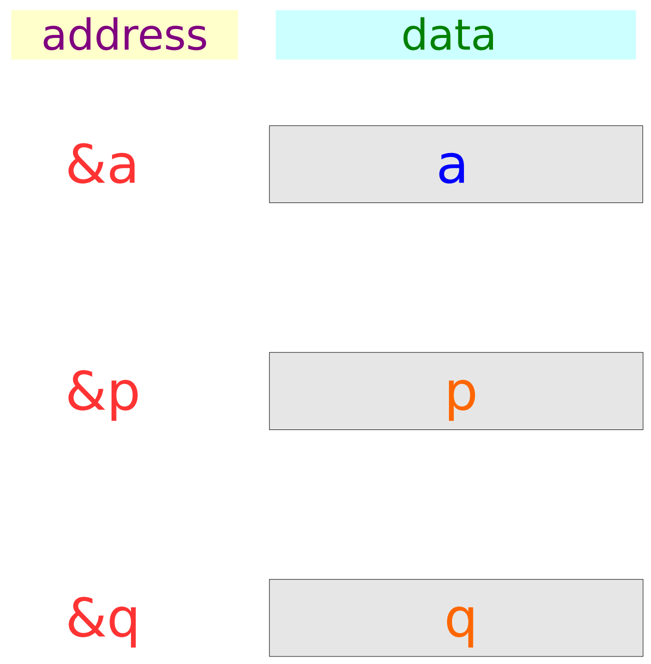
Applications of Pointers (1A)

Young Won Lim
3/21/18

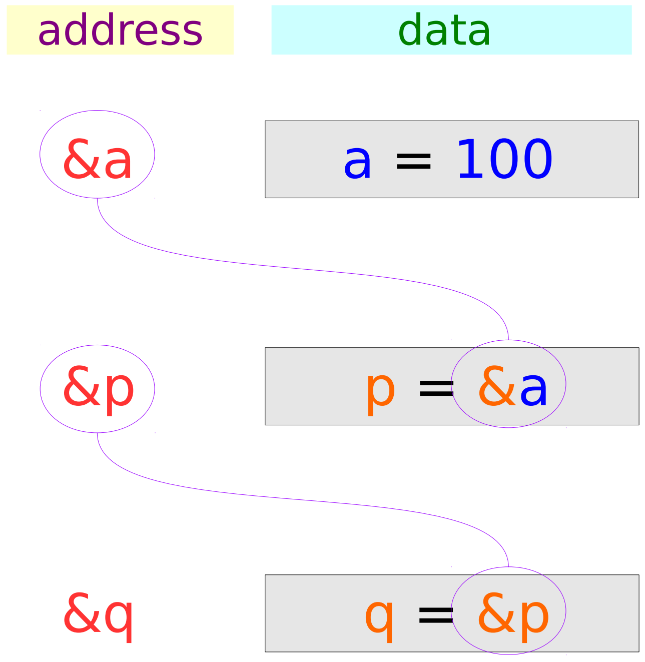
Variables and their addresses

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

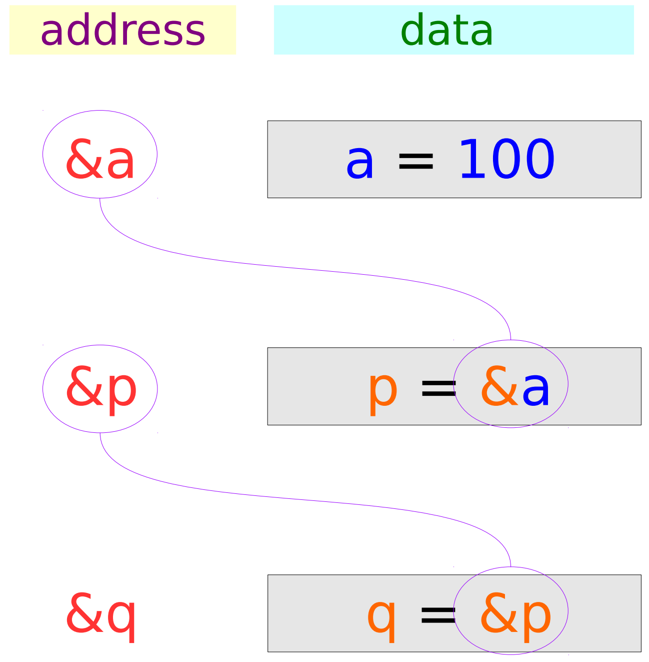


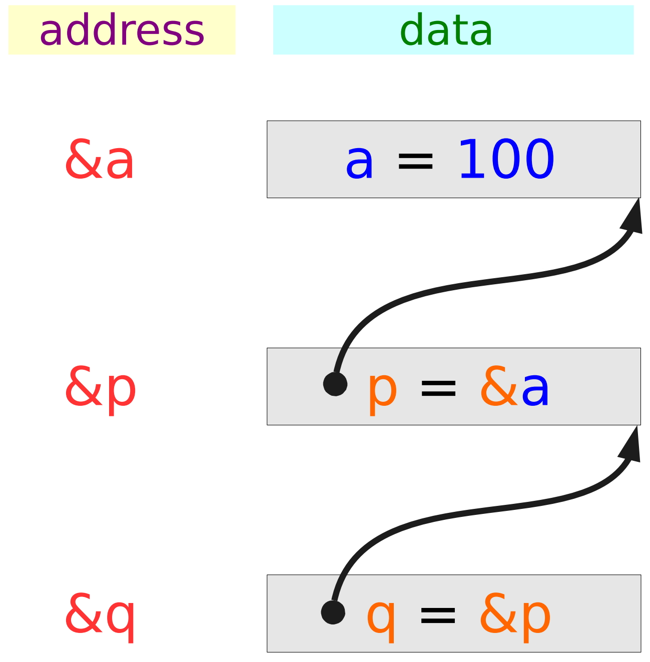
Initialization of Variables

```
int     a = 100;
int   * p = &a;
int  **q = &p;
```



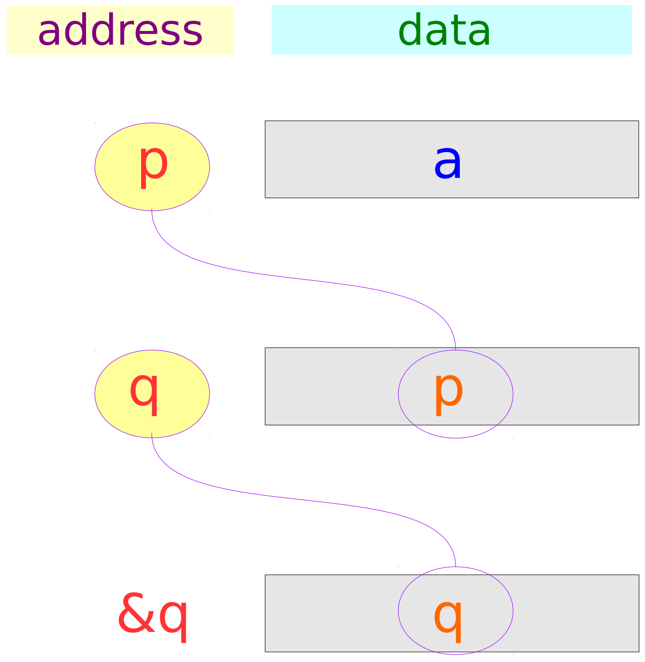
Traditional arrow notations





Pointed addresses : p, q

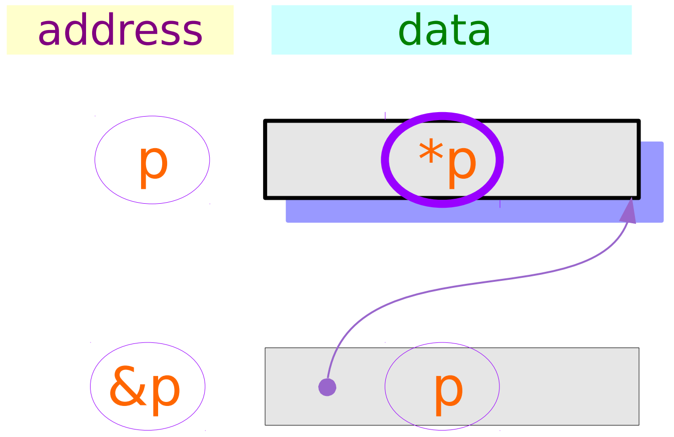
```
int    a;

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

Variable aliasing

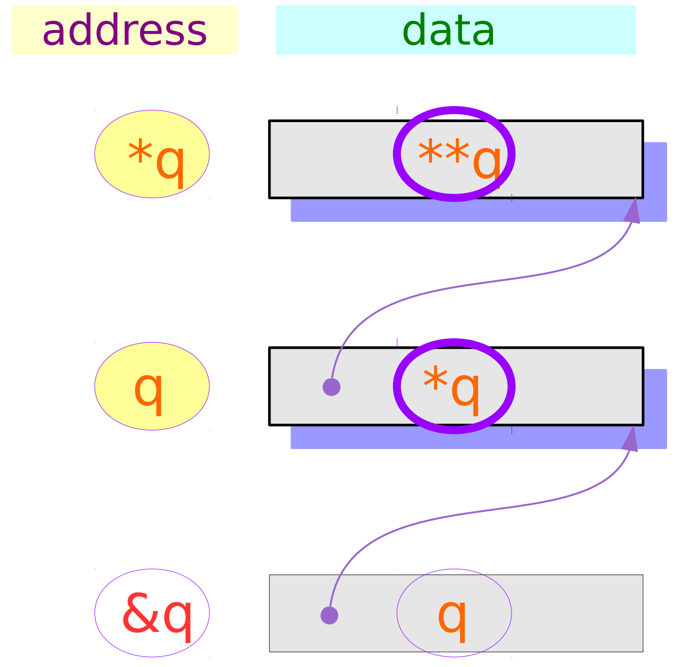
$p = \&a \Rightarrow *p \equiv a$

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

Relations after address assignment

Dereferenced Variables : *q, **q

```
int    a;

int  * p = &a;

int **q = &p;
```



$**q \equiv a$

$*q \equiv p$

Dereferenced Variables : *q, **q

```
int     a;

int  * p = &a;

