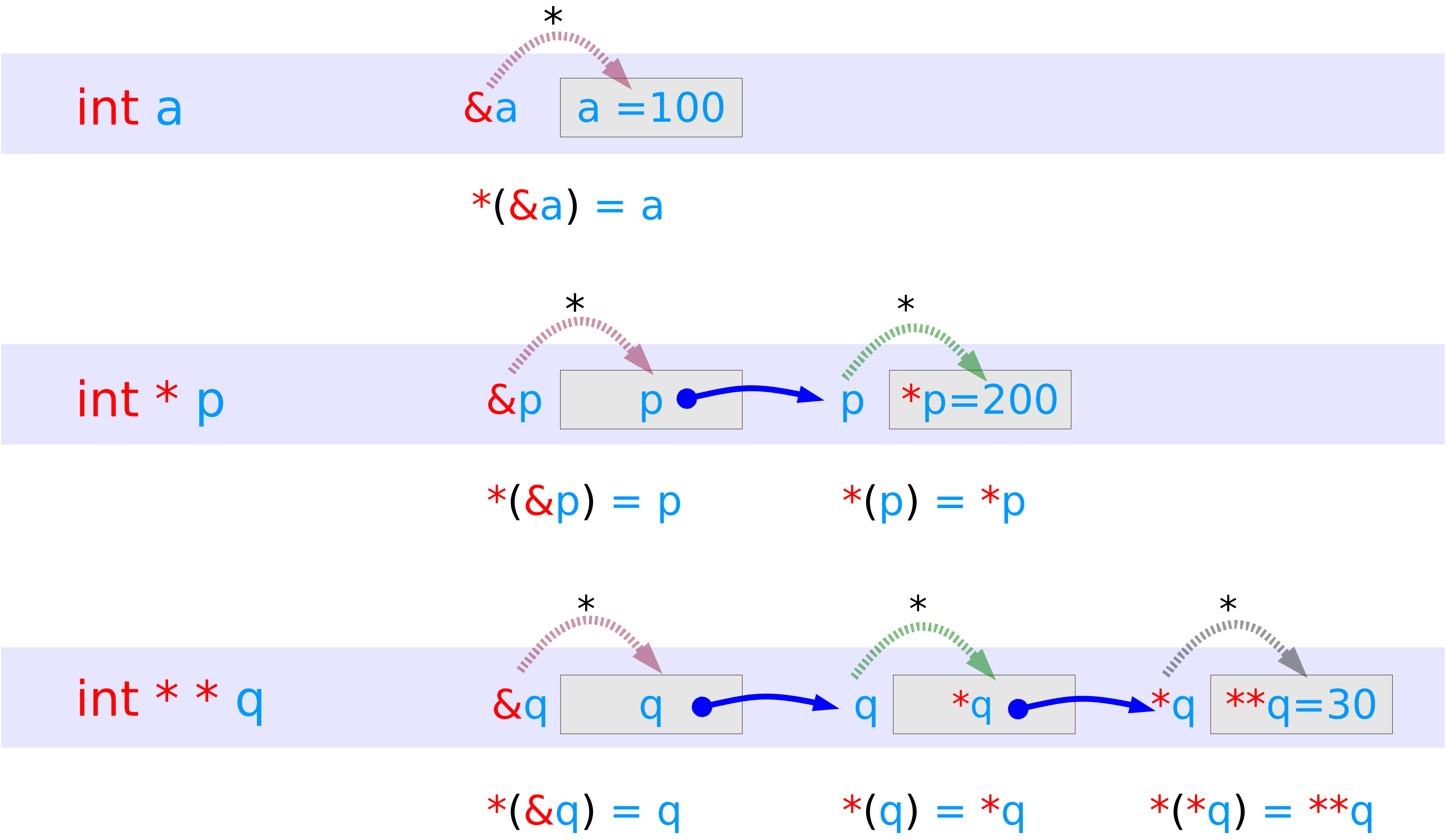


Indirect Access	address	value	
Dereference Operator *	&p	p	address
the content of the pointed location	p	*p	integer



	address	value	
Double Indirect Access	&q	q	address
Dereference Operator *	q	*q	address
the content of the pointed location	*q	**q	integer

Access Data Via Pointer Variables (4)



