Applications of Pointers (1A)

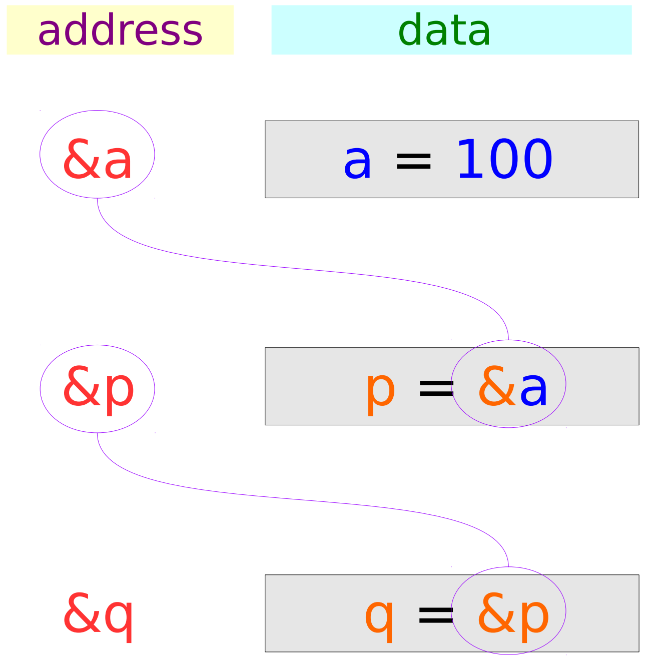
Young Won Lim
4/4/18

Variables and their addresses

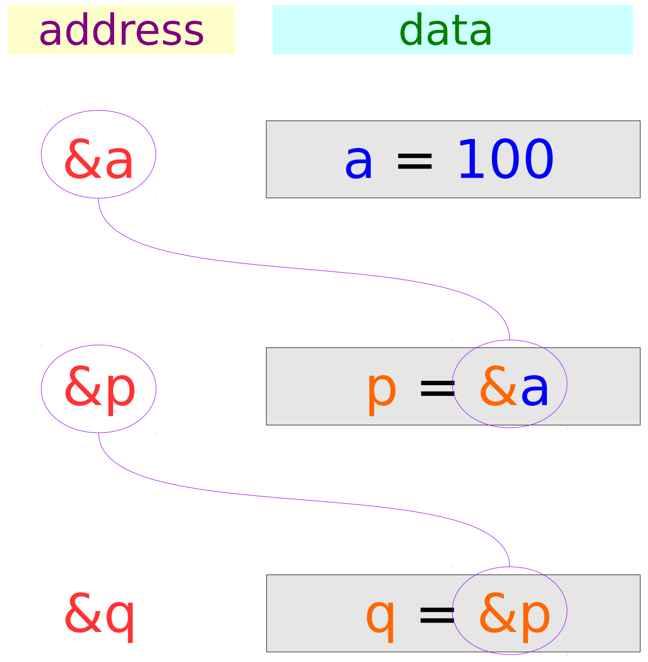
	address	data
`int    a;`	&a	a
`int  * p;`	&p	p
`int **q;`	&q	q

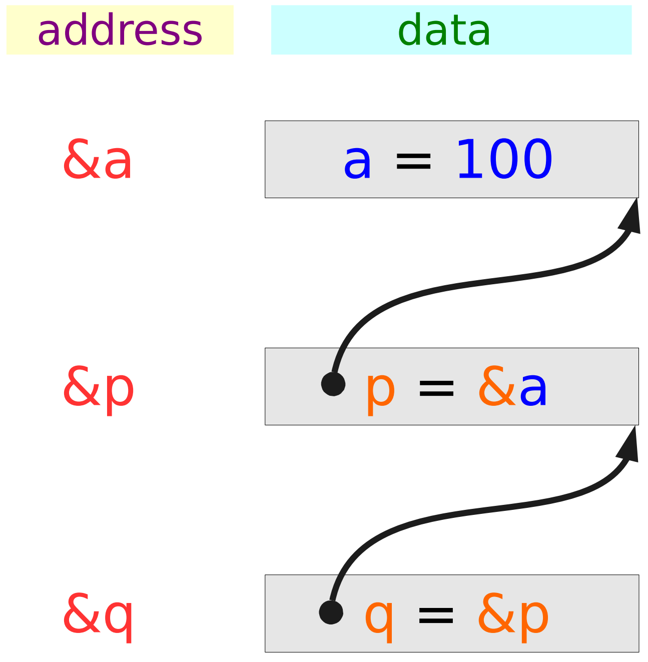
Initialization of Variables

```
int    a = 100;

int  * p = &a;

int **q = &p;
```



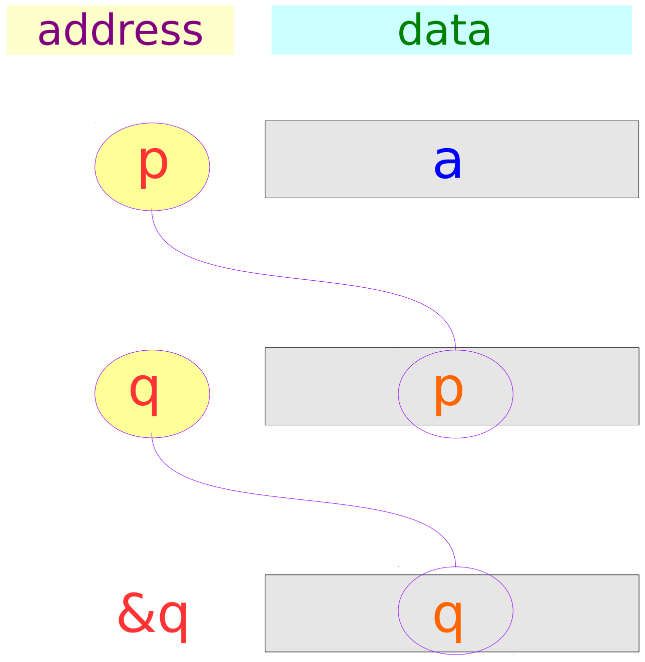
Traditional arrow notations





Pointed addresses : p, q

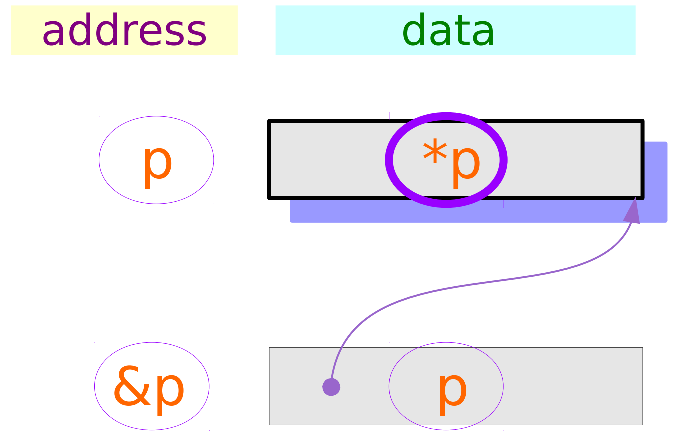
```
int    a;

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

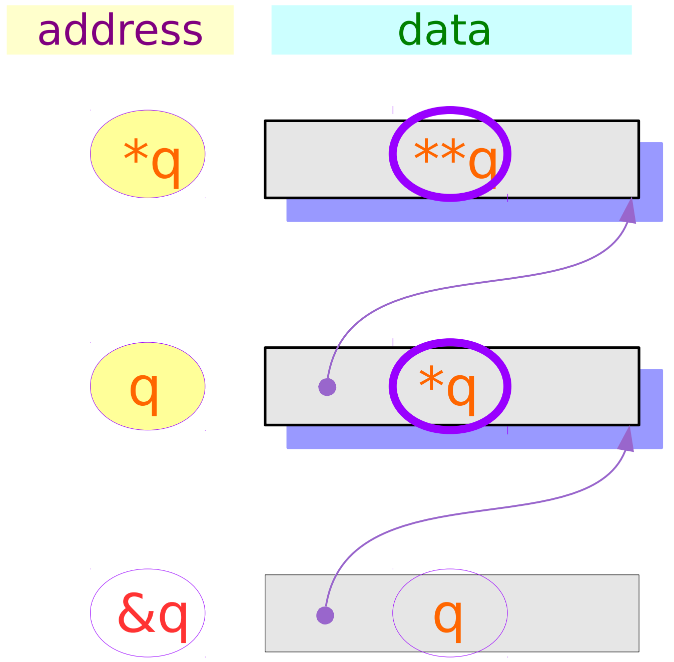
p = &a ➡ *p ≡ a