int **q = &p;
```

Address assignment

Variable aliasing

p = &a ➡ *p ≡ a

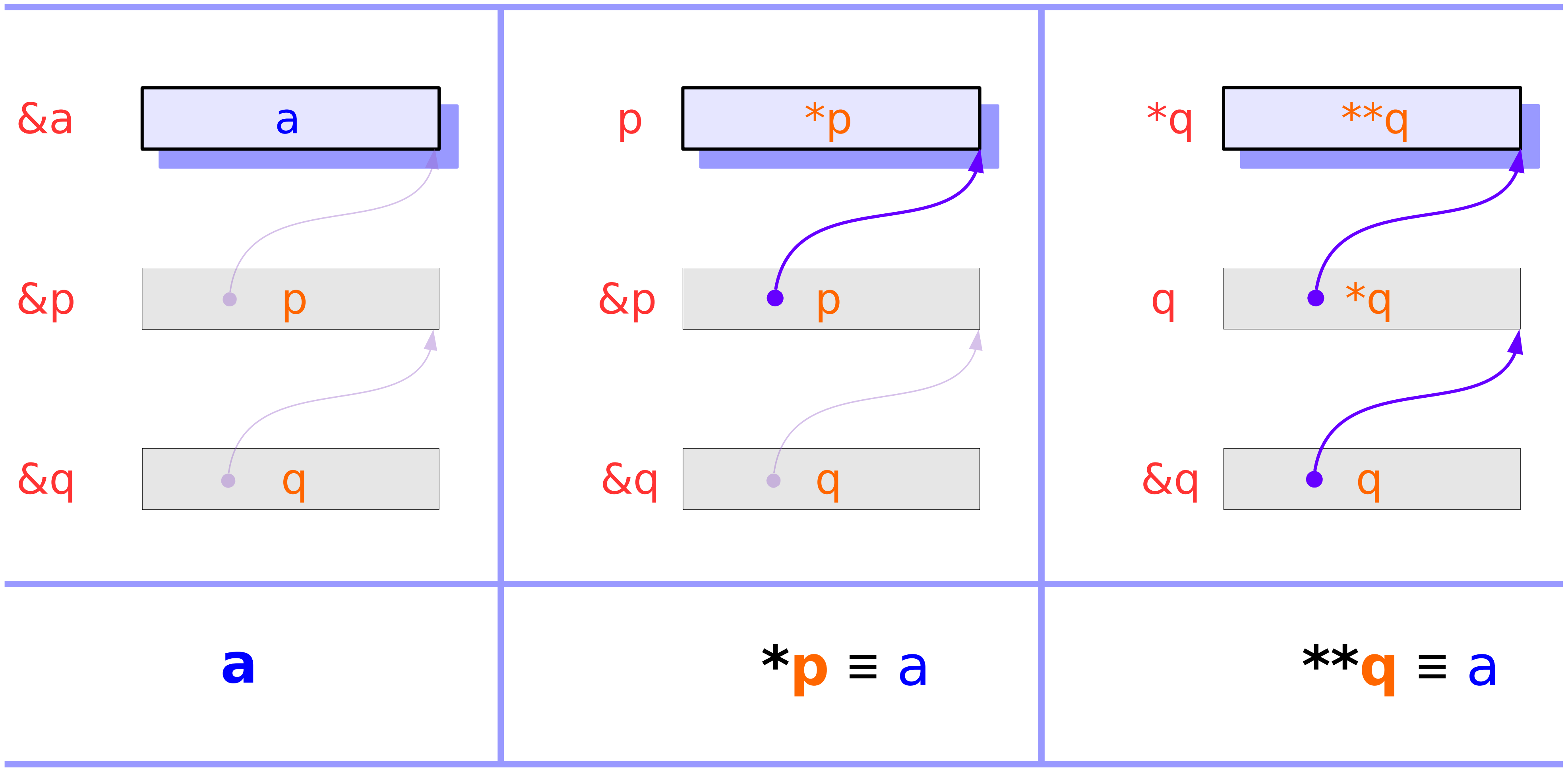
q = &p ➡ *q ≡ p

➡ **q ≡ a

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

Relations after address assignment

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

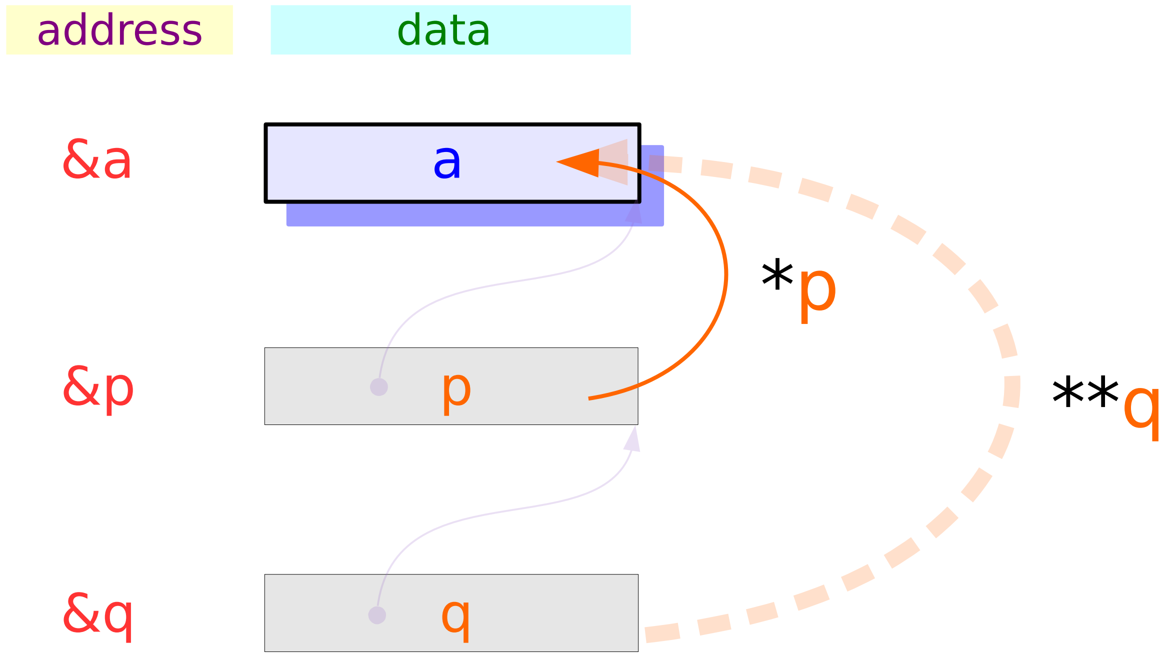


a

*p ≡ a

**q ≡ a

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

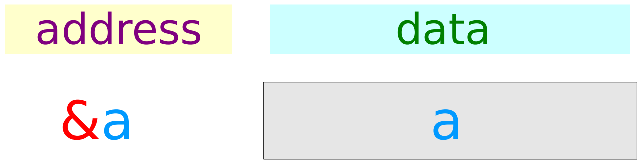


1) Read / Write a
2) Read / Write *p
3) Read / Write **q

Variables

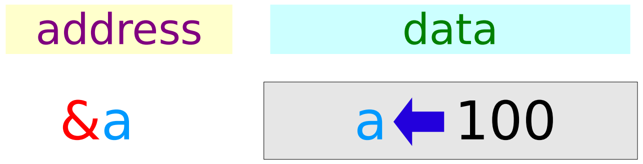
```
int     a;
```

a can hold an *integer*



```
a = 100;
```

a holds 100



Pointer Variables

```
int *      p;
```

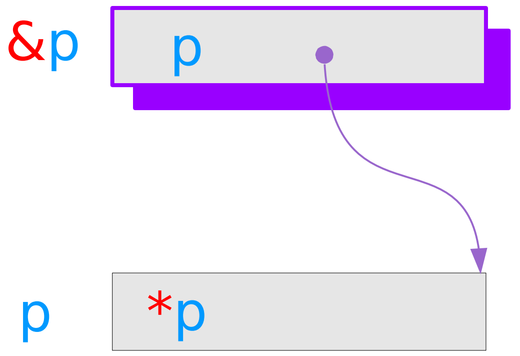
p can hold an ***address***

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

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

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

p** holds a int type ***data



Pointer to Pointer Variable

```
int **     q;
```

q holds an *address*

$\underbrace{\text{int **}}_{\textit{pointer to pointer to int}}$ q;

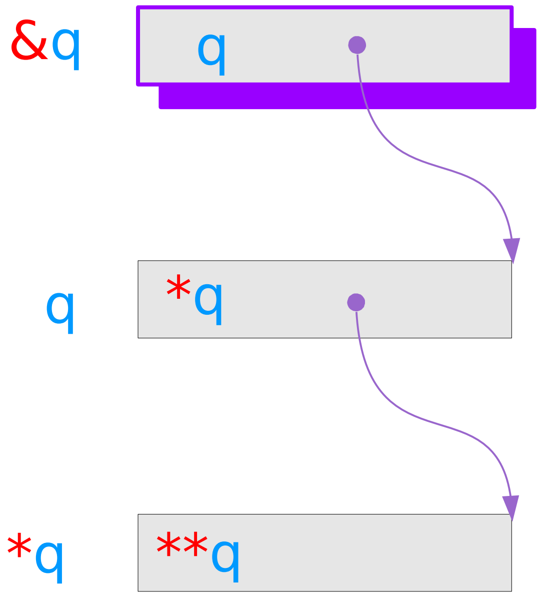
q holds an ***address*** of
a pointer to int type data

$\underbrace{\text{int *}}_{\textit{pointer to int}}$ *q;

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

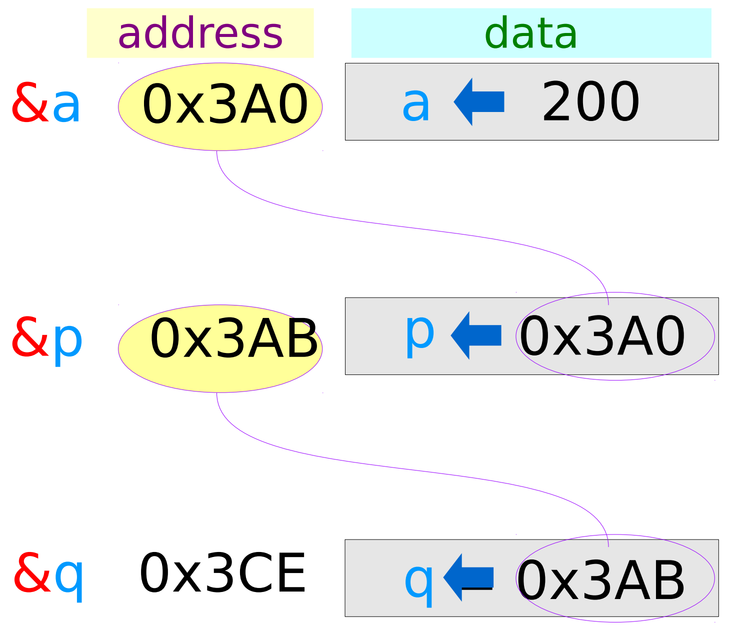
$\underbrace{\text{int}}_{\textit{int}}$ **q;

****q** holds a int type data



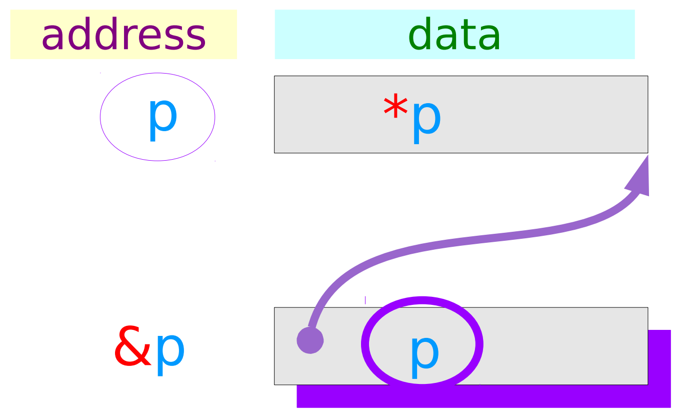
Pointer Variables Examples

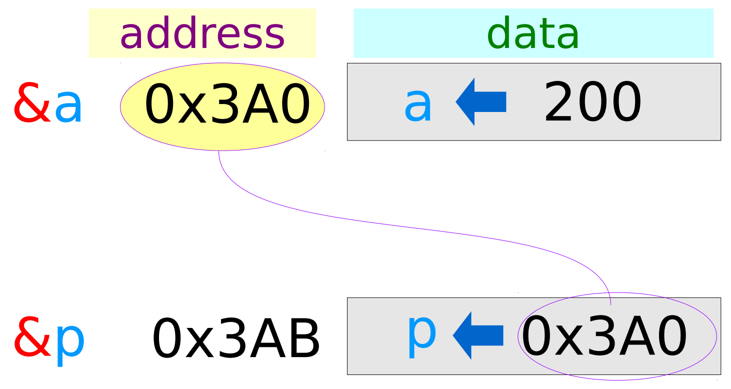
```
int      a = 200;

int *    p = & a;