int a

&a * a =100

*(&a) = a

int * p

&p * p → p * *p=200

*(&p) = p

*(p) = *p

int * * q

&q * q → q * *q → *q * **q=30

*(&q) = q

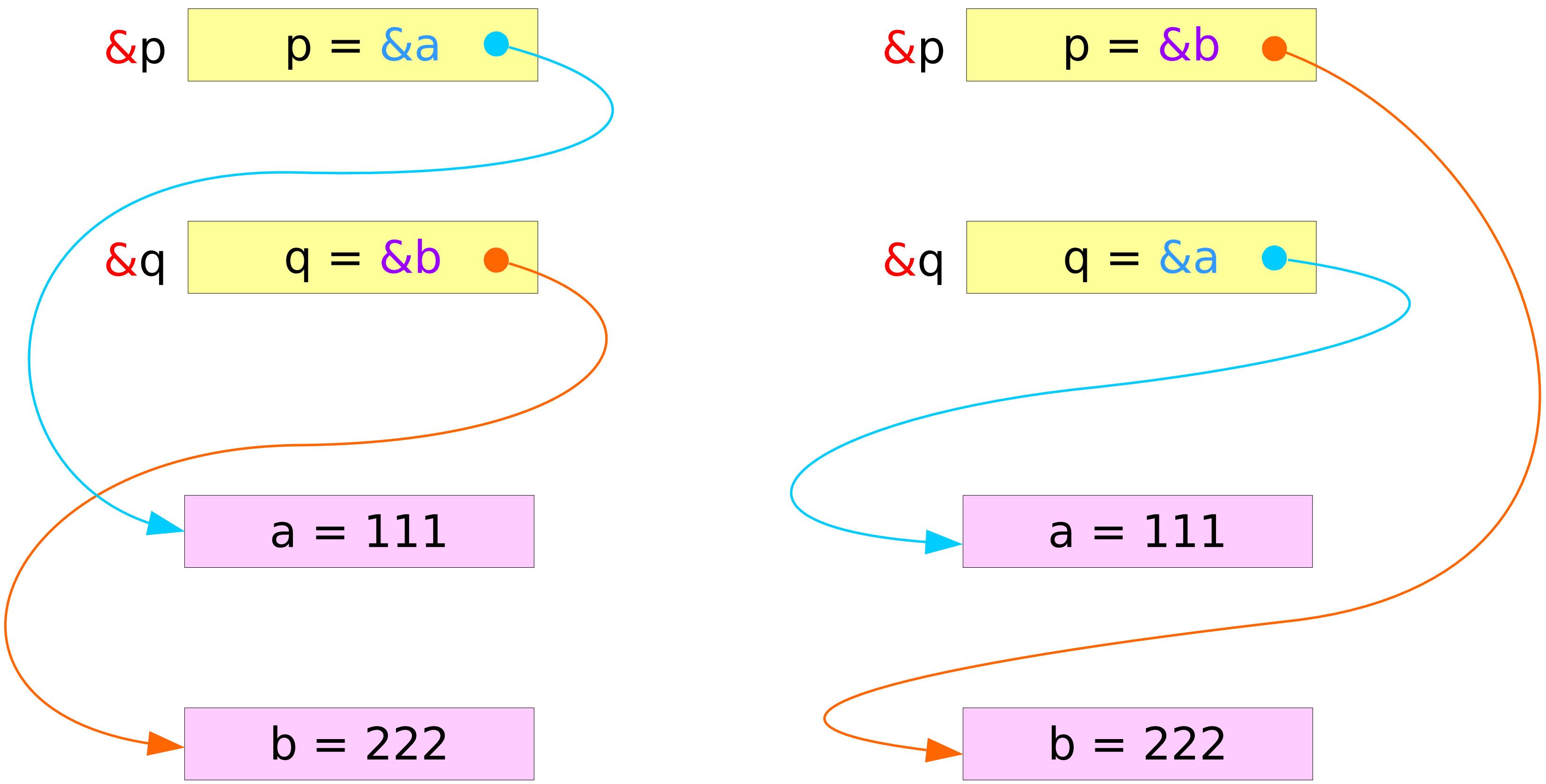
*(q) = *q

*(*q) = **q

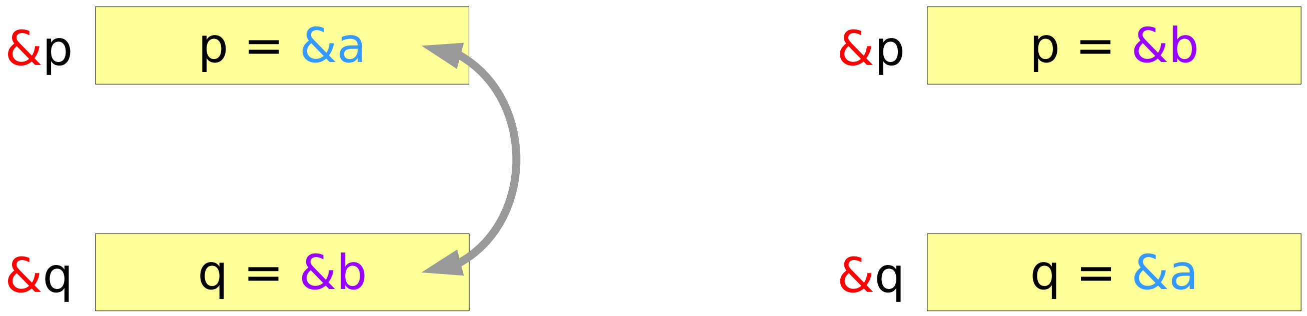
Swapping pointers

- pass by reference
- double pointers

Swapping integer pointers



Swapping integer pointers



&p p = &a

&q q = &b

&p p = &b

&q q = &a

```
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 **	m
int *	*m
int **	n
int *	*n
int *	tmp

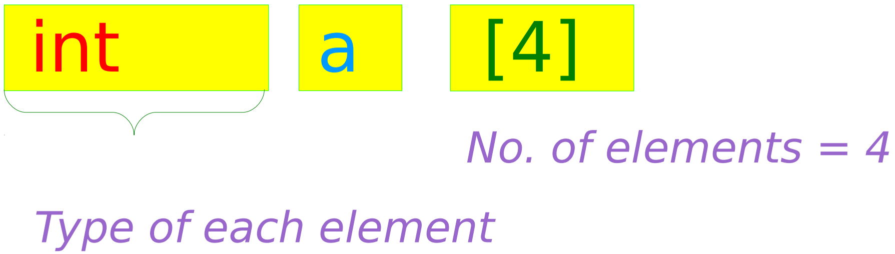
```
int   a,  b;
int *p, *q;          p=&a, q=&b;
    ...
swap_pointers( &p, &q );
```

Array of Pointers

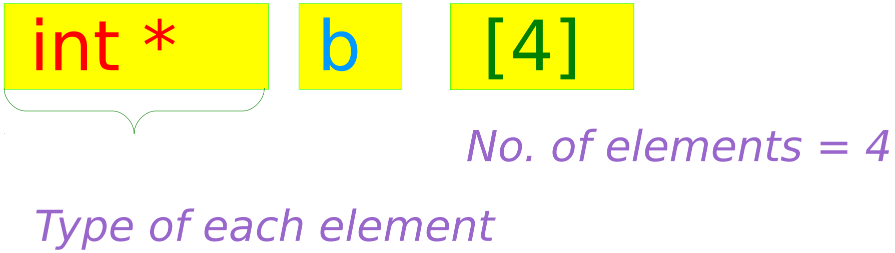
Array of Pointers (1)

```
int        a [4];