```
  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 | Variable aliasing

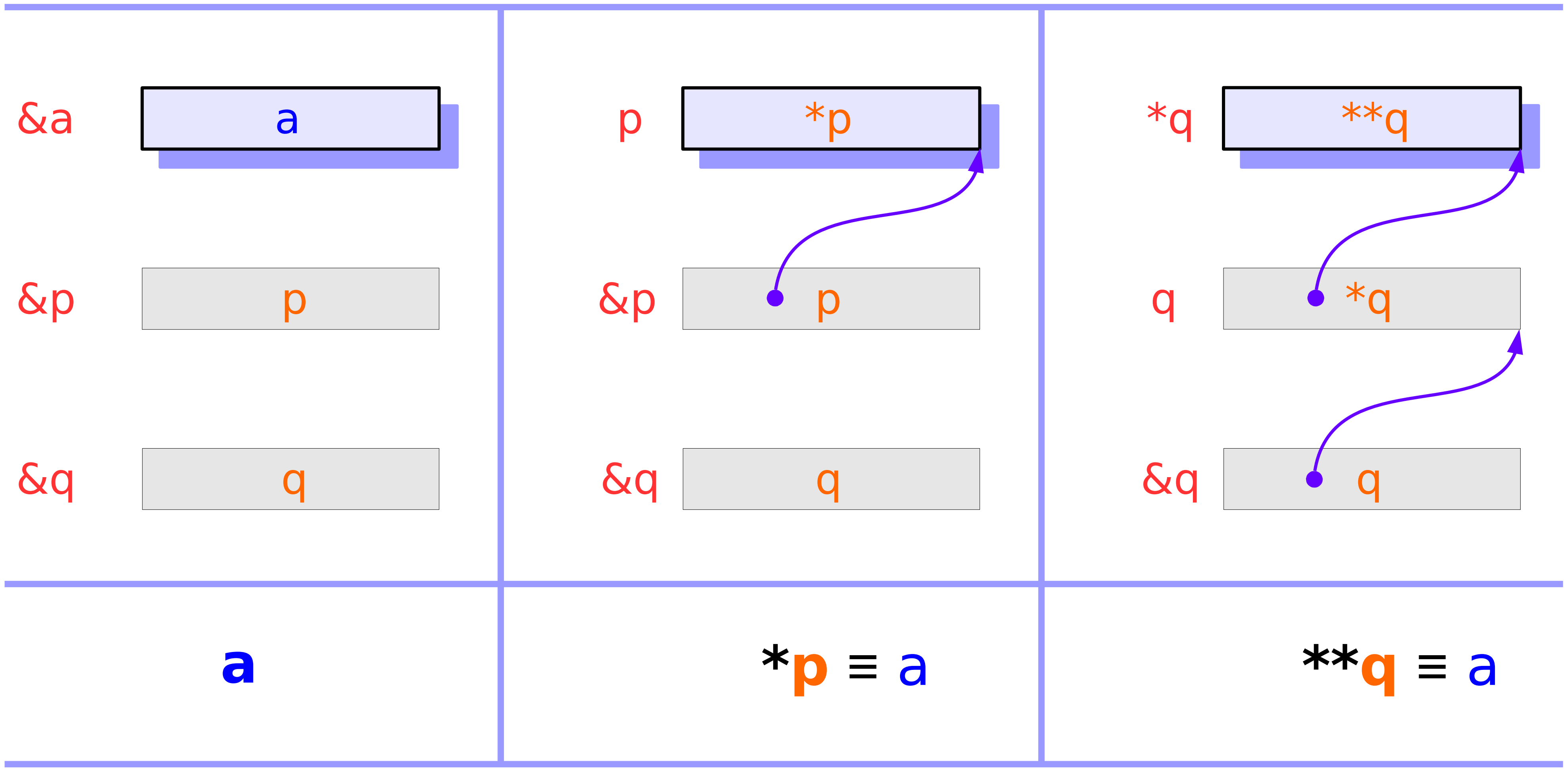
p = &a ➡ *p ≡ a

q = &p ➡ *q ≡ p
➡ **q ≡ a

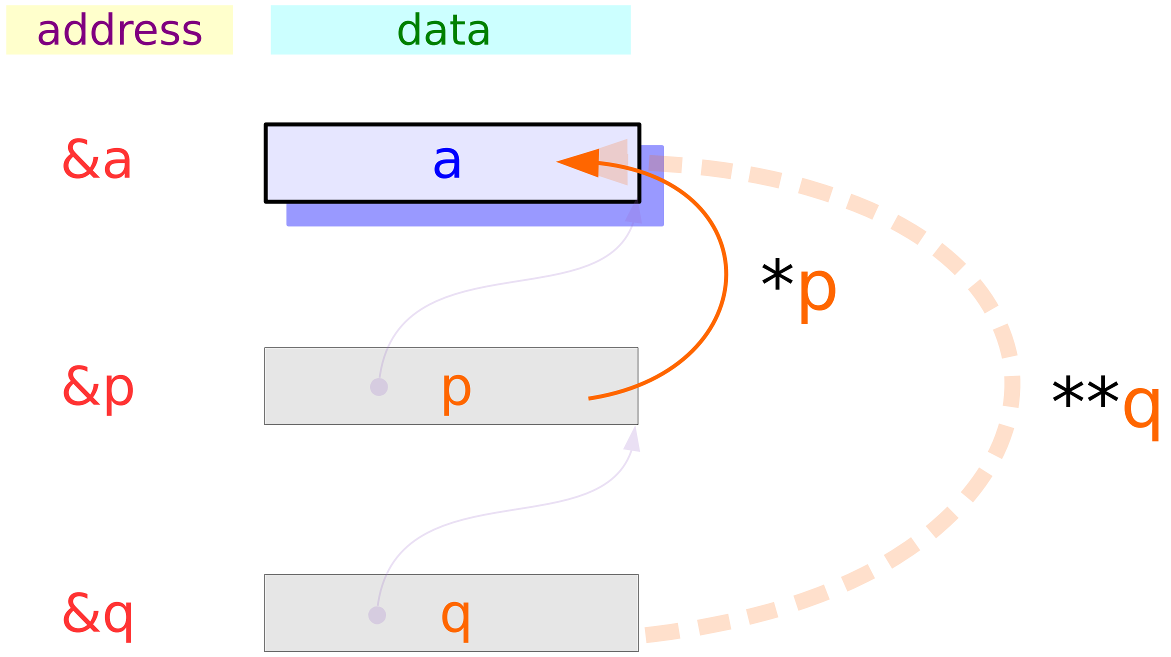
```
   q  ≡    &p
*(q) ≡ *(&p)
* q  ≡      p
**q  ≡    *p
**q  ≡     a
```

Relations after address assignment

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



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

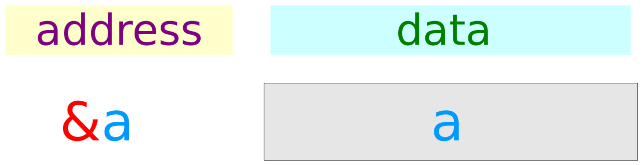


1) Read / Write a
2) Read / Write *p
3) Read / Write **q

Variables

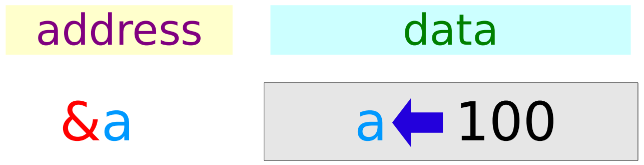
```
int    a;
```

a can hold an *integer*



```
a = 100;
```

a holds 100



Pointer Variables

```
int *      p;
```

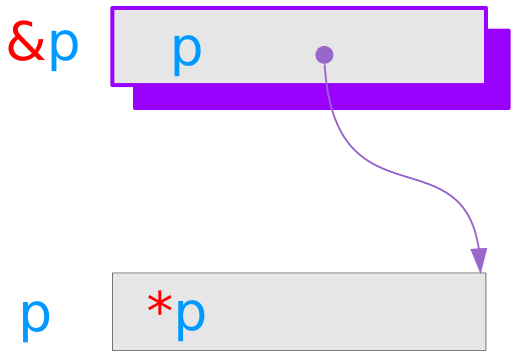
p can hold an ***address***

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

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

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

***p** holds
a int type ***data***



Pointer to Pointer Variable

```
int **     q;
```

q holds an *address*

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

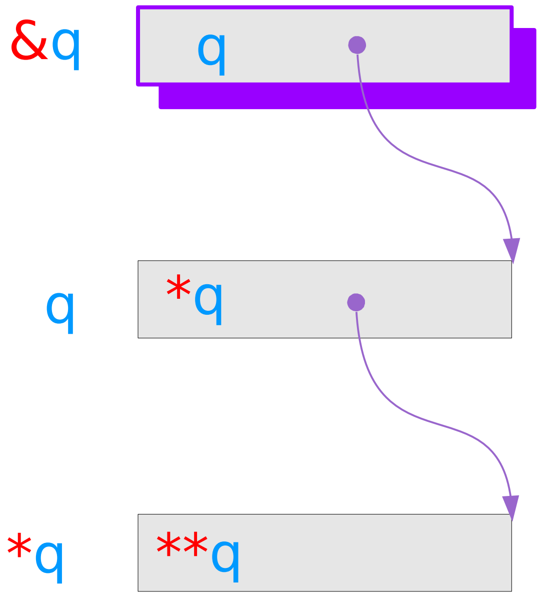
q holds an ***address*** of
a pointer to int type data

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

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

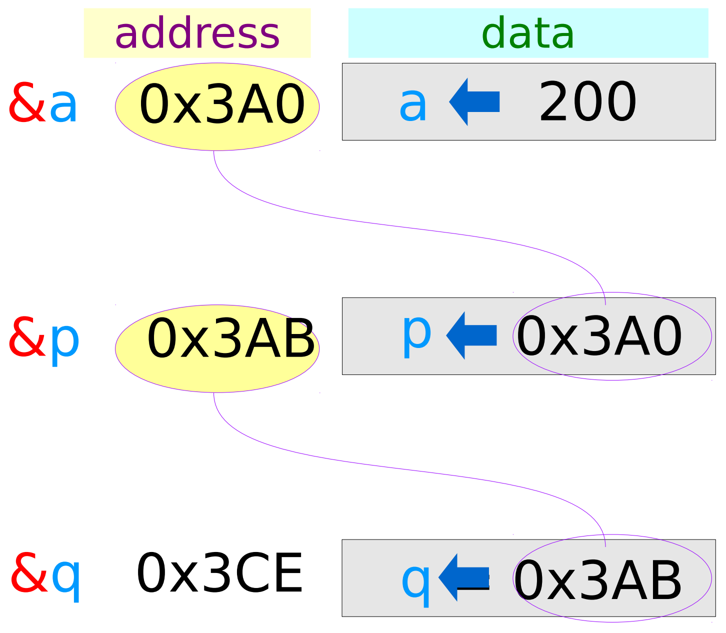
$\underbrace{\texttt{int}}_{\text{int}}$ `**q;`

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



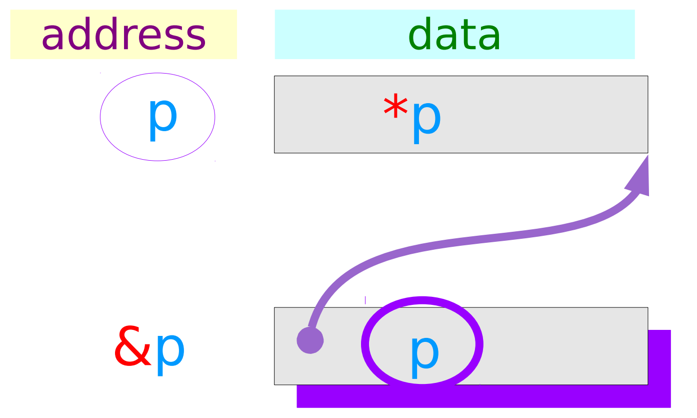
Pointer Variables Examples

```
int      a = 200;

int *    p = & a;

int **   q = & p;
```