int **   q = & p;
```



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

Pointer Variable **p** with an arrow notation





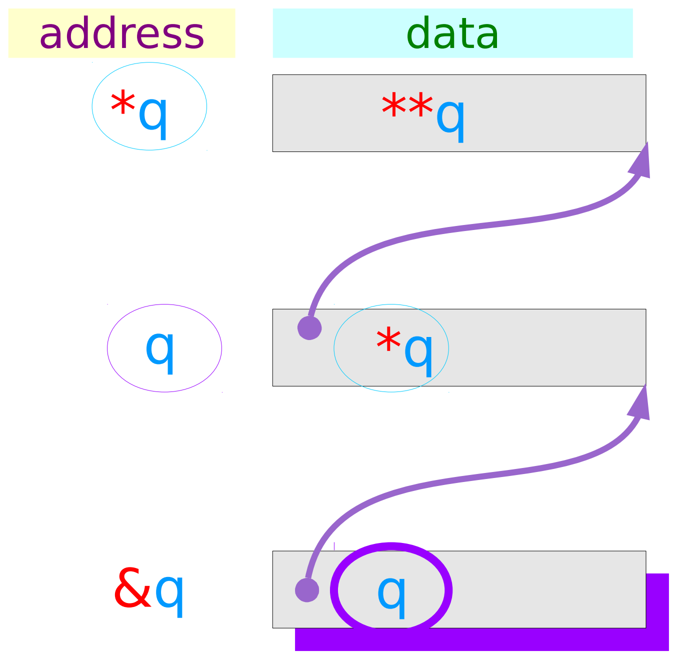
using an arrow notation

$\&p \Rightarrow$ 0x3AB

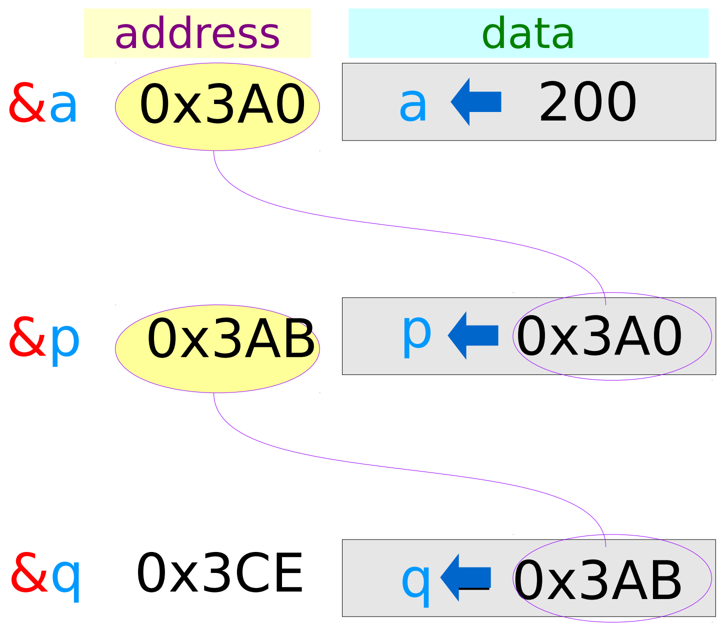
$p \Rightarrow$ 0x3A0

$*p \Rightarrow$ 200

Pointer Variable **q** with an arrow notation

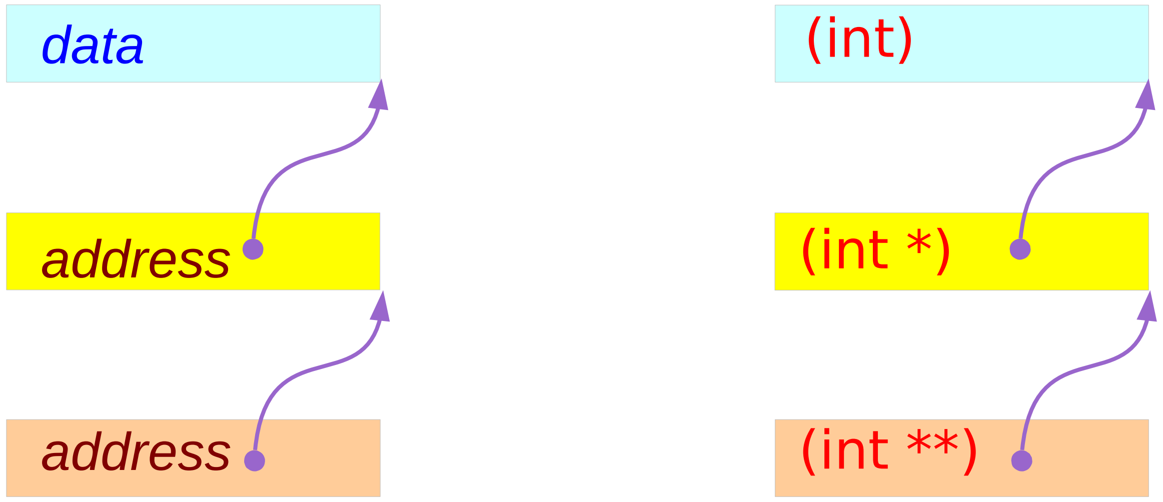


using an arrow notation



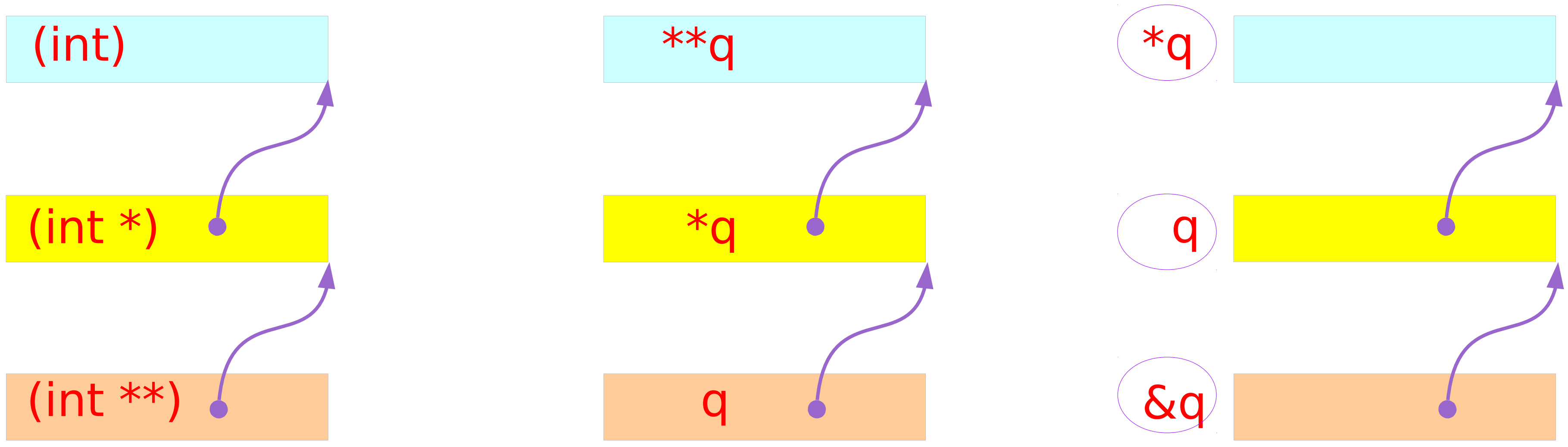
&q ➡	0x3CE
q ➡	0x3AB
*q ➡	0x3A0
**q ➡	200

The type view point of pointers



Types

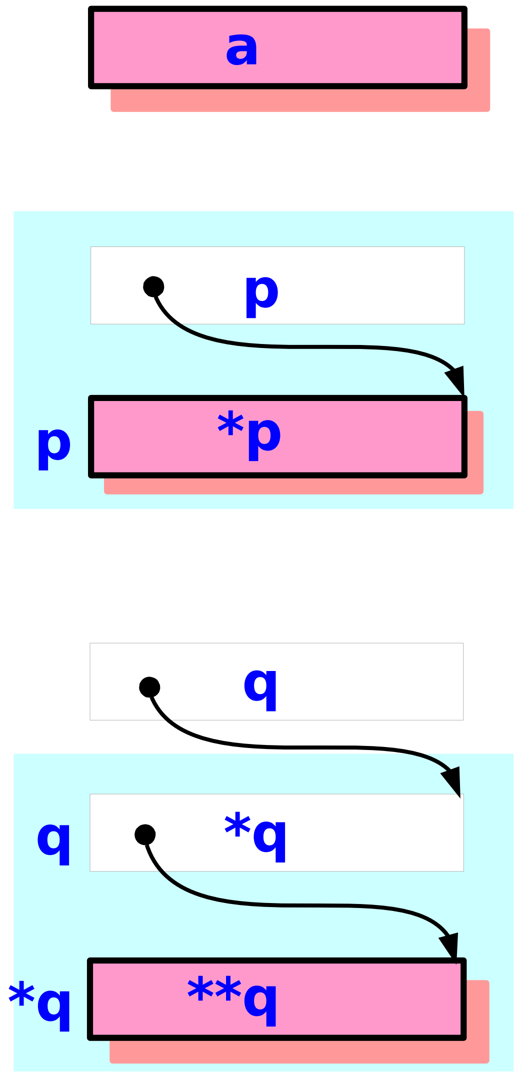
The different view points of pointers



Types **Variables** **Addresses**

Single and Double Pointer Examples (1)

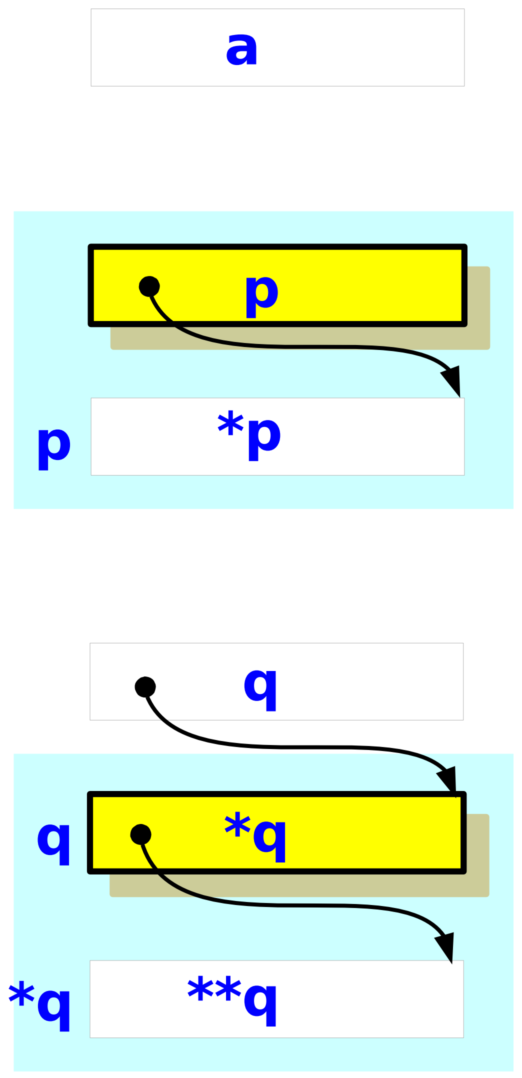
```
int    a ;
int   *p ;
int  **q ;
```



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

Single and Double Pointer Examples (2)