int *      b [4];
```

Array name a holds the starting *address*



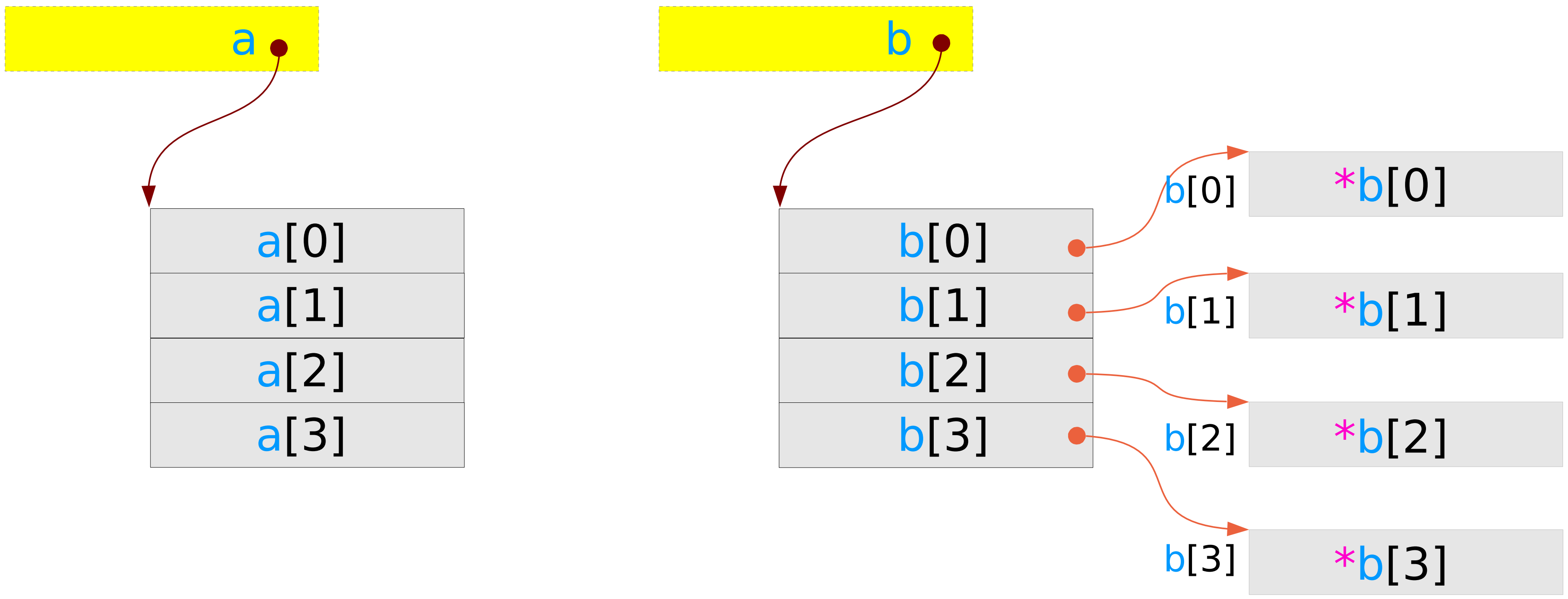
Array name b holds the starting *address*



Array of Pointers (2)

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

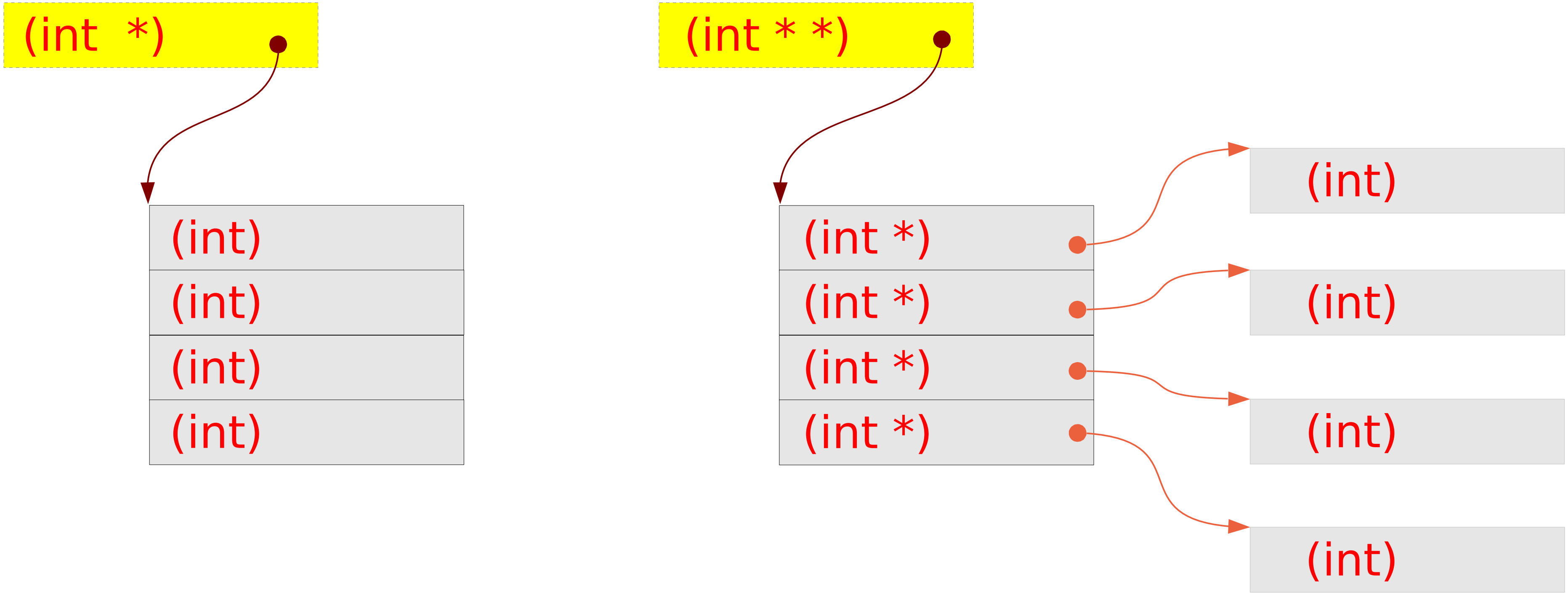
```
int *      b [4];
```



Array of Pointers (3)

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