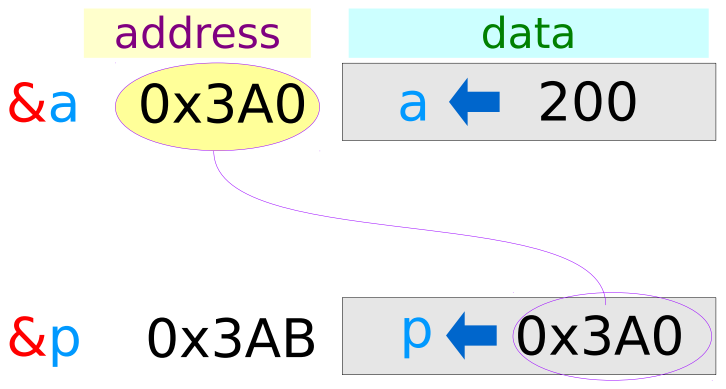


&q ➡	0x3CE
q ➡	0x3AB
*q ➡	0x3A0
**q ➡	200

Pointer Variable **p** with an arrow notation



using an arrow notation

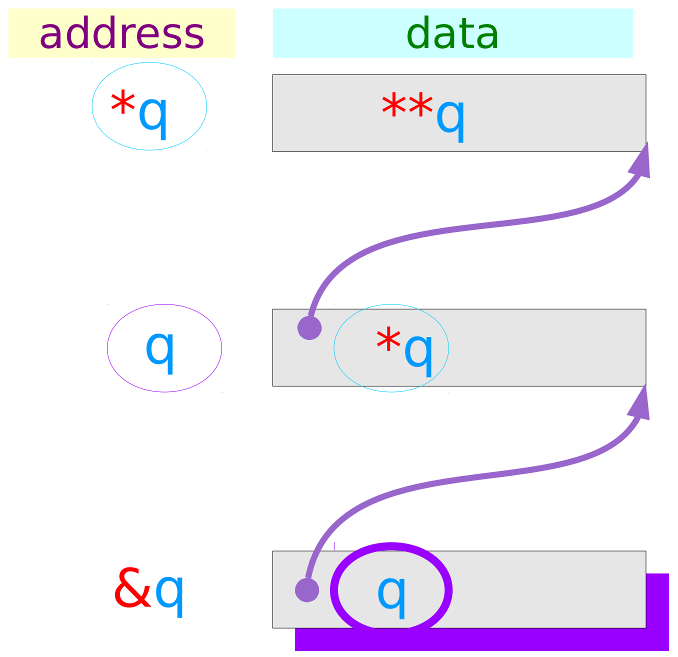


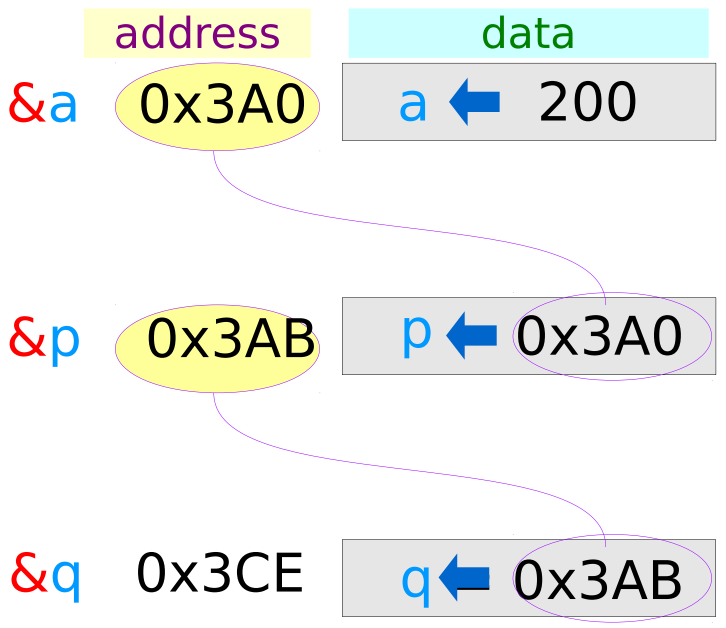
&p ➡ 0x3AB

p ➡ 0x3A0

*p ➡ 200

Pointer Variable **q** with an arrow notation





using an arrow notation

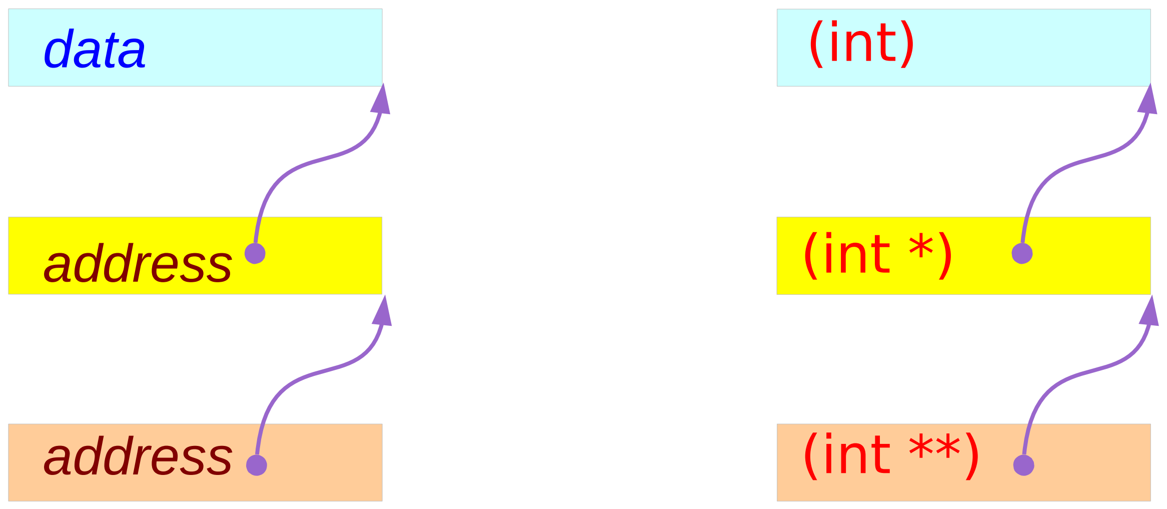
&q ➡ 0x3CE

q ➡ 0x3AB

*q ➡ 0x3A0

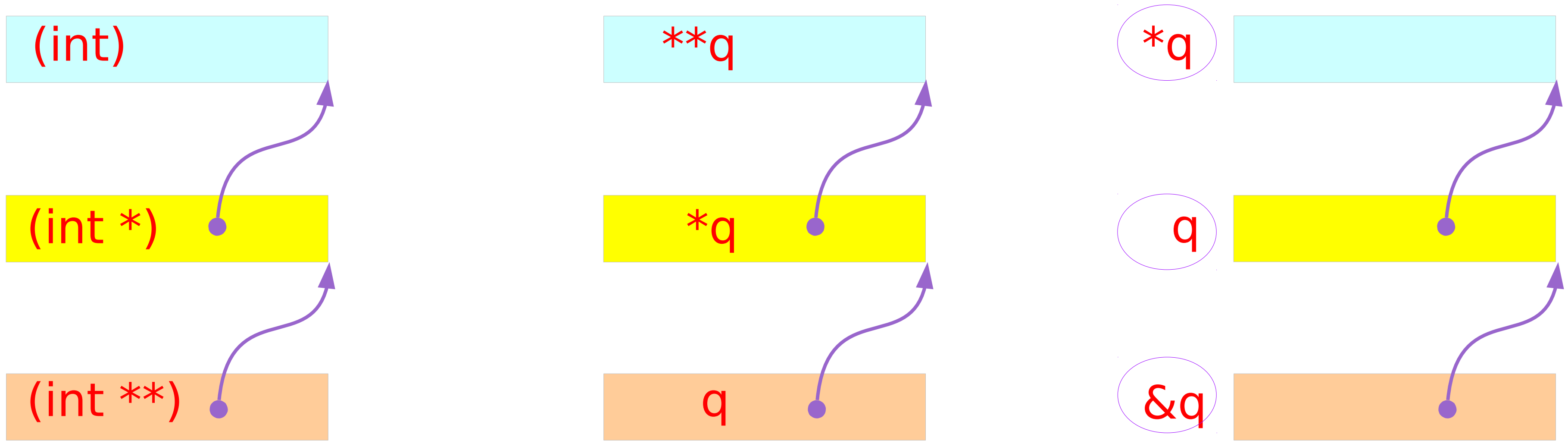
**q ➡ 200

The type view point of pointers



Types

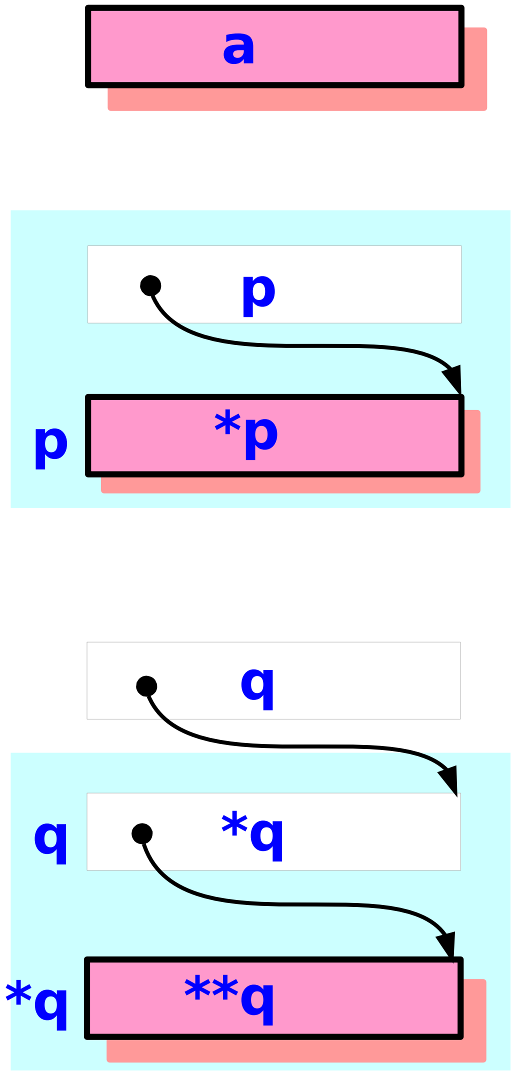
The different view points of pointers



Types **Variables** **Addresses**

Single and Double Pointer Examples (1)