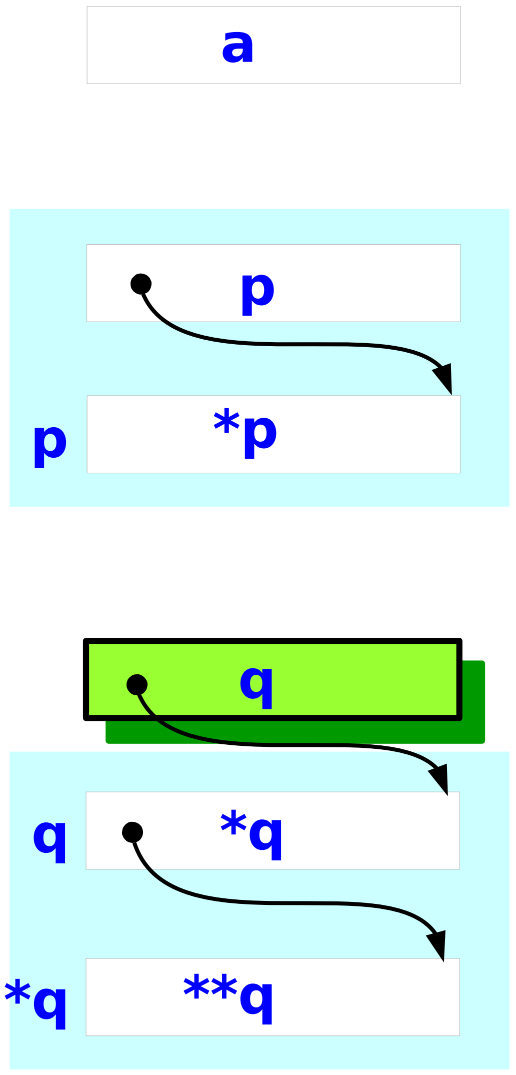
```
int    a  ;
int *  p  ;
int * *q  ;
```



p and ***q :**
int pointer variables
(singlepointers)

Single and Double Pointer Examples (3)

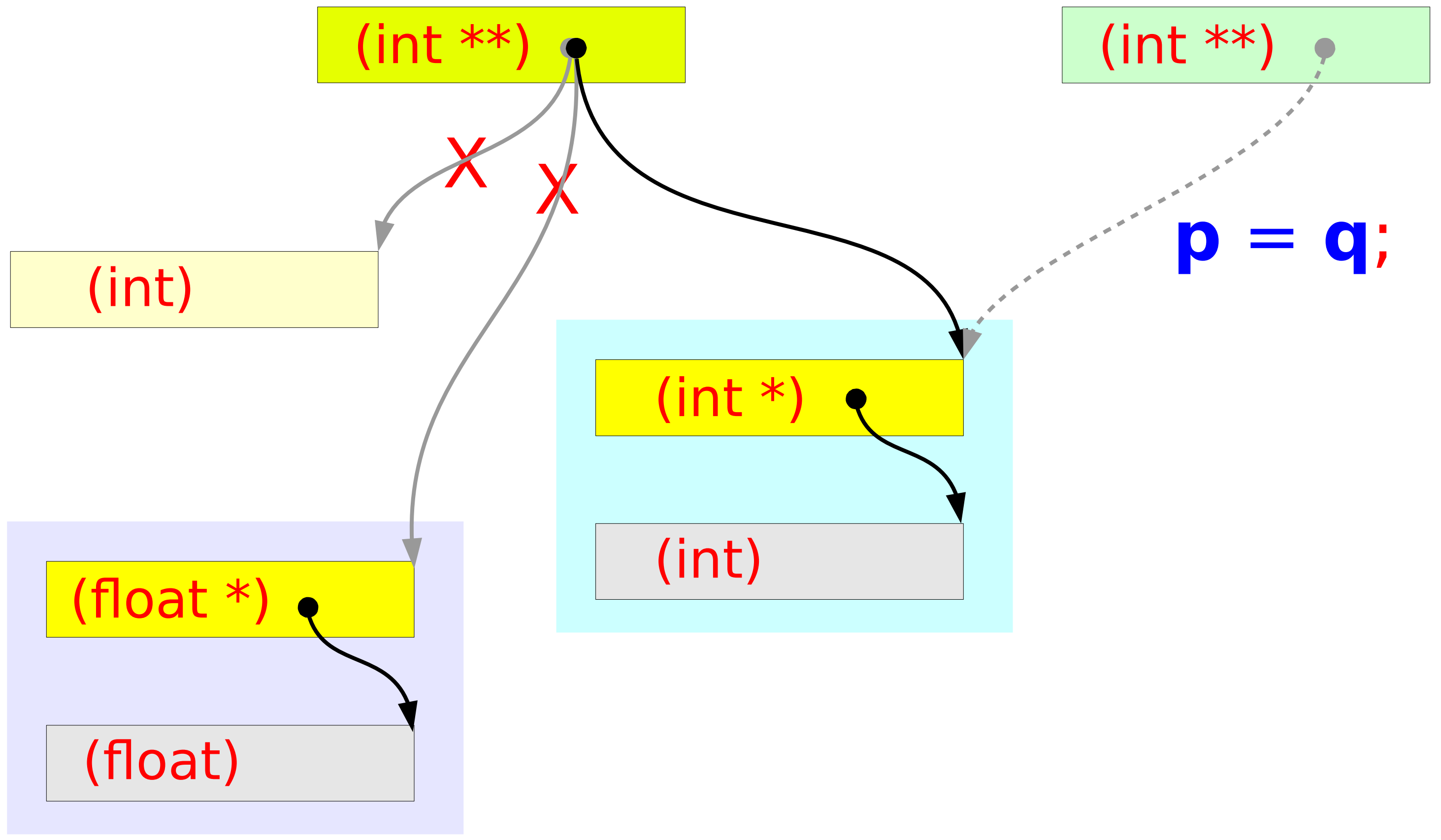
```
int      a ;
int *    p ;
int **   q ;
```



q :
double int pointer variables

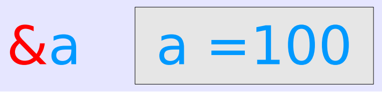
Values of double pointer variables

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



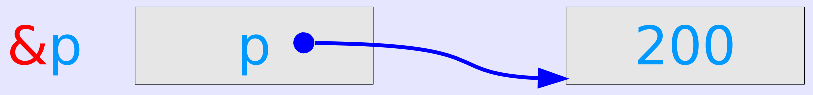
Variable Declarations

int a ;



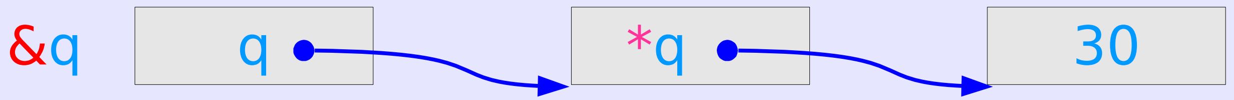
The variable a holds an integer data

int * p ;



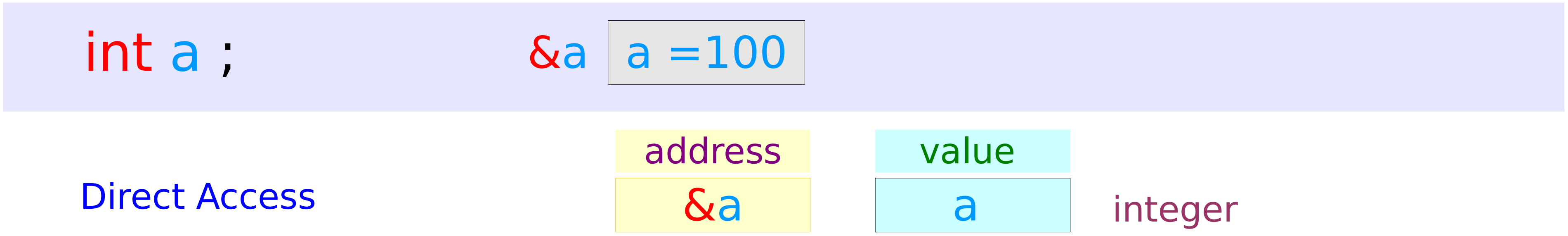
The **pointer** variable p holds an address,
at this address, an integer data is stored

int * * q ;

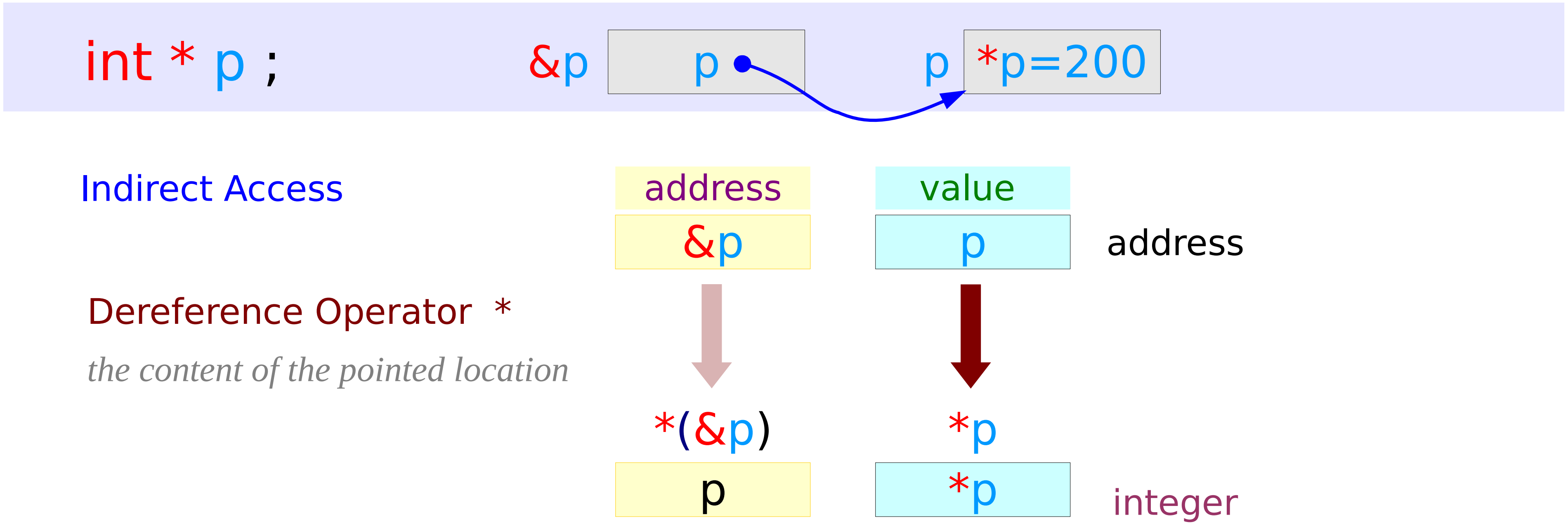


The **pointer** variable q holds an address,
at the address q, another address *q is stored,