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



(int *)

(int)
(int)
(int)
(int)

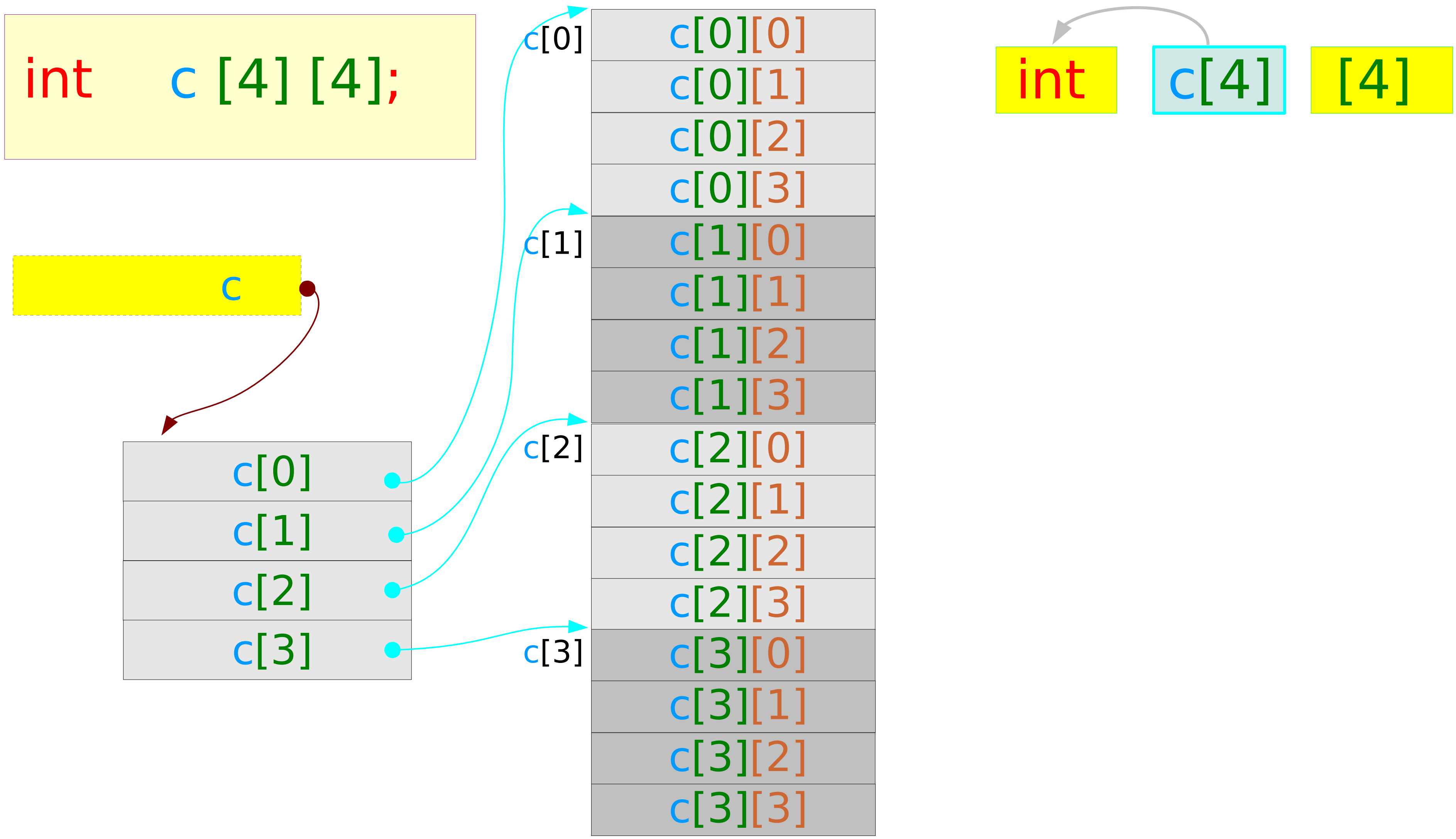
(int * *)

(int *)
(int *)
(int *)
(int *)

(int)
(int)
(int)
(int)

2-d Arrays

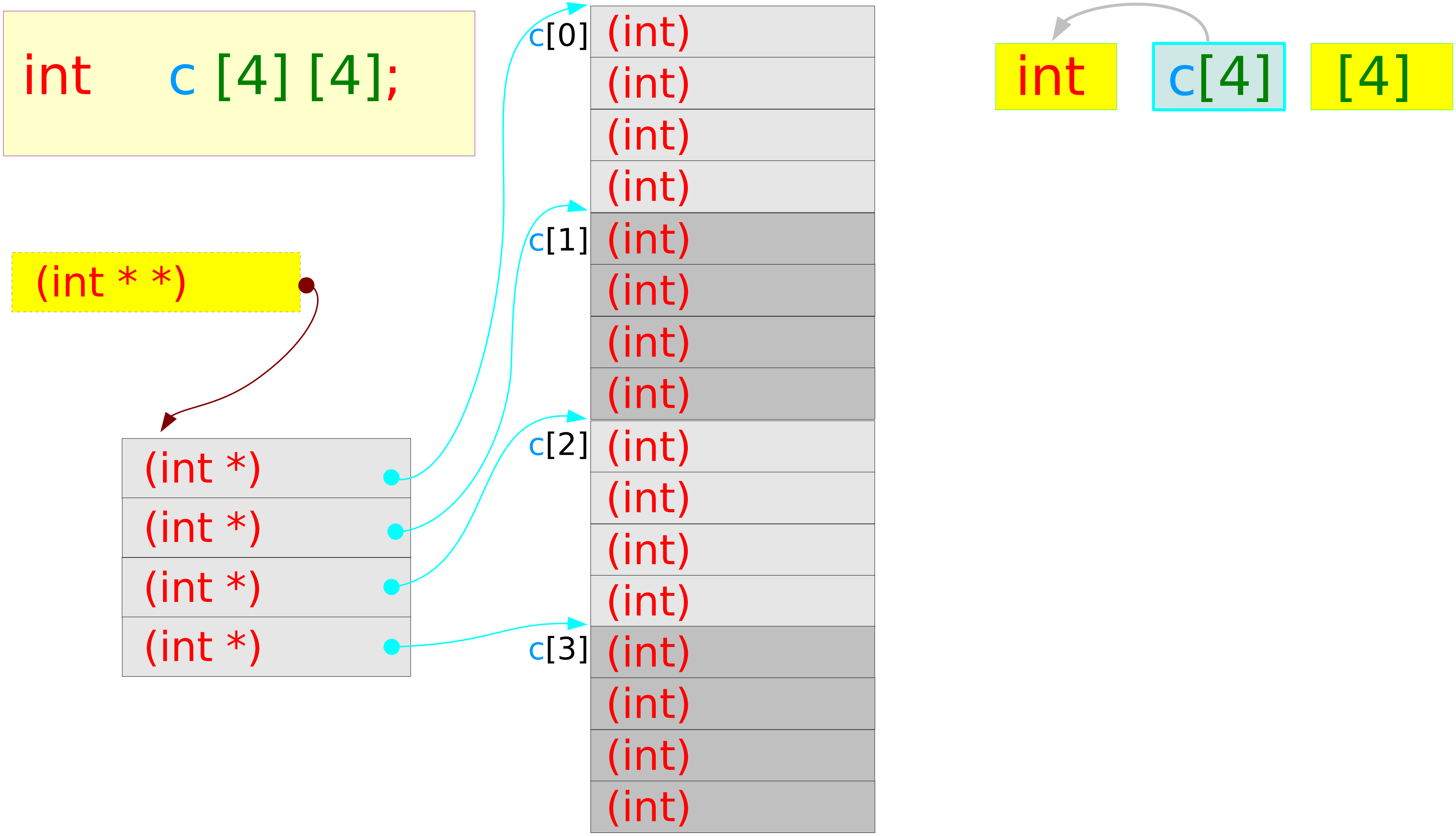
A 2-D Array



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

A 2-D Array

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



A 2-D Array via a double pointer