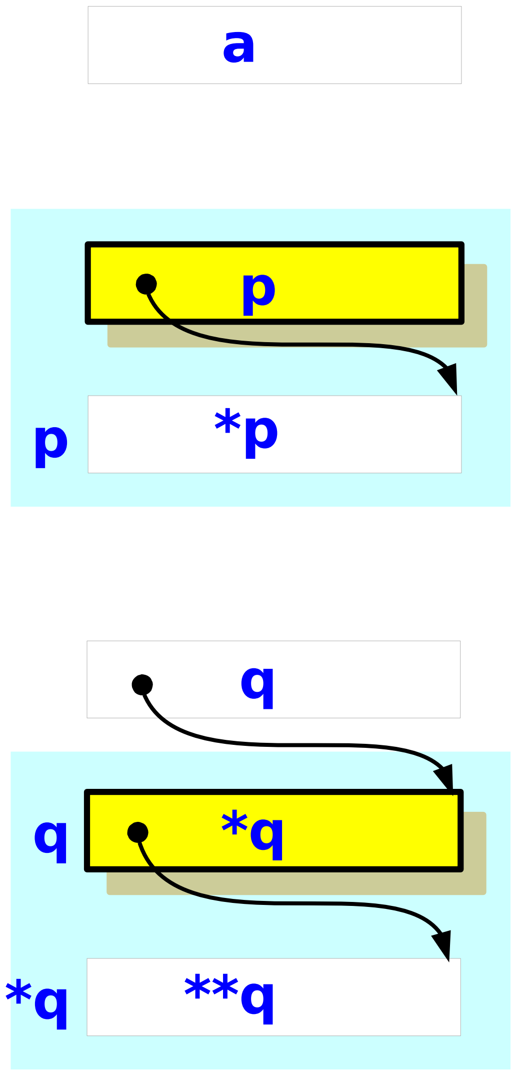
```
int    a  ;
int   *p  ;
int  **q  ;
```



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

Single and Double Pointer Examples (2)

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

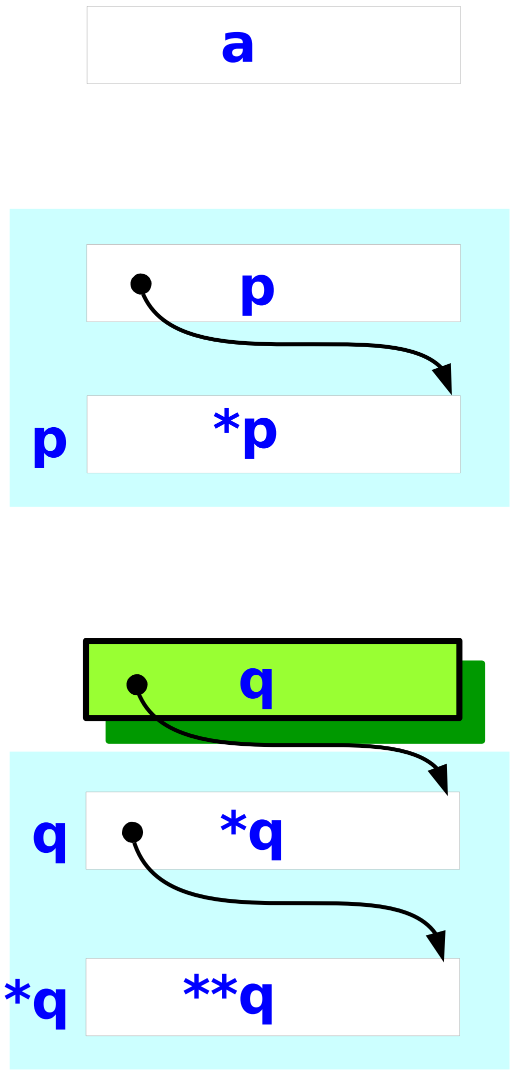


p and ***q :**
int pointer variables
(singlepointers)

Single and Double Pointer Examples (3)

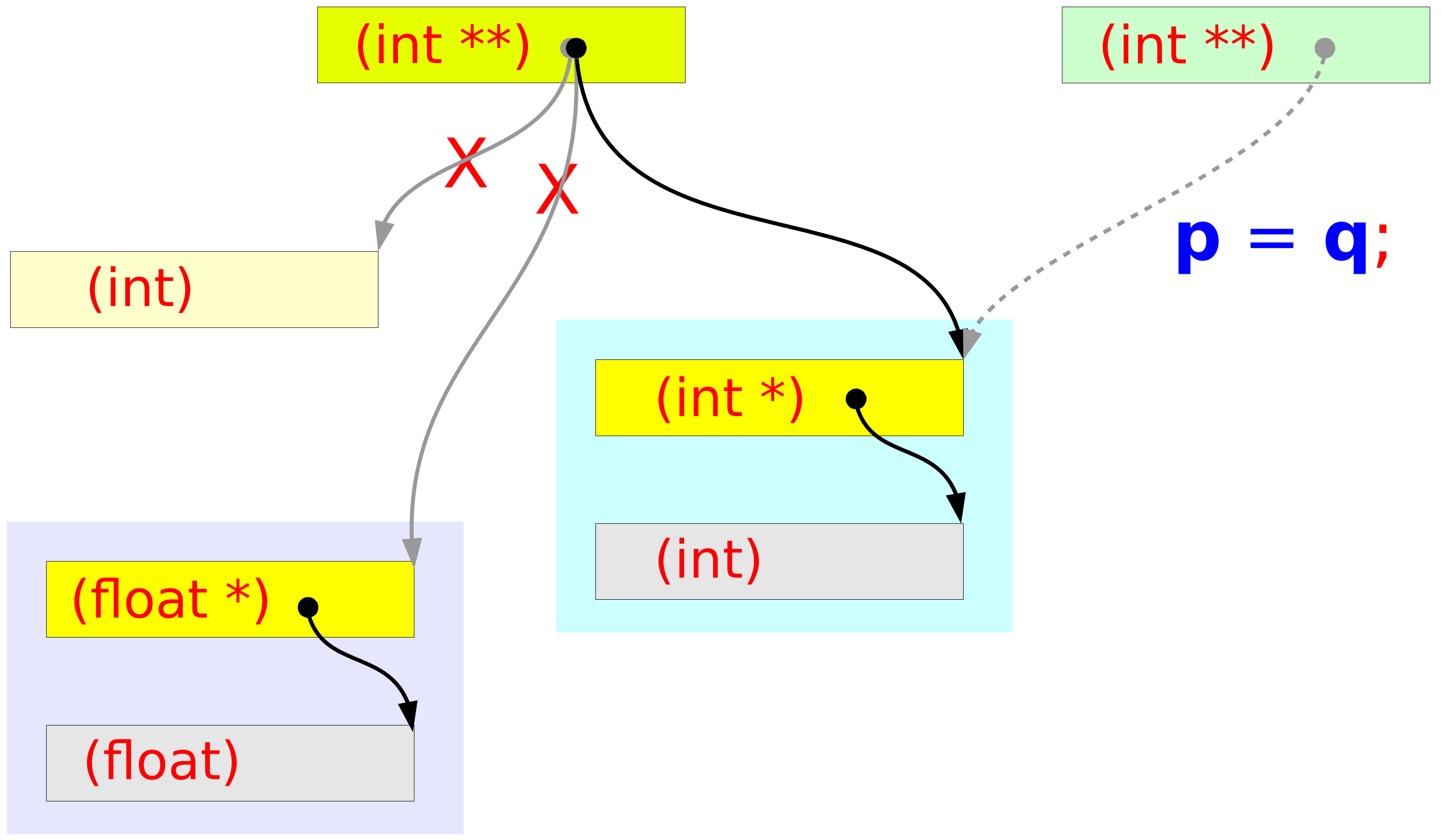
```
int     a ;
int *   p ;
int **  q ;
```

q :
double int pointer variables



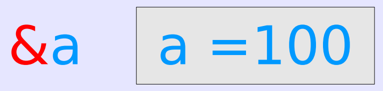
Values of double pointer variables

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



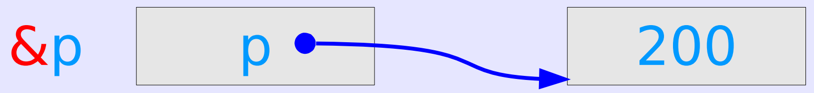
Pointed Addresses and Data

int a ;



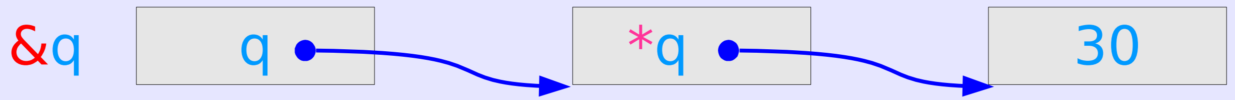
The variable a holds an integer data

int * p ;