at the address *q, an integer data **q is stored

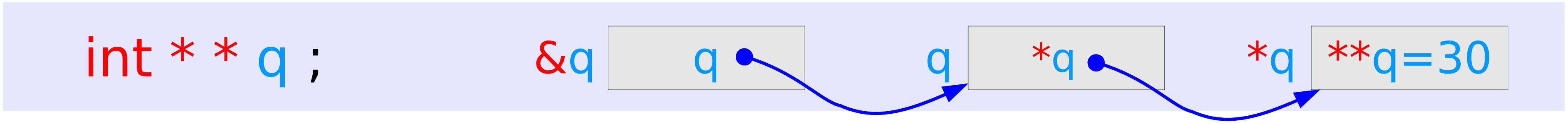
Access Data Via Pointer Variables (1)



Access Data Via Pointer Variables (2)



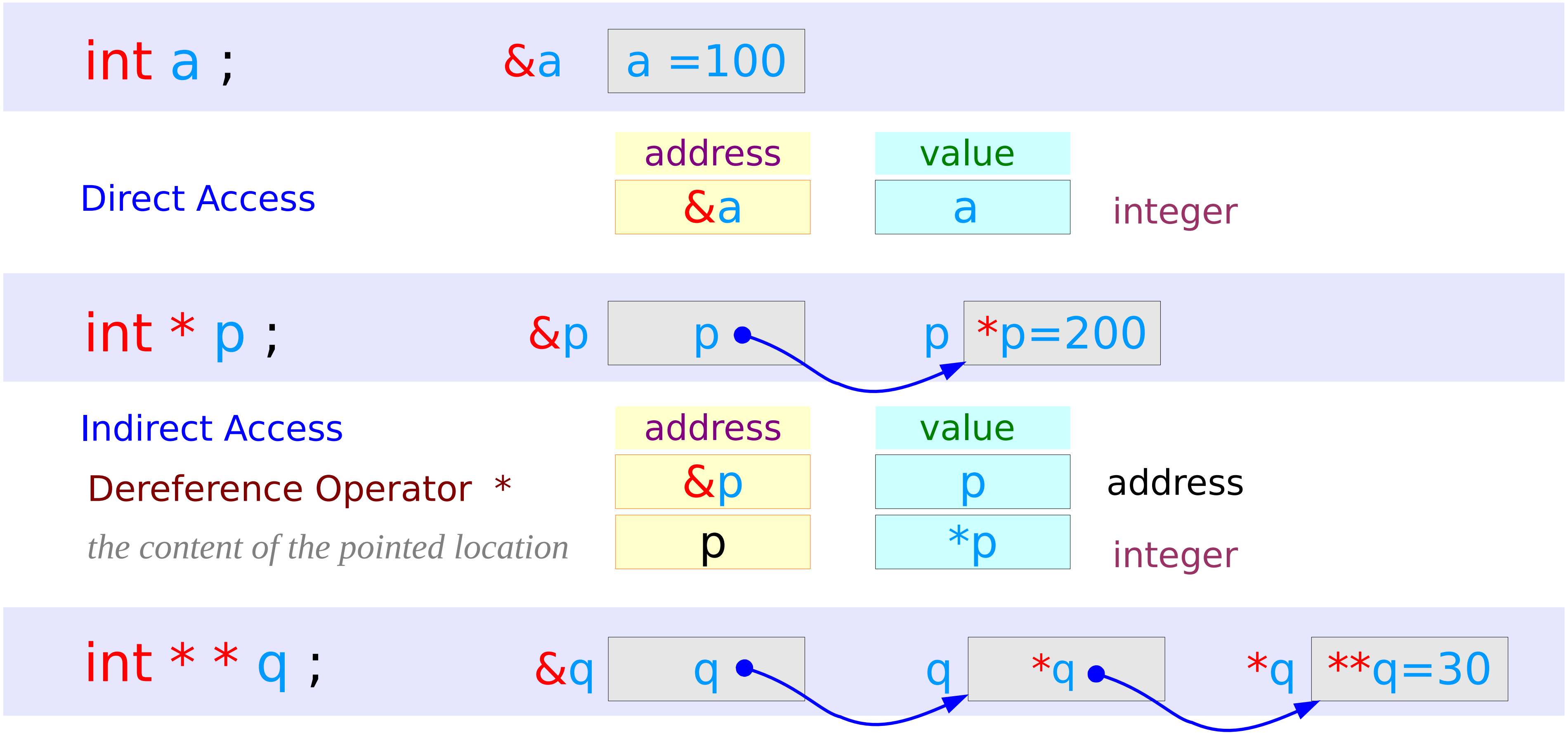
Access Data Via Pointer Variables (3)



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 (4)



Direct Access

address	value	
&a	a	integer

Indirect Access

Dereference Operator *

the content of the pointed location

address	value	
&p	p	address
p	*p	integer

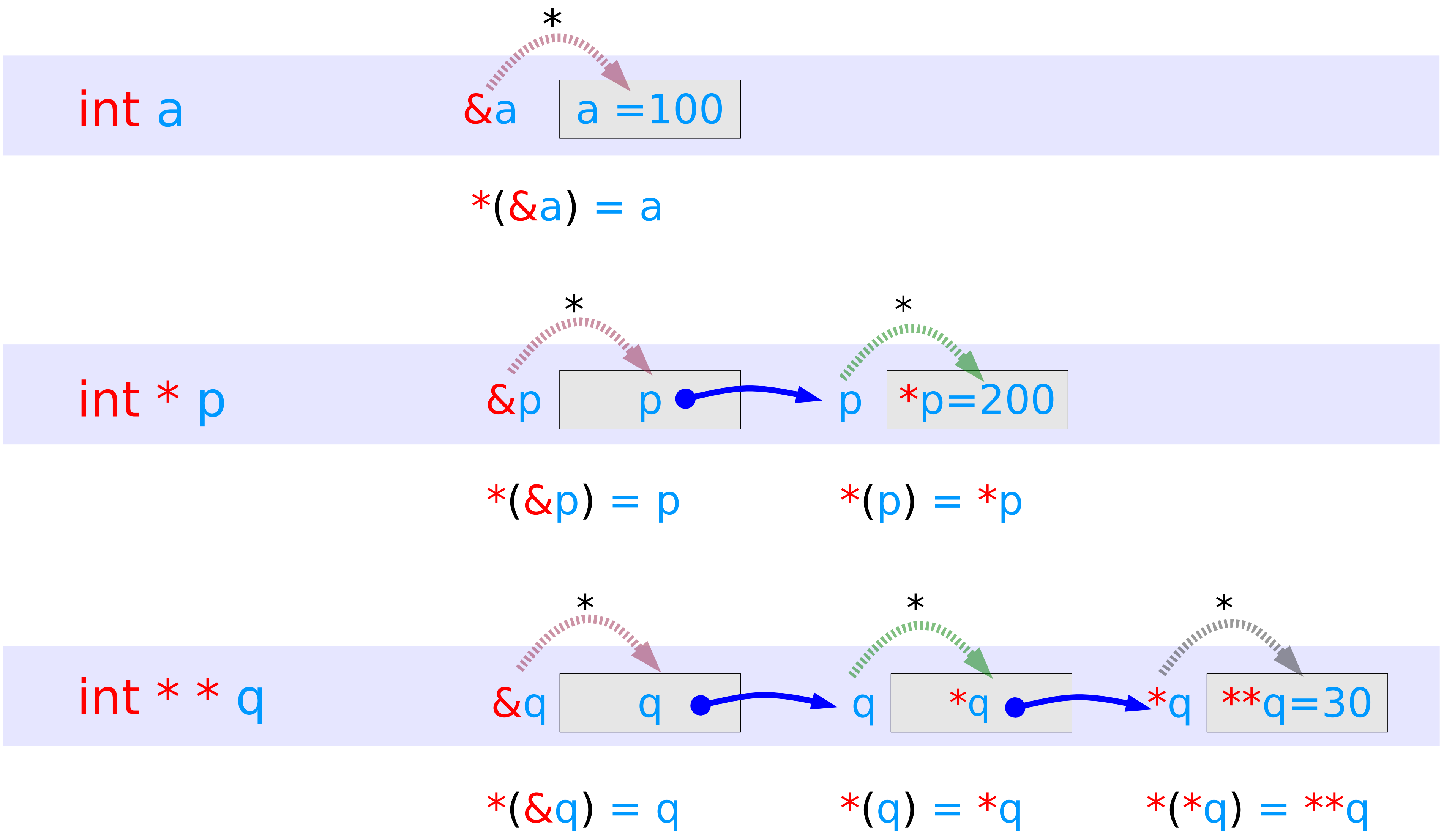
Double Indirect Access

Dereference Operator *

the content of the pointed location

address	value	
&q	q	address
q	*q	address
*q	**q	integer

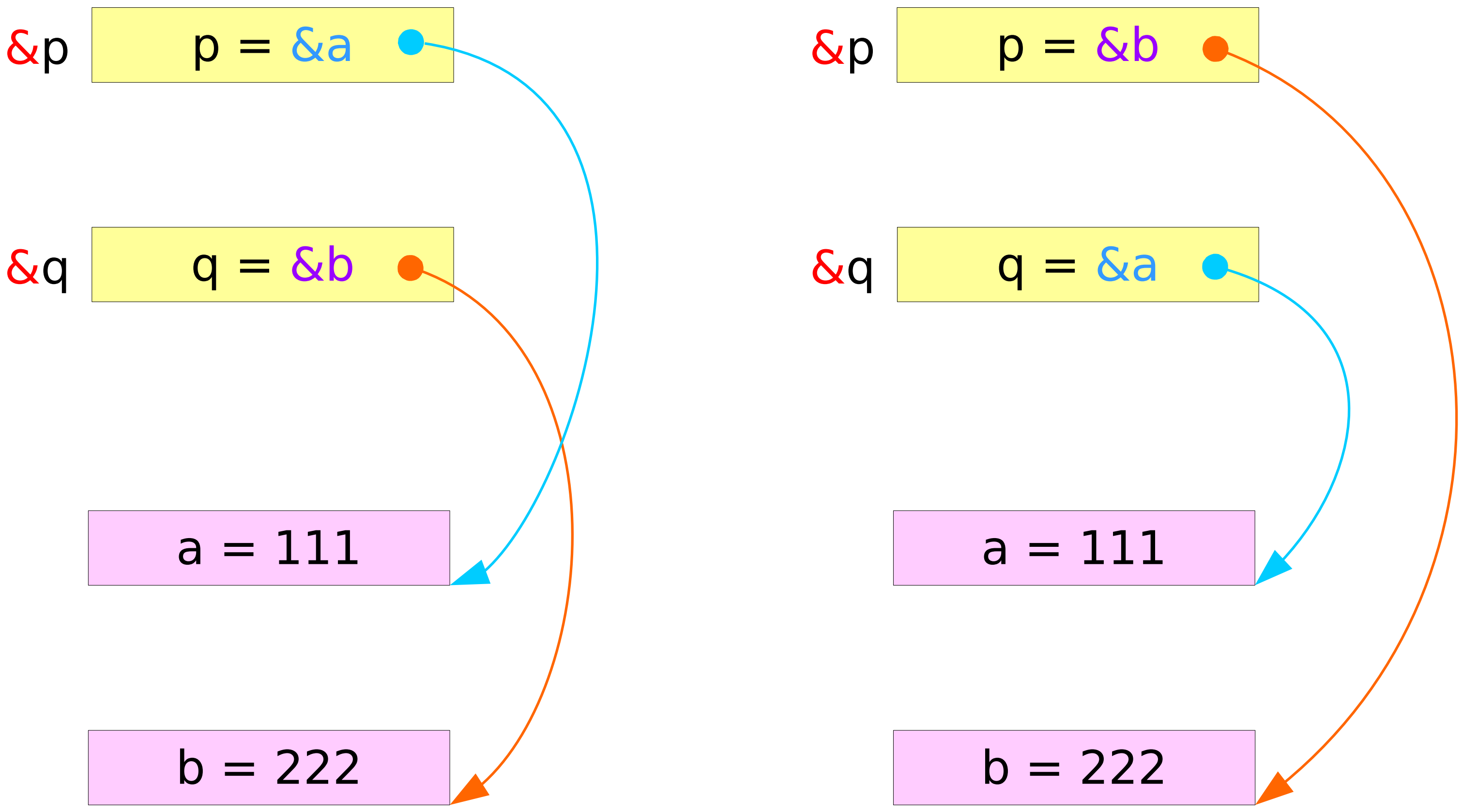
Access Data Via Pointer Variables (5)



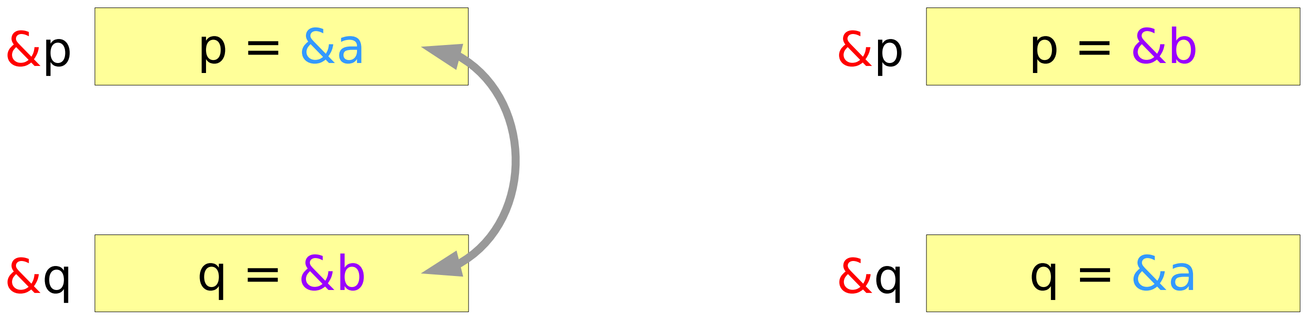
Swapping pointers

- pass by reference
- double pointers

Swapping integer pointers



Swapping integer pointers