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

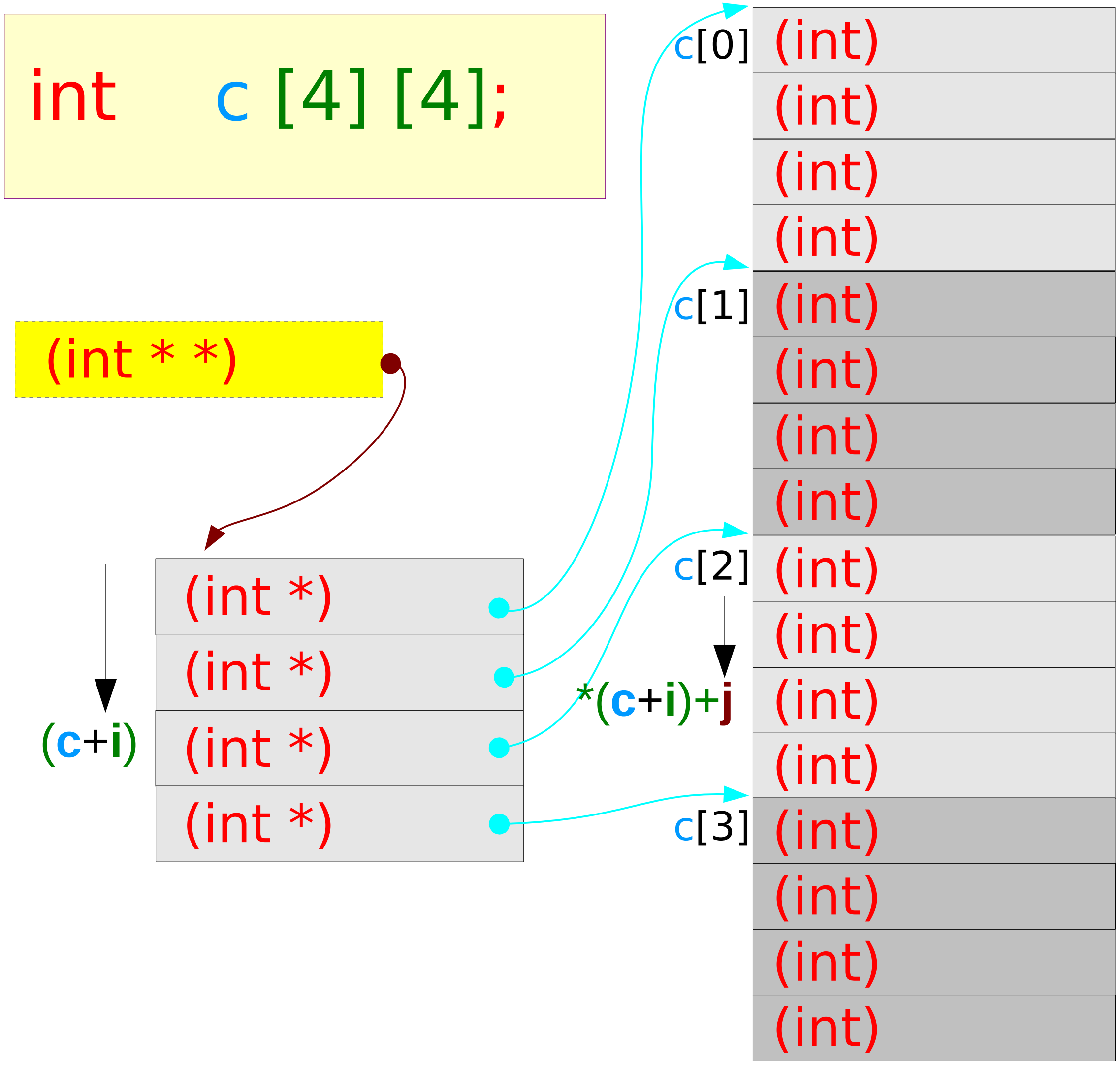
(c [i])[j] ➡ (c [I]) = (*(c+i))

(*(c+i))[j] ➡ (_)[j] = *((_)+j)

((c+i)+j)

A 2-D Array

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

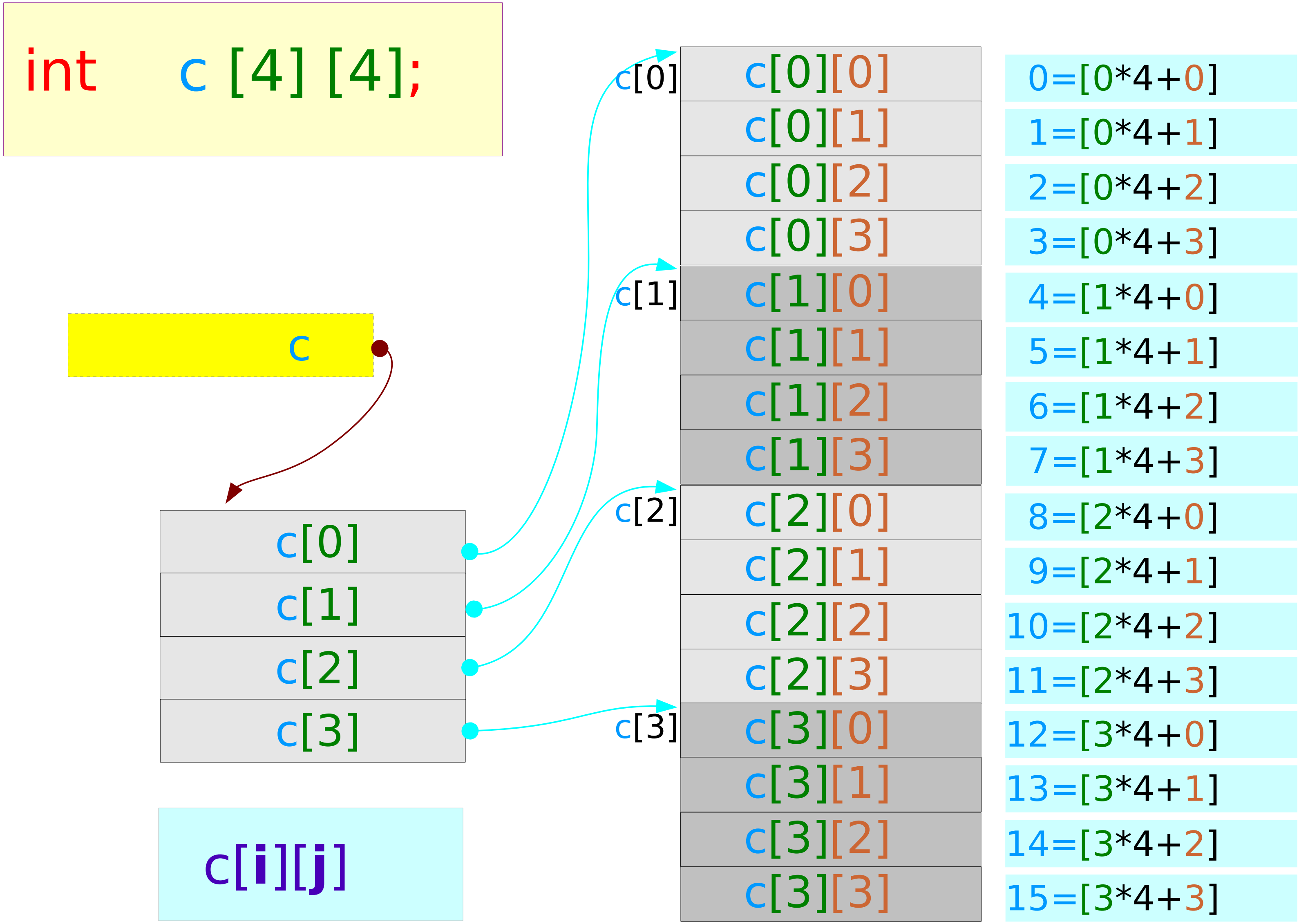


(c [i])[j] ➡

(*(c+i))[j] ➡

((c+i)+j)

A 2-D array via a single pointer

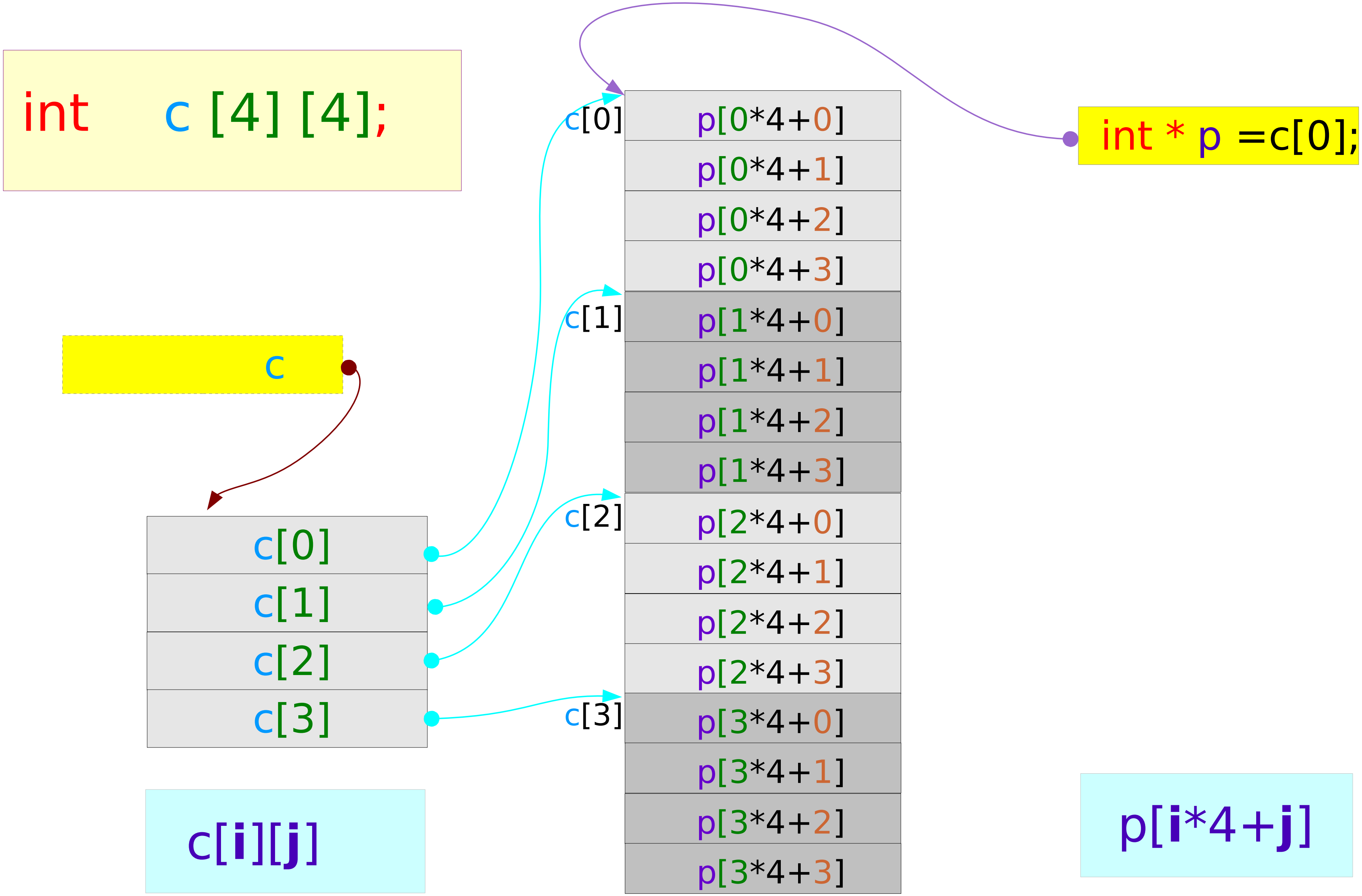