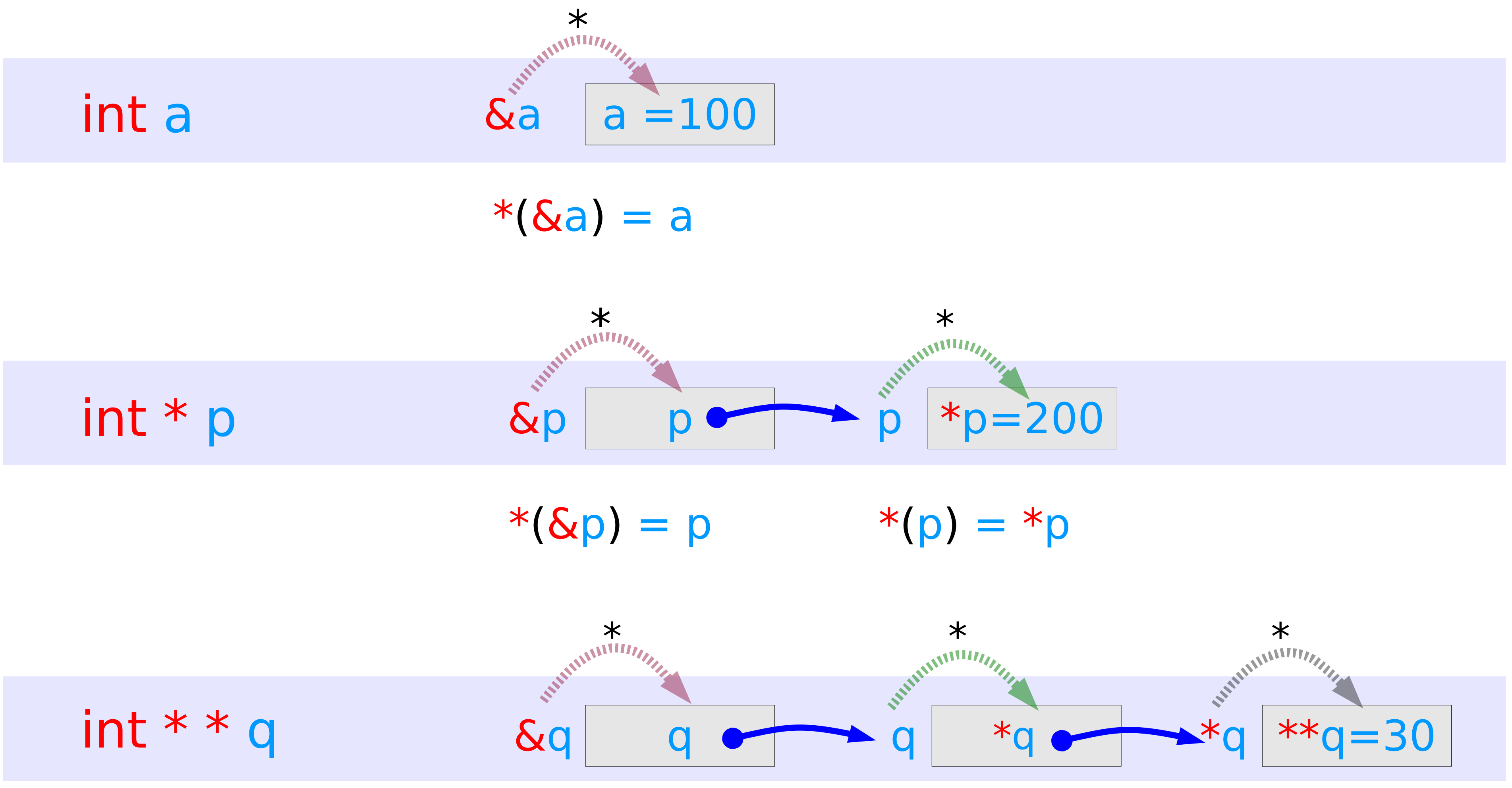
The **pointer** variable p holds an address,
at this address, an integer data is stored