```
int *p, *q;

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

function call

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

```
int        a [4];

int *      b [4];
```

Array name a holds the starting <u>*address*</u>

$\underbrace{\texttt{int}}$ a   [4]

No. of elements = 4

Type of each element

Array name b holds the starting <u>*address*</u>

$\underbrace{\texttt{int *}}$ b   [4]

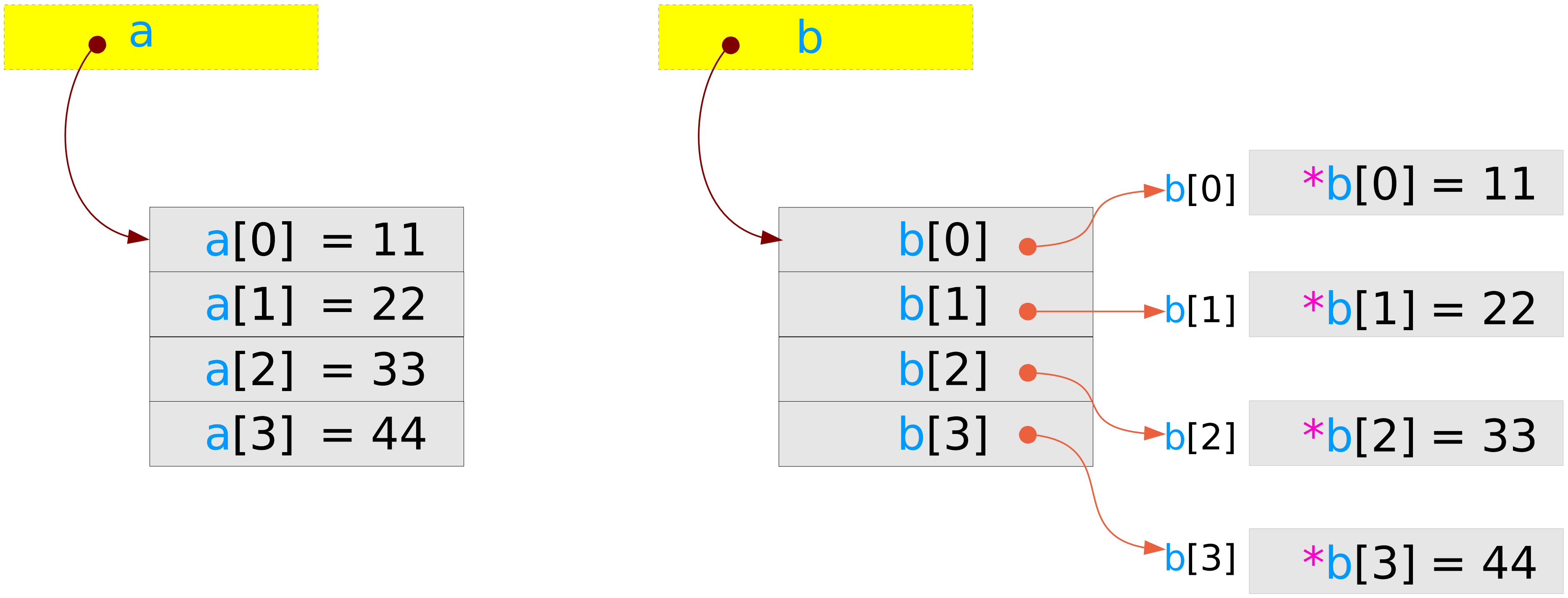
No. of elements = 4

Type of each element

Array of Pointers – variable view

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

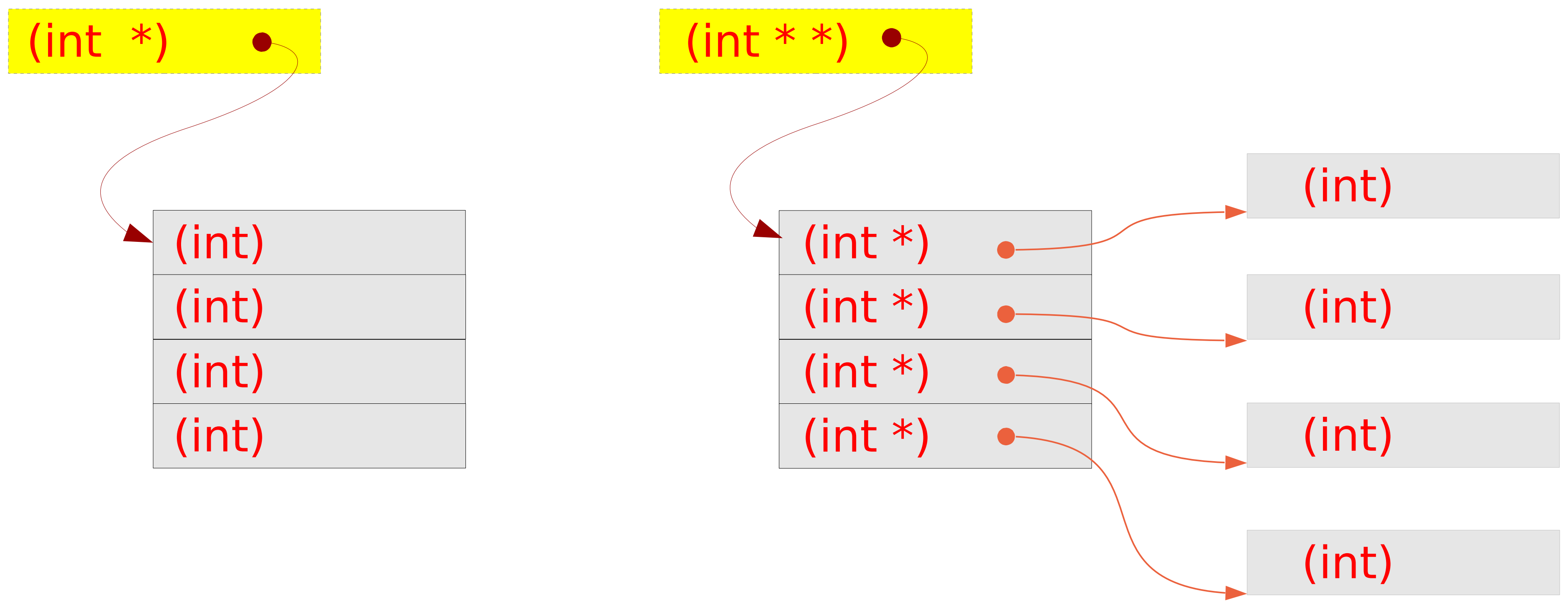
```
int *    b [4];
```



Array of Pointers – type view

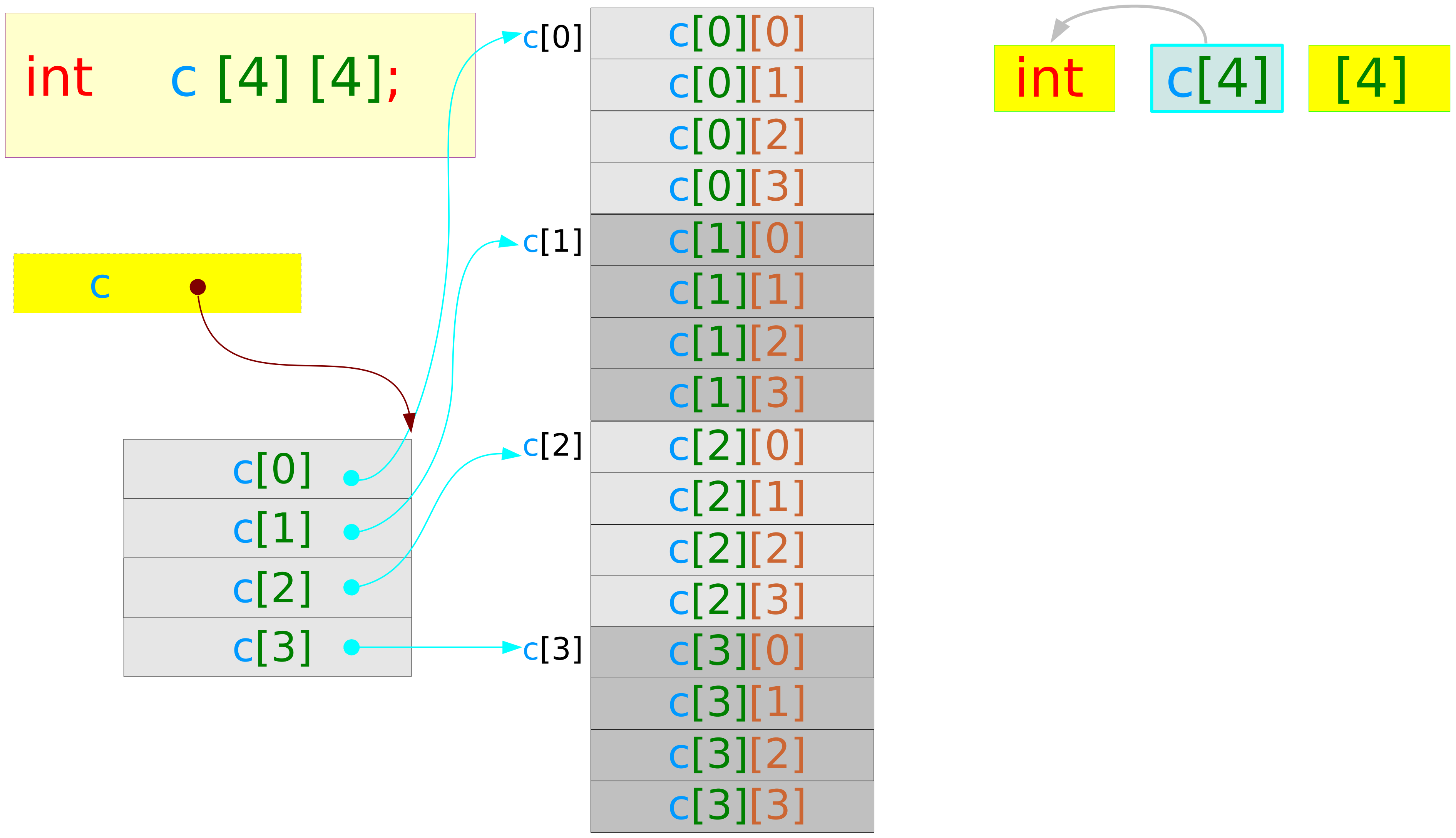
```
int        a [4];
```

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



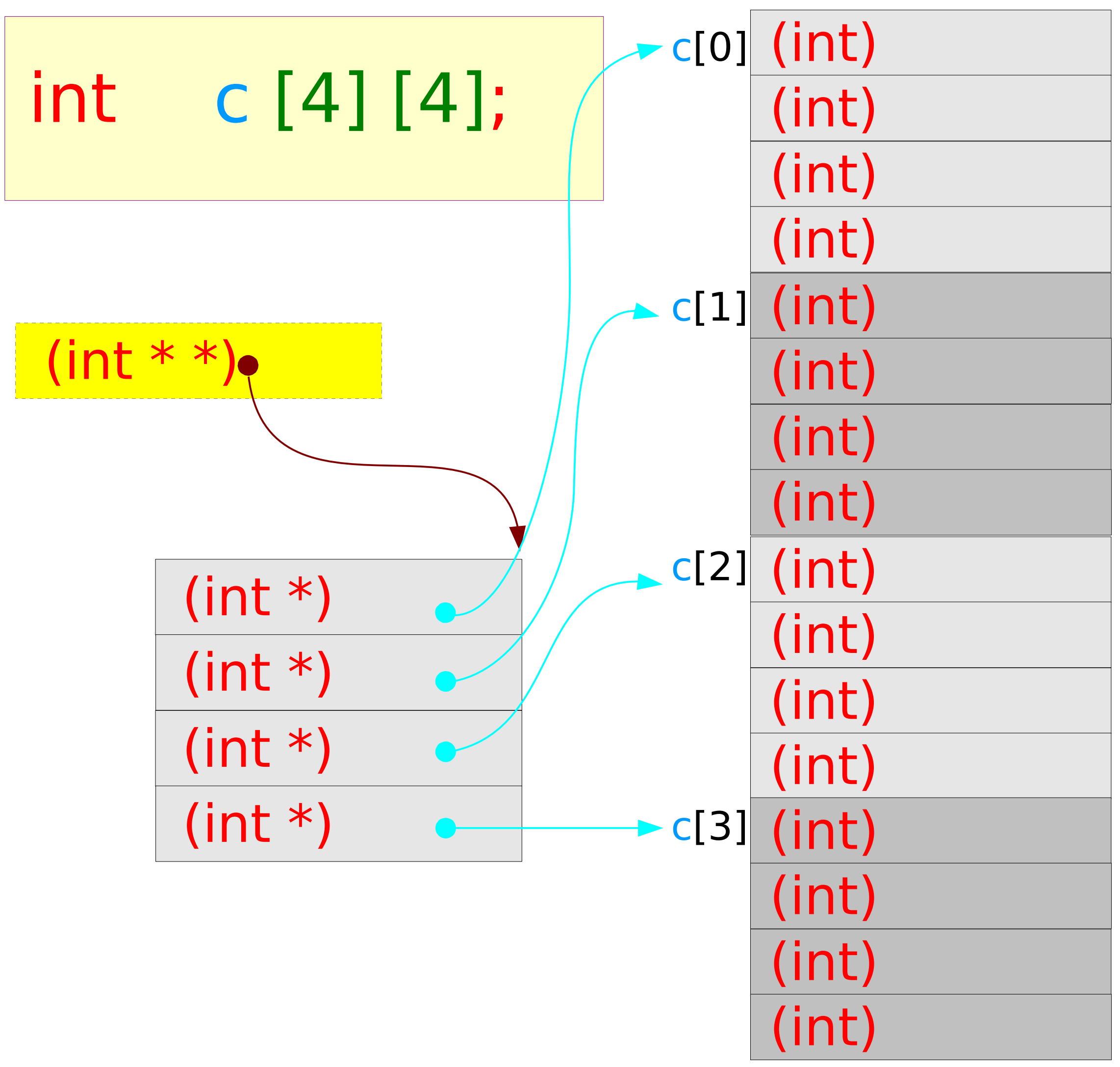
2-d Arrays

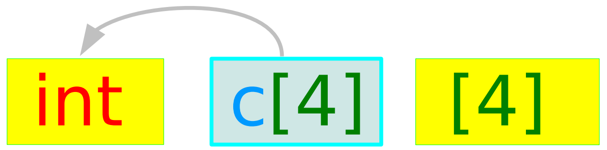
A 2-D Array


int c [4] [4];
c
c[0]
c[1]
c[2]
c[3]
c[0][0]
c[0][1]
c[0][2]
c[0][3]
c[1][0]
c[1][1]
c[1][2]
c[1][3]
c[2][0]
c[2][1]
c[2][2]
c[2][3]
c[3][0]
c[3][1]
c[3][2]
c[3][3]
int
c[4]
[4]