```
int    c [4] [4];
```

c[0][0]	0=[0*4+0]
c[0][1]	1=[0*4+1]
c[0][2]	2=[0*4+2]
c[0][3]	3=[0*4+3]
c[1][0]	4=[1*4+0]
c[1][1]	5=[1*4+1]
c[1][2]	6=[1*4+2]
c[1][3]	7=[1*4+3]
c[2][0]	8=[2*4+0]
c[2][1]	9=[2*4+1]
c[2][2]	10=[2*4+2]
c[2][3]	11=[2*4+3]
c[3][0]	12=[3*4+0]
c[3][1]	13=[3*4+1]
c[3][2]	14=[3*4+2]
c[3][3]	15=[3*4+3]

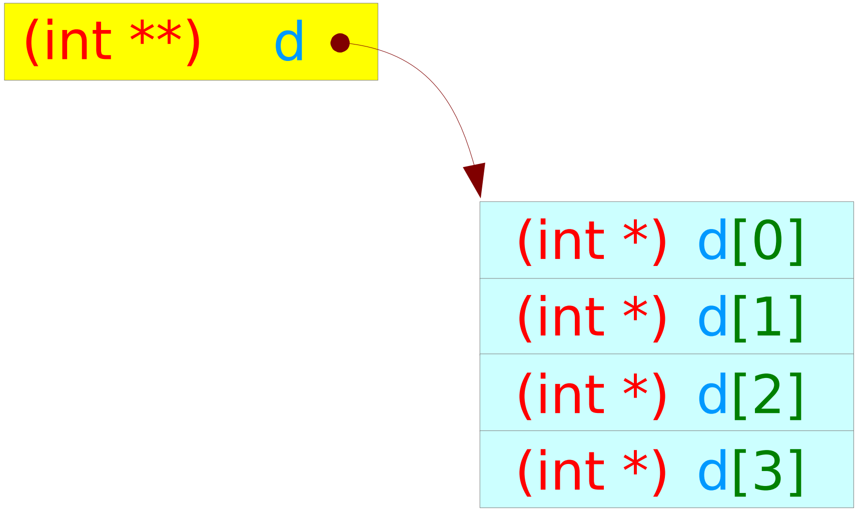
c[**i**][**j**]

A 2-D array via a single pointer



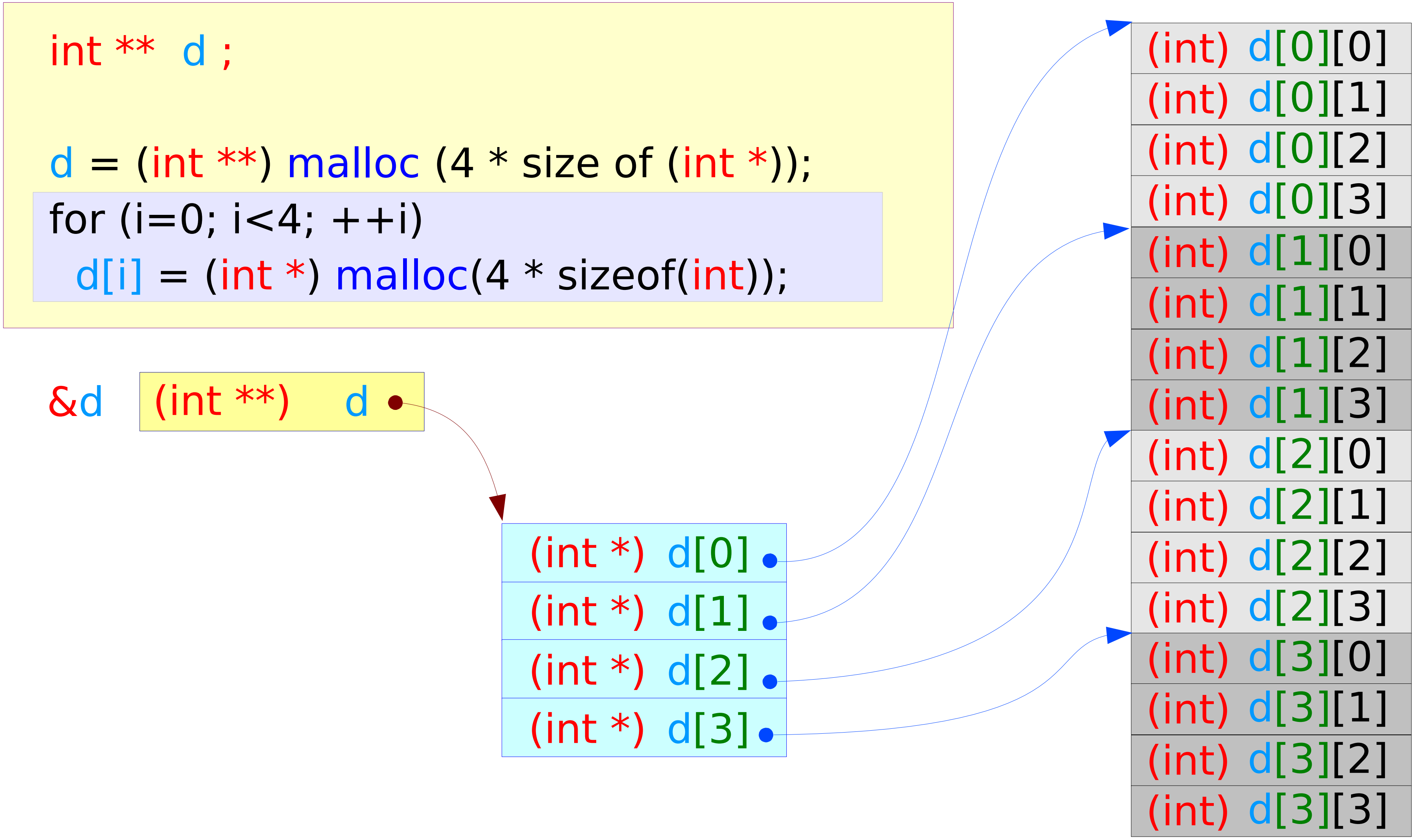
2-D Array Dynamic Memory Allocation (1)