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

Dereferencing Operations



*(&a) = a

*(&p) = p   *(p) = *p

*(&q) = q   *(q) = *q   *(*q) = **q

Direct Access to an integer **a**

```
int a ;
```

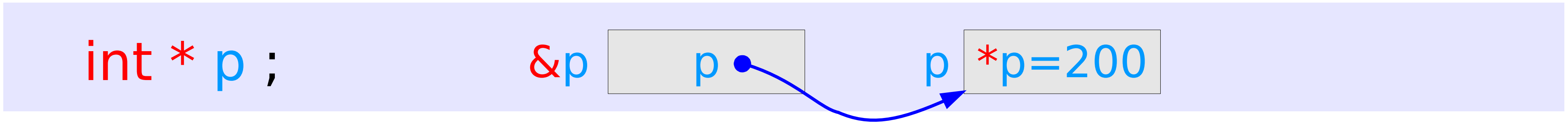
&a $\boxed{a = 100}$

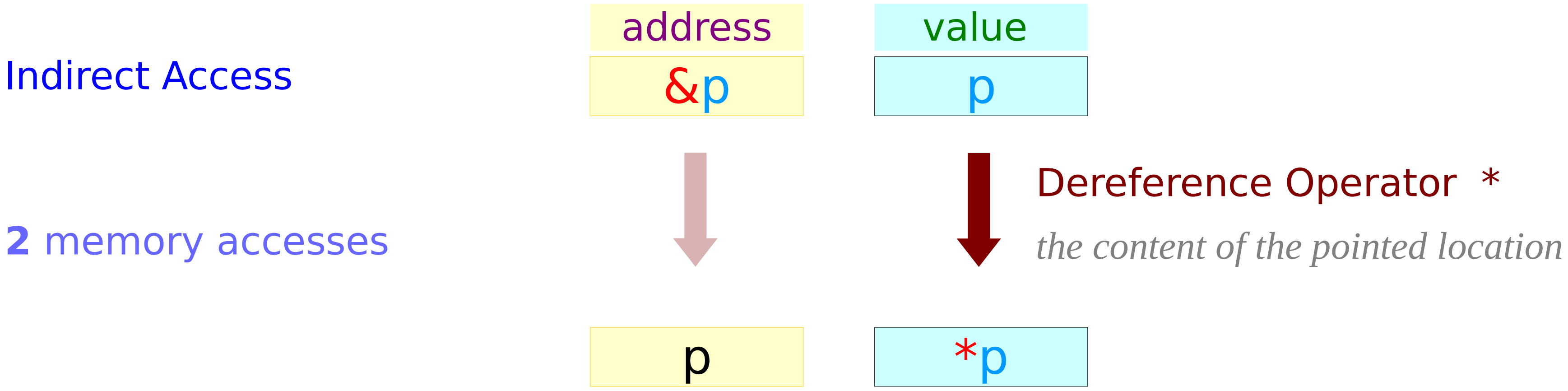
Direct Access

address	value	
&a	a	integer

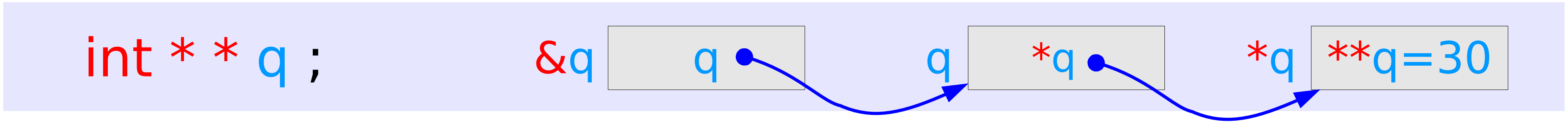
1 memory access

Indirect Access *p to an integer a



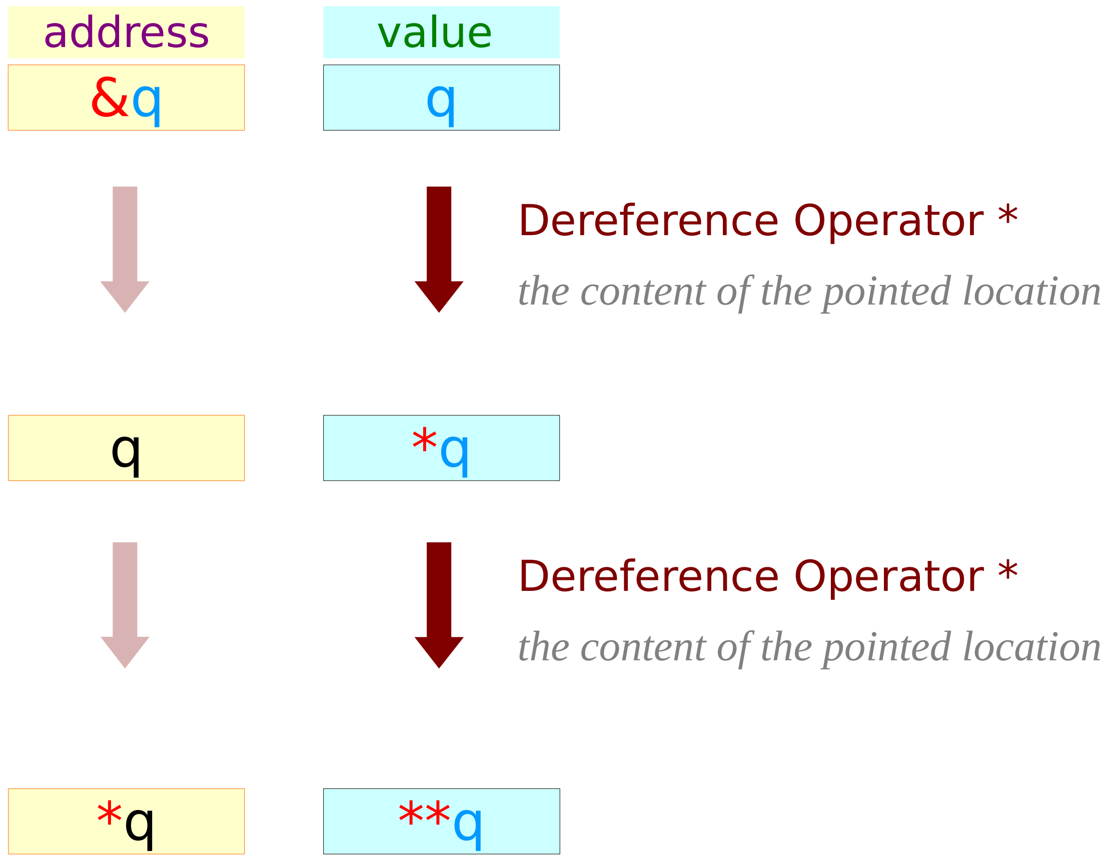


Double Indirect Access **q to an integer a

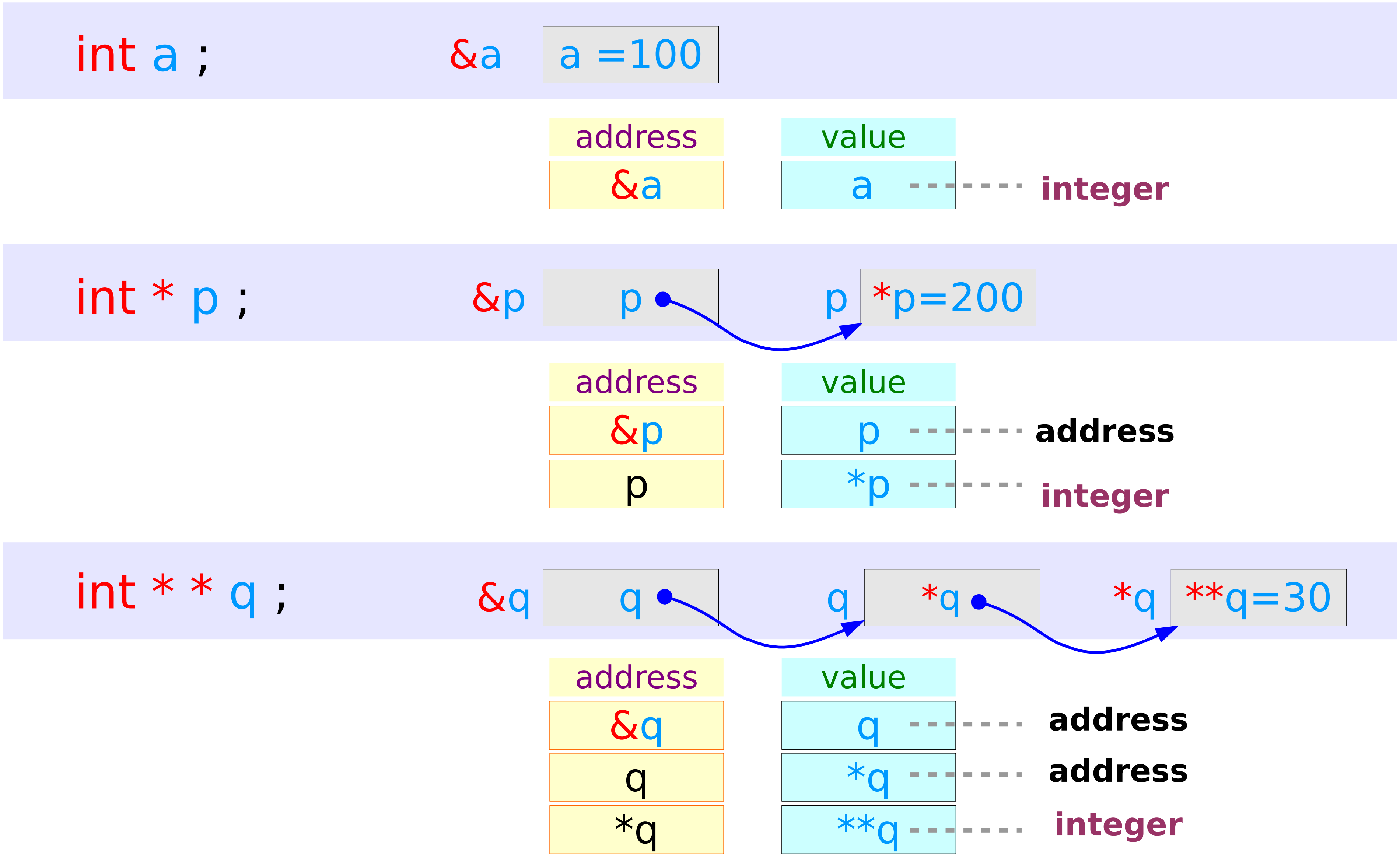


Double Indirect Access

3 memory accesses


address
value
&q
q
Dereference Operator *
the content of the pointed location
q
*q
Dereference Operator *
the content of the pointed location
*q
**q

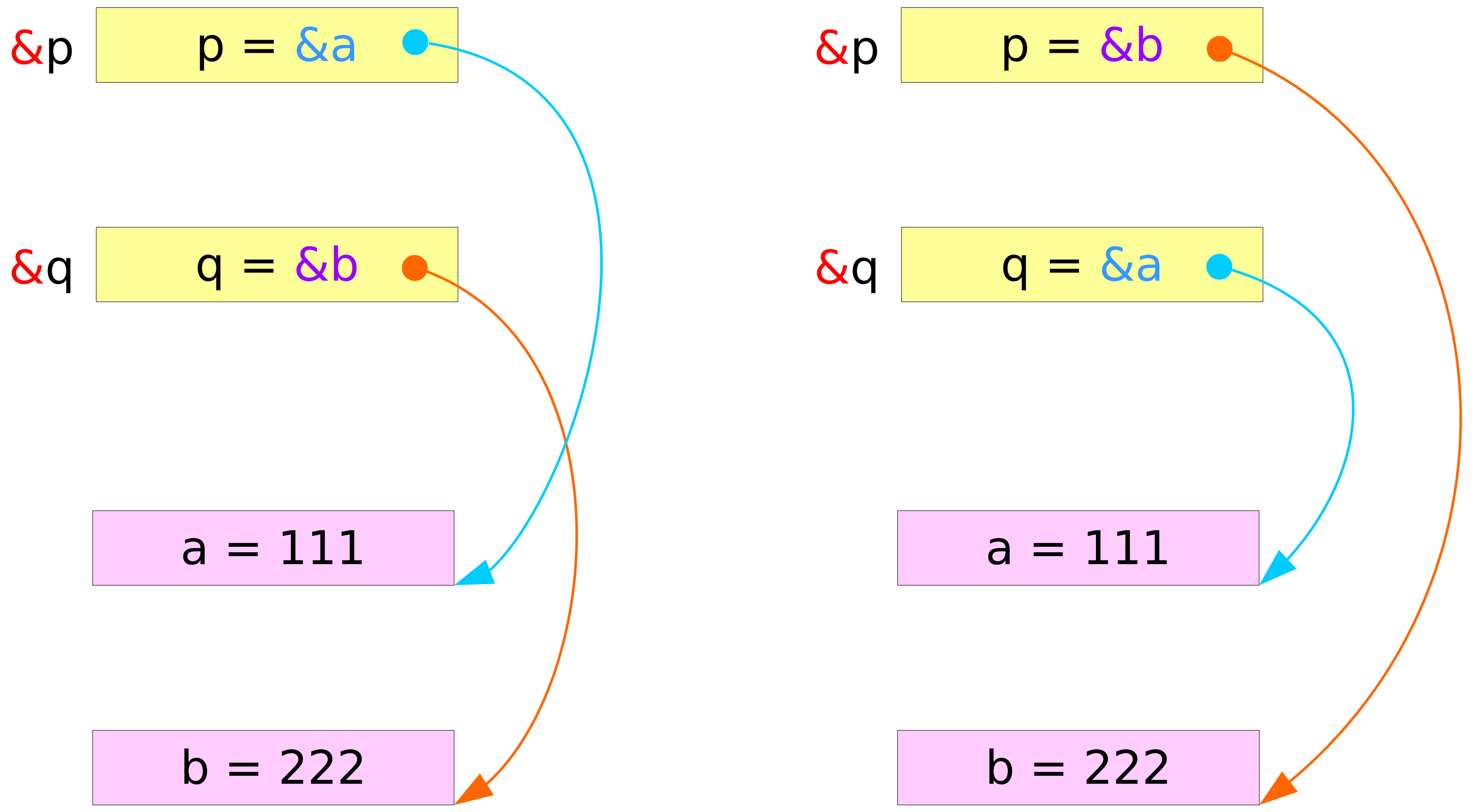
Values of Variables



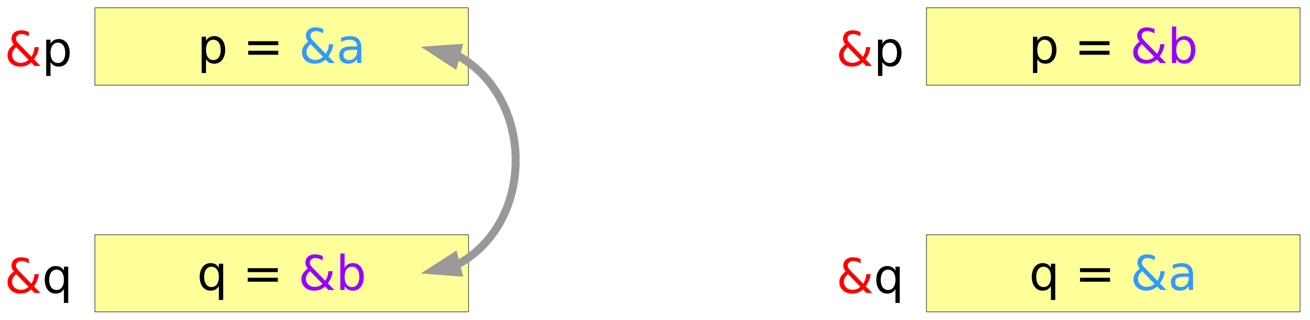
Swapping pointers

- 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 **	m	int **	n
int *	*m	int *	*n