A 2-D Array

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





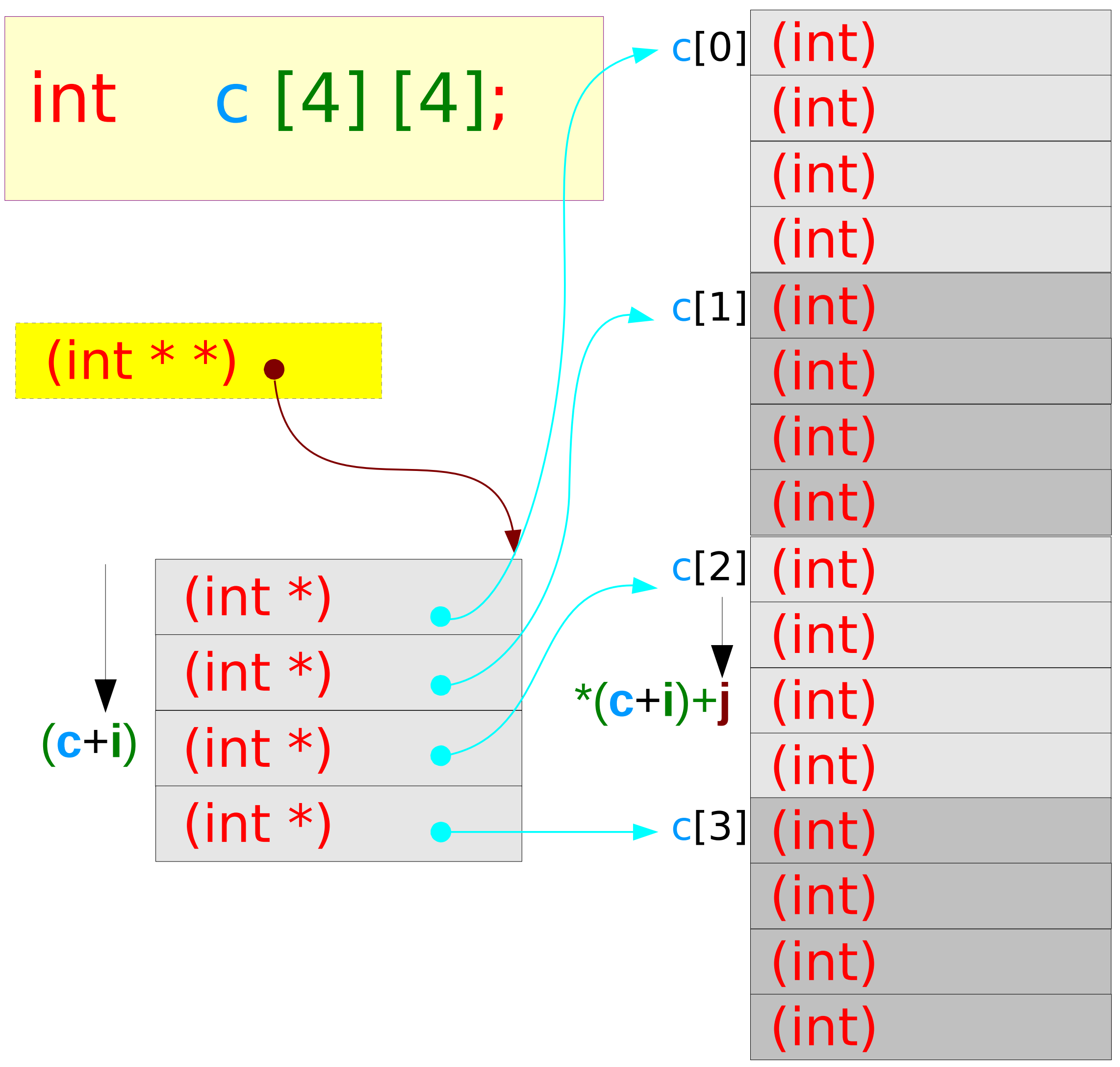
A 2-D Array via a double pointer

```
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];
```



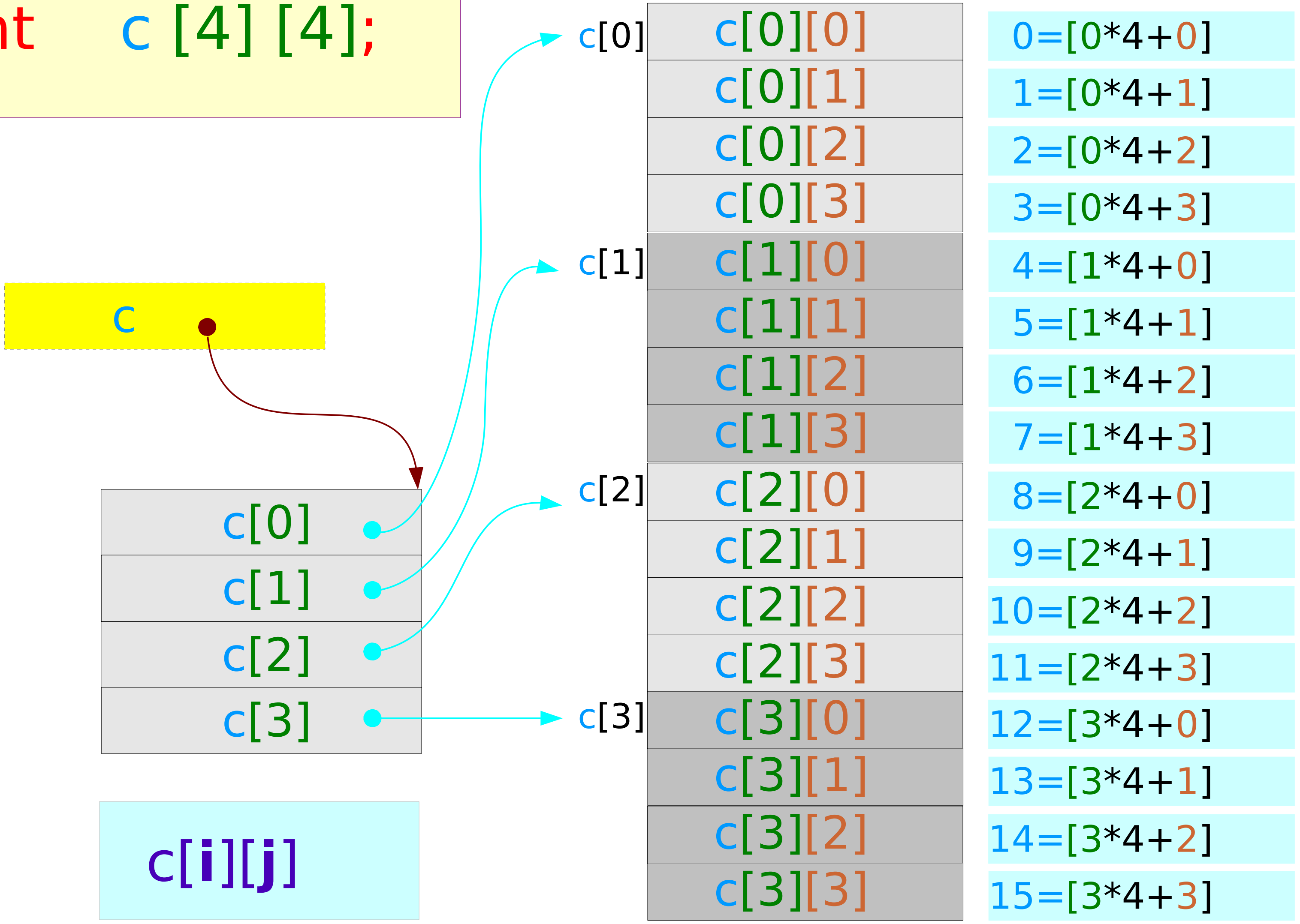
(c [i])[j] ➡

(*(c+i))[j] ➡

((c+i)+j)