```
int **  d ;

d = (int **) malloc (4 * size of (int *));
for (i=0; i<4; ++i)
  d[i] = (int *) malloc(4 * sizeof(int));
```



2-D Array Dynamic Memory Allocation (2)

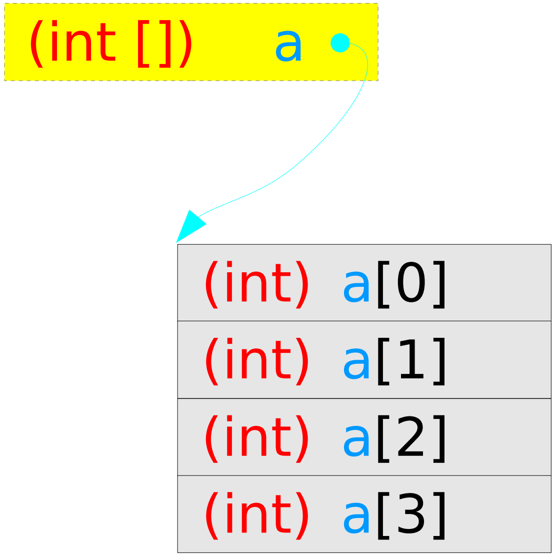
```
int **  d ;

d = (int **) malloc (4 * size of (int *));
for (i=0; i<4; ++i)
  d[i] = (int *) malloc(4 * sizeof(int));
```



Pointer to Arrays

Pointer to array (1)

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



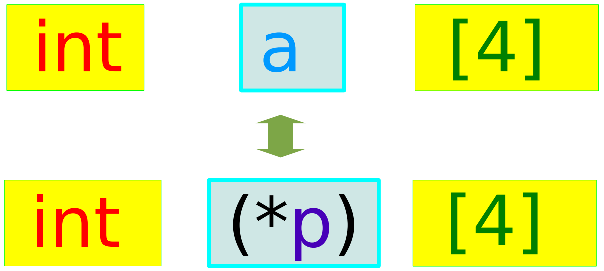
(int []) a

(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]
```