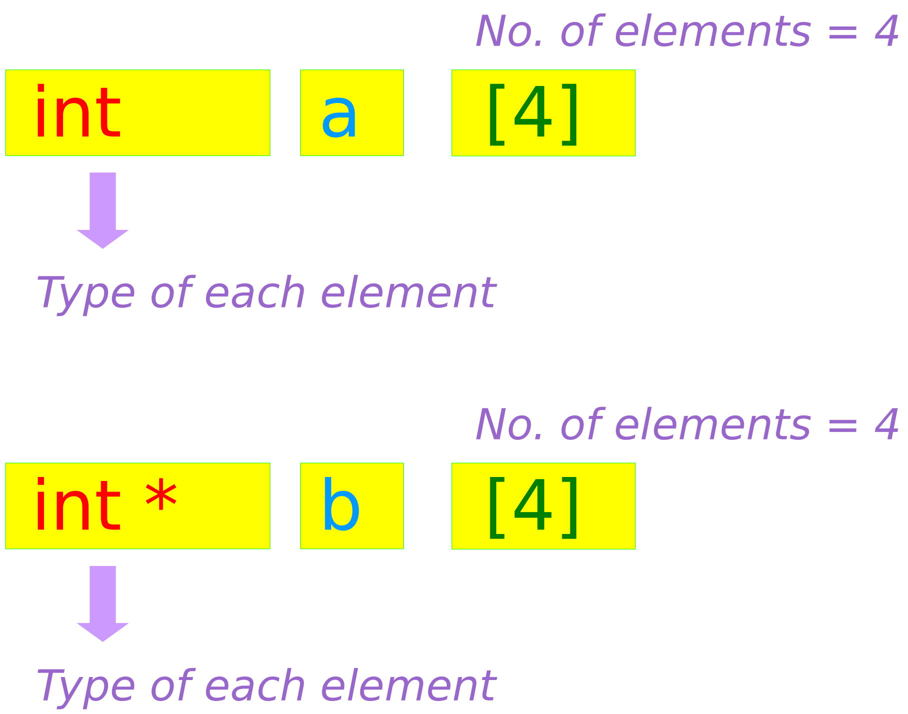
int *	tmp

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

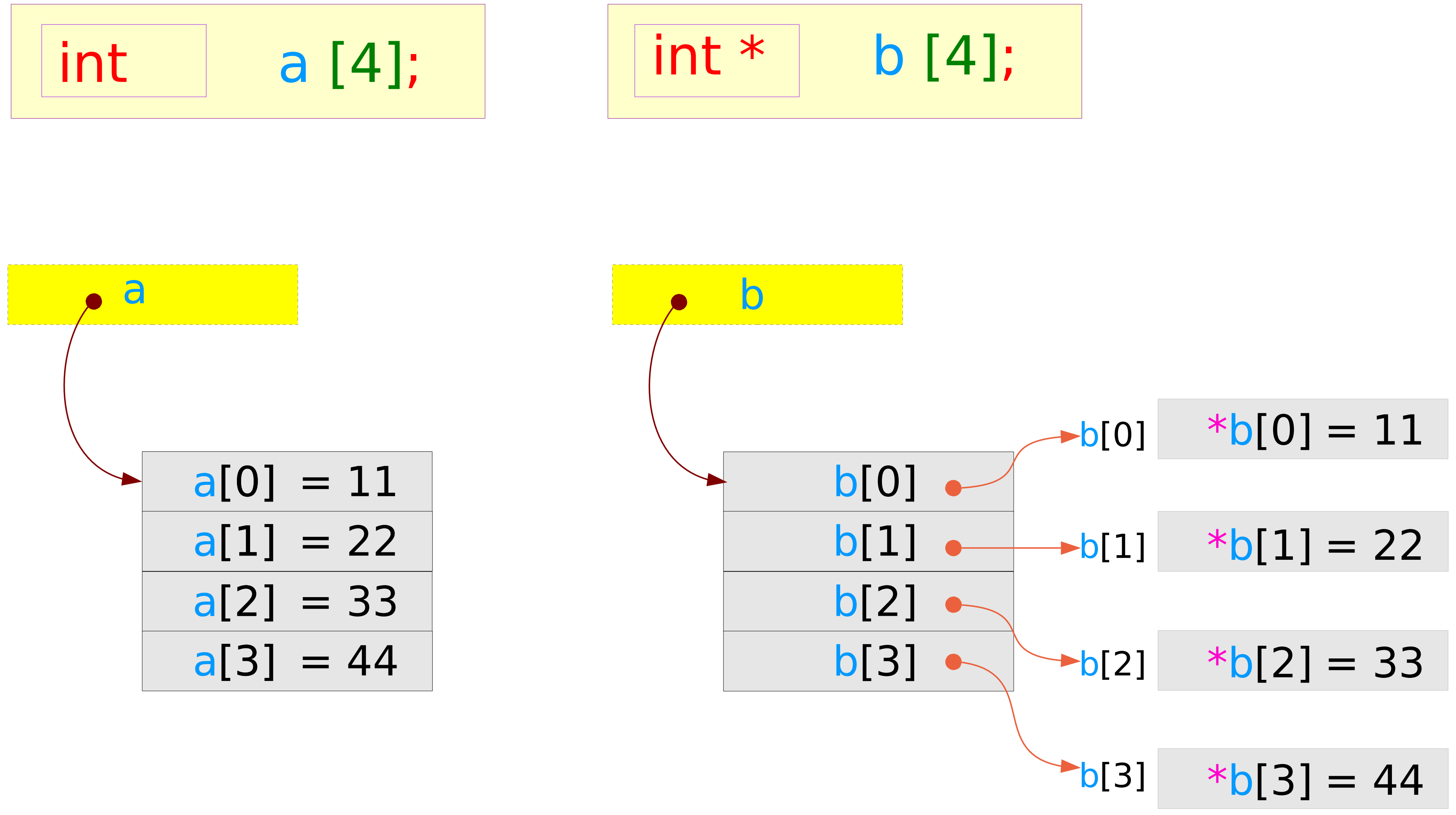
Array of Pointers

Array of Pointers

```
int       a [4];

int *     b [4];
```



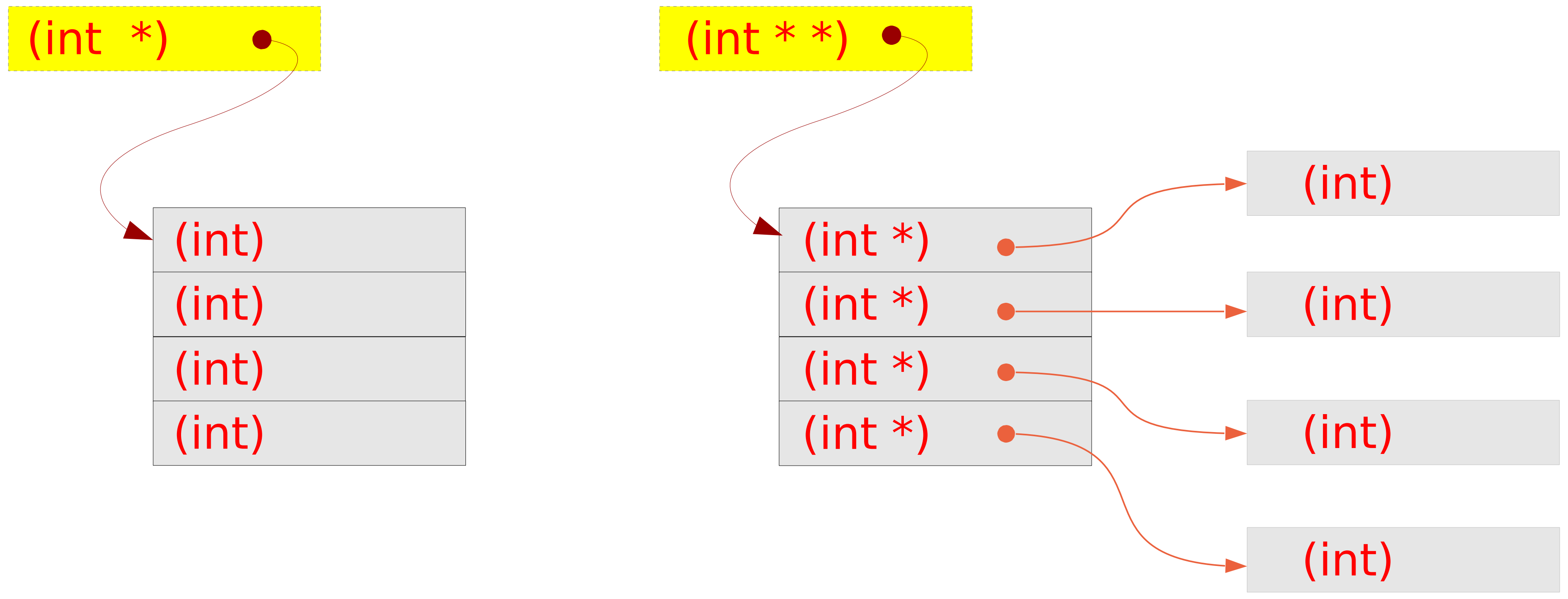
Array of Pointers – variable view



Array of Pointers – type view

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

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



Pointer to Arrays

Pointer to array – variable declarations

```
int   m ;
       ↕
int  *n ;
```

an integer pointer

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

an integer array pointer

```
int func    (int a, int b) ;
     ↕
int (* fp)  (int a, int b) ;
```

a function pointer

Pointer to array – type

int

int *

an integer pointer

int []

int (*) []

an integer array pointer

int (int, int)

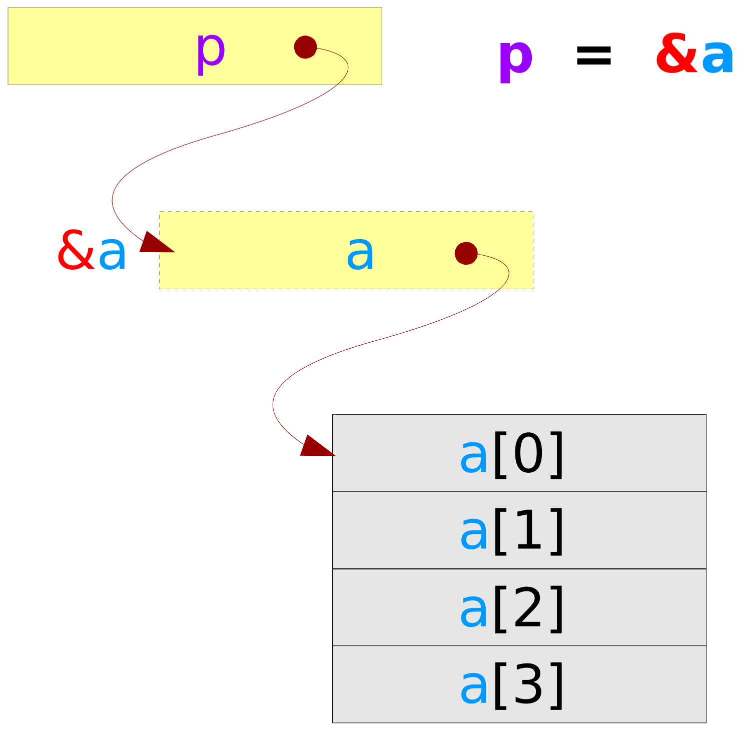
int (*) (int, int)