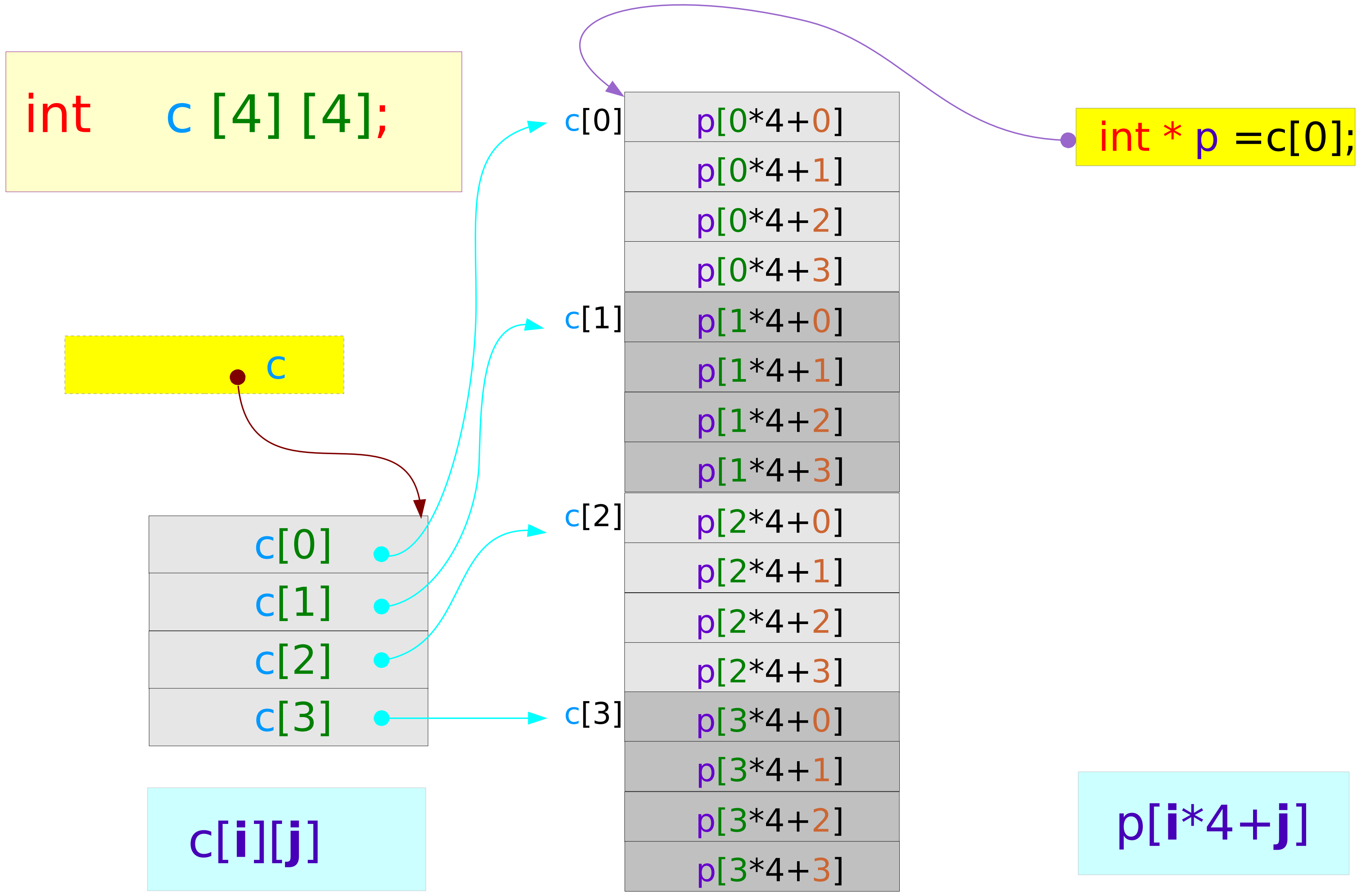
A 2-D array via a single pointer

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



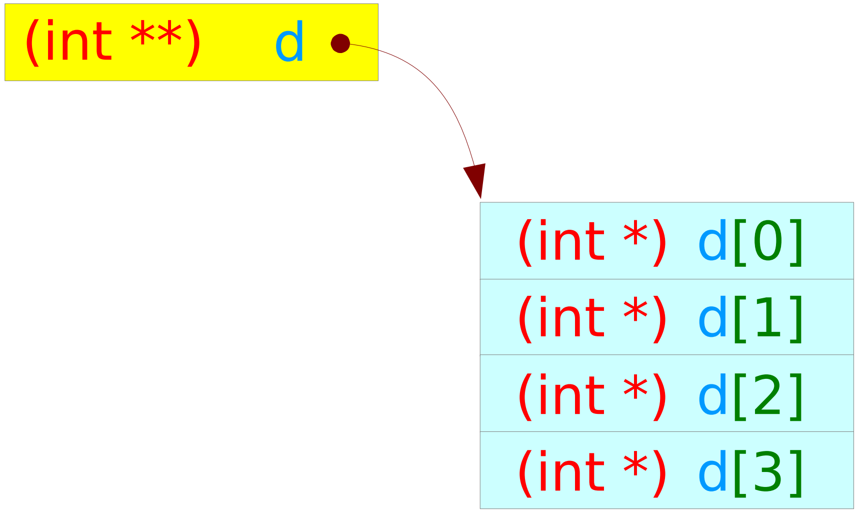
c[**i**][**j**]

A 2-D array via a single pointer



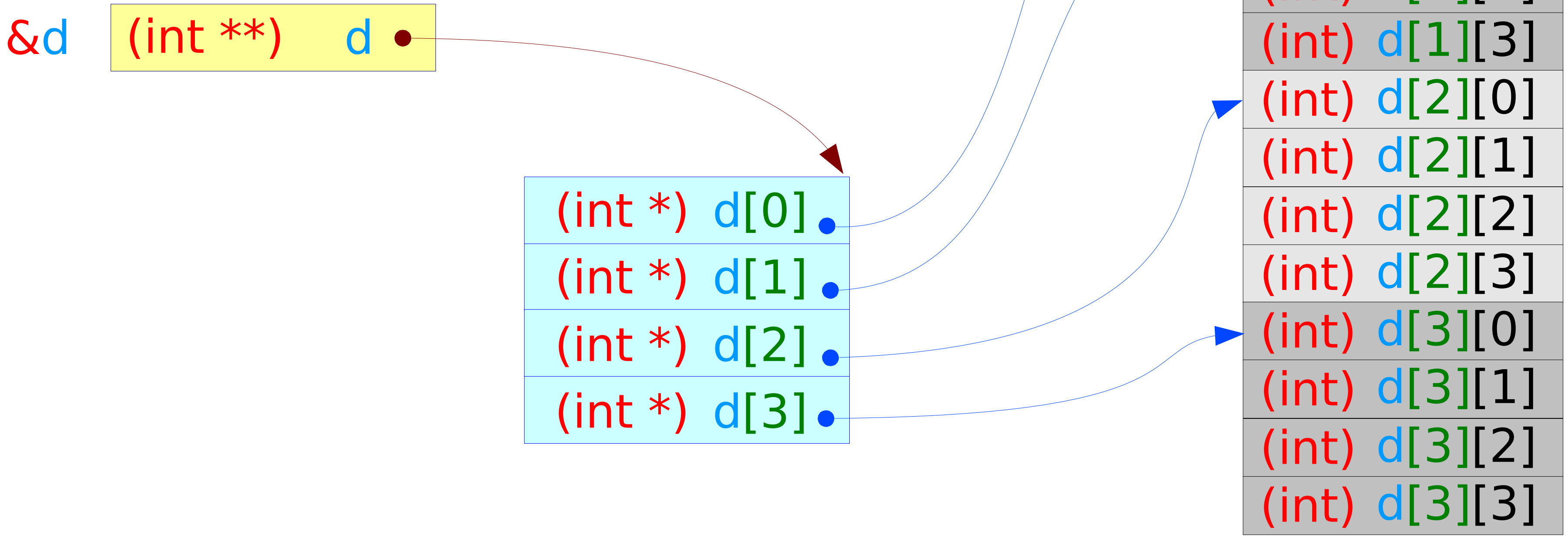
2-D Array Dynamic Memory Allocation (1)

```
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)

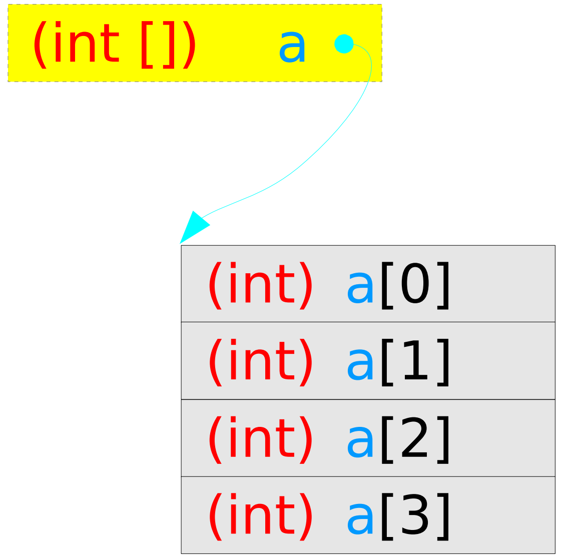
```
int **  d ;

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

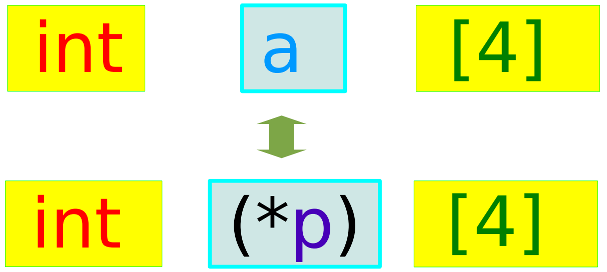


Pointer to Arrays

Pointer to array (1)

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





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]
```

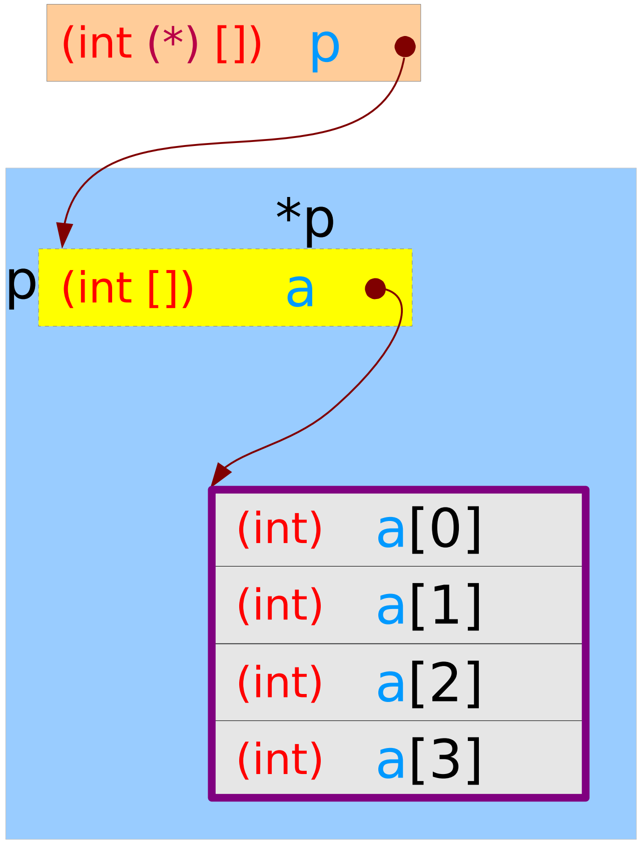
(*p) = a

&(*p) = &a

p = &a

sizeof(p)= 4 bytes

sizeof(*p)= 16 bytes



*p = a

an array with 4 integer elements

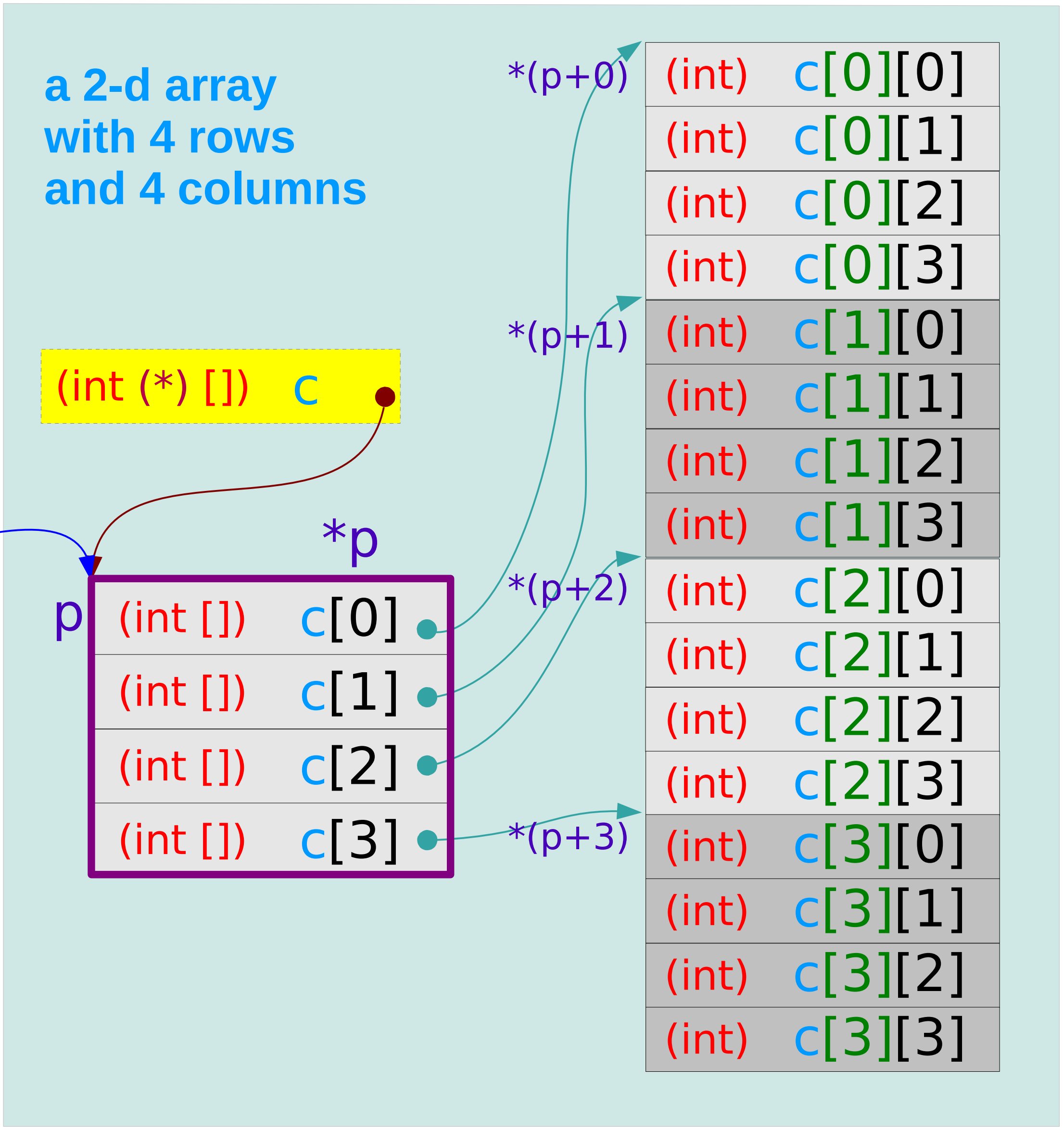
Pointer to array (3)

int (*p) [4] ;

int c[4] [4]

p = 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