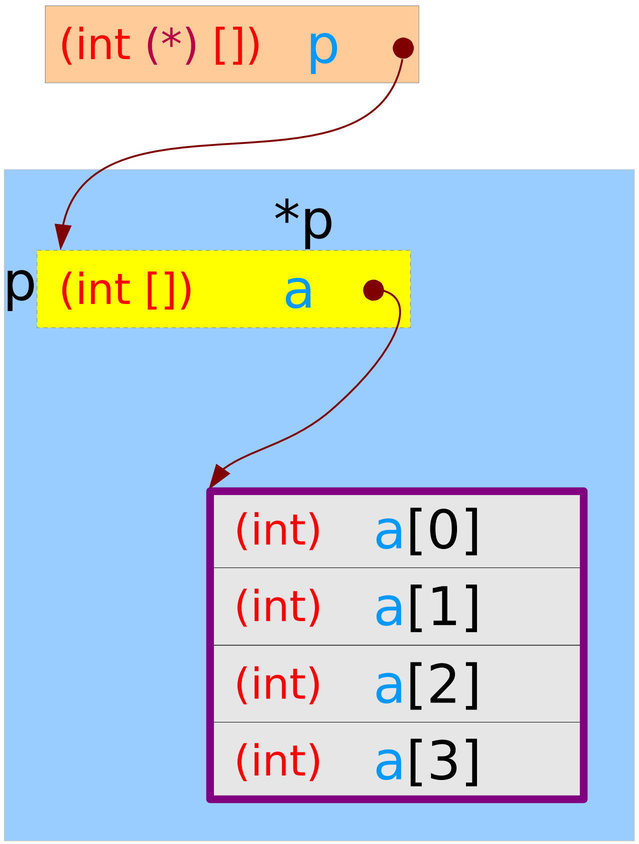
(*p) = a

&(*p) = &a

p = &a

sizeof(p)= 4 bytes

sizeof(*p)= 16 bytes



*p = a

an array with 4 integer elements

Pointer to array (3)

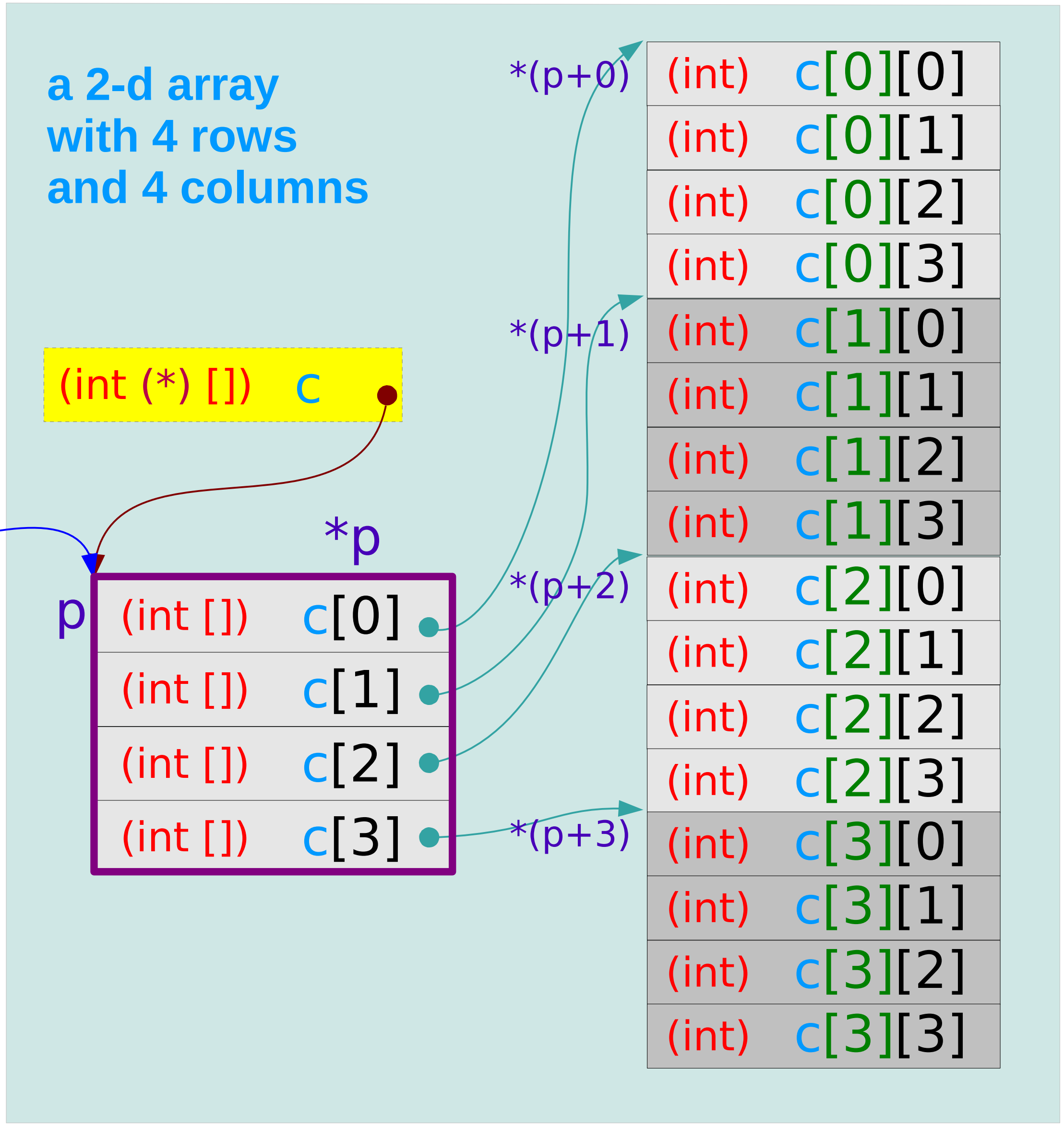
int (*p) [4] ;

int c[4] [4]

&p (int (*) []) p

p = c

(*p) [i][j];



a 2-d array
with 4 rows
and 4 columns

(int (*) []) c

*p

p

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

*(p+0)

*(p+1)

*(p+2)

*(p+3)

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

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