a function pointer

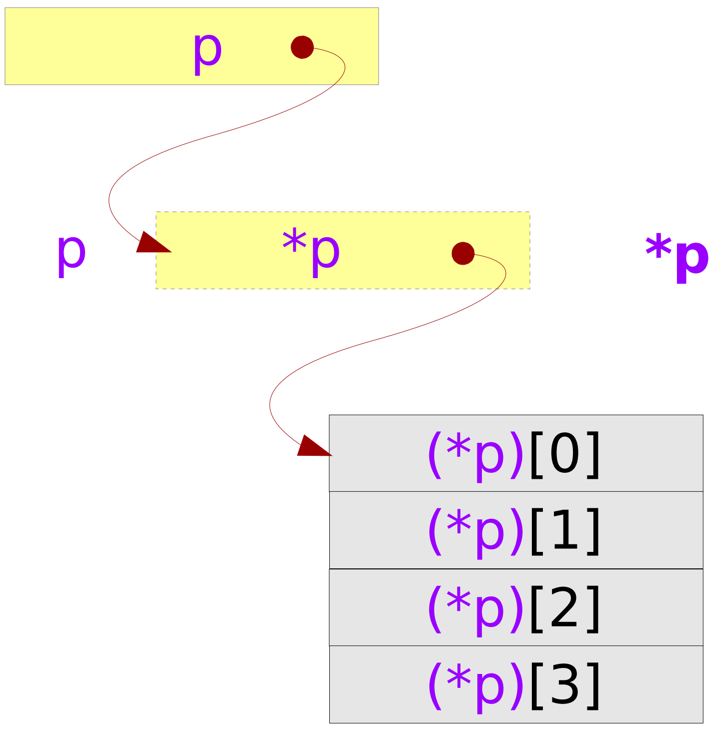
Pointer to array – a variable view

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

```
int        (*p) [4]      = &a;
```



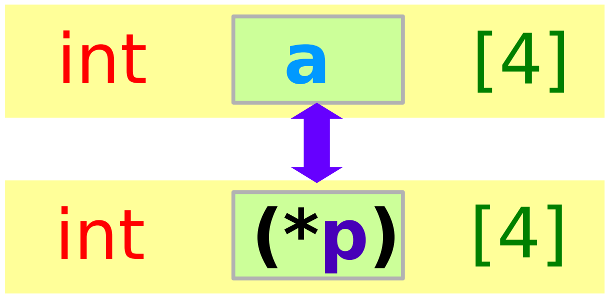
p = &a

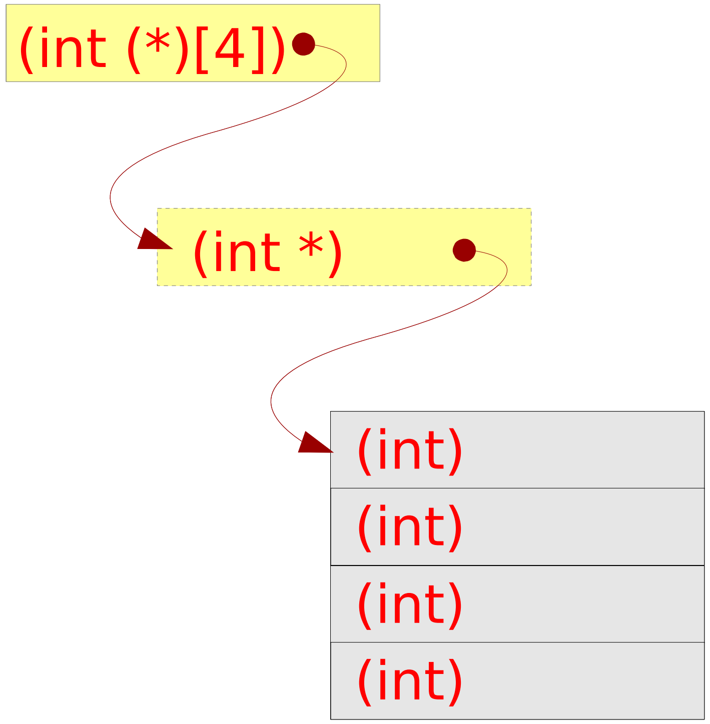


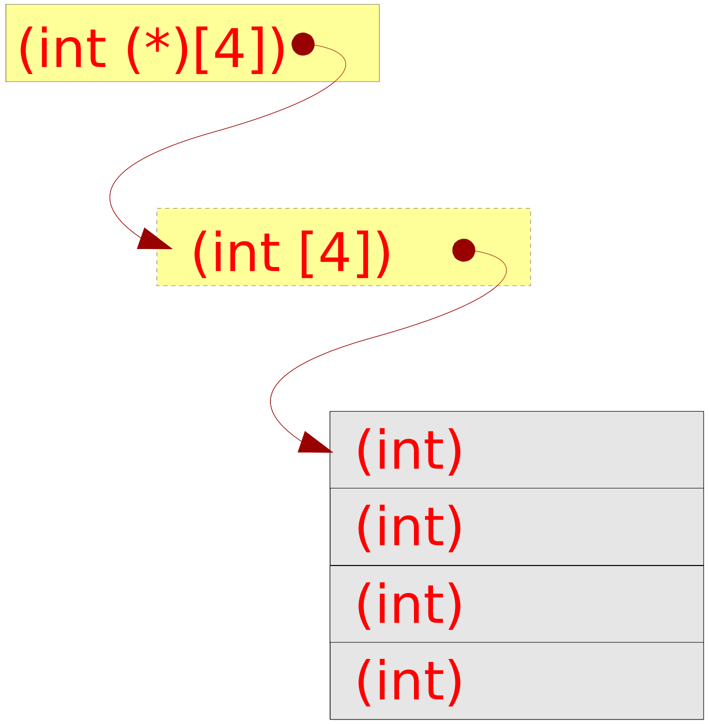
***p ≡ a**

Pointer to array – a variable view

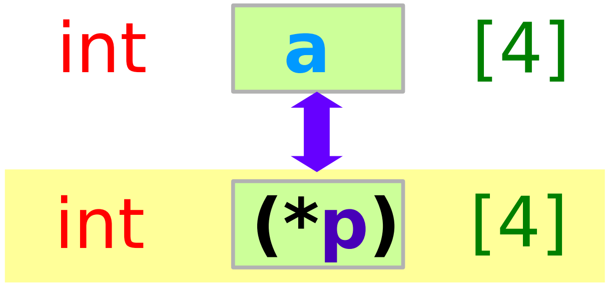
```
int        a [4];
```







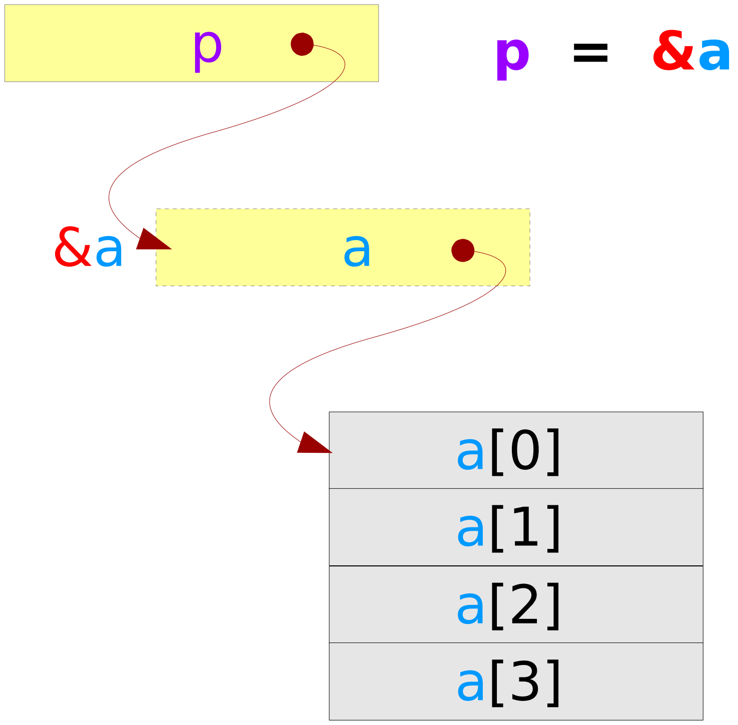
Pointer to array (2)



(*p) = a

&(*p) = &a

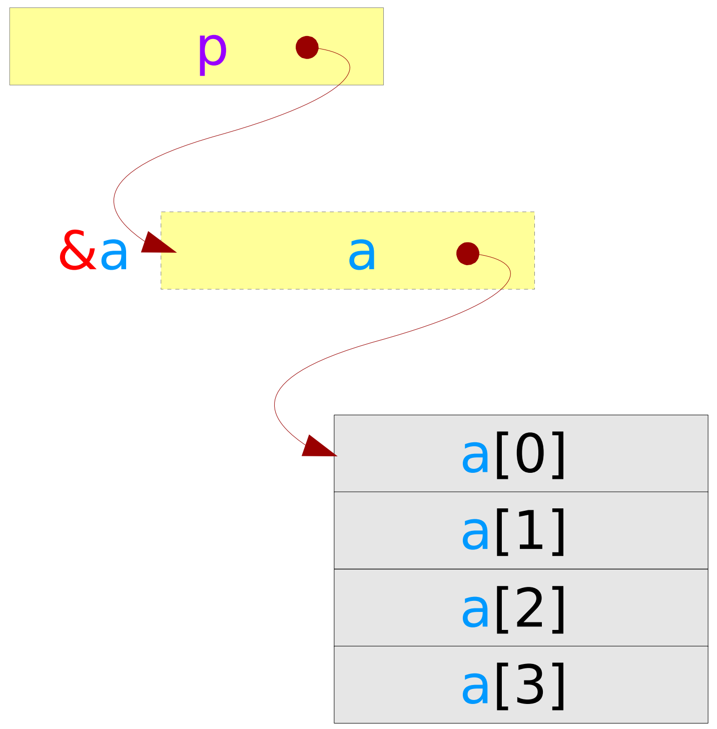
p = &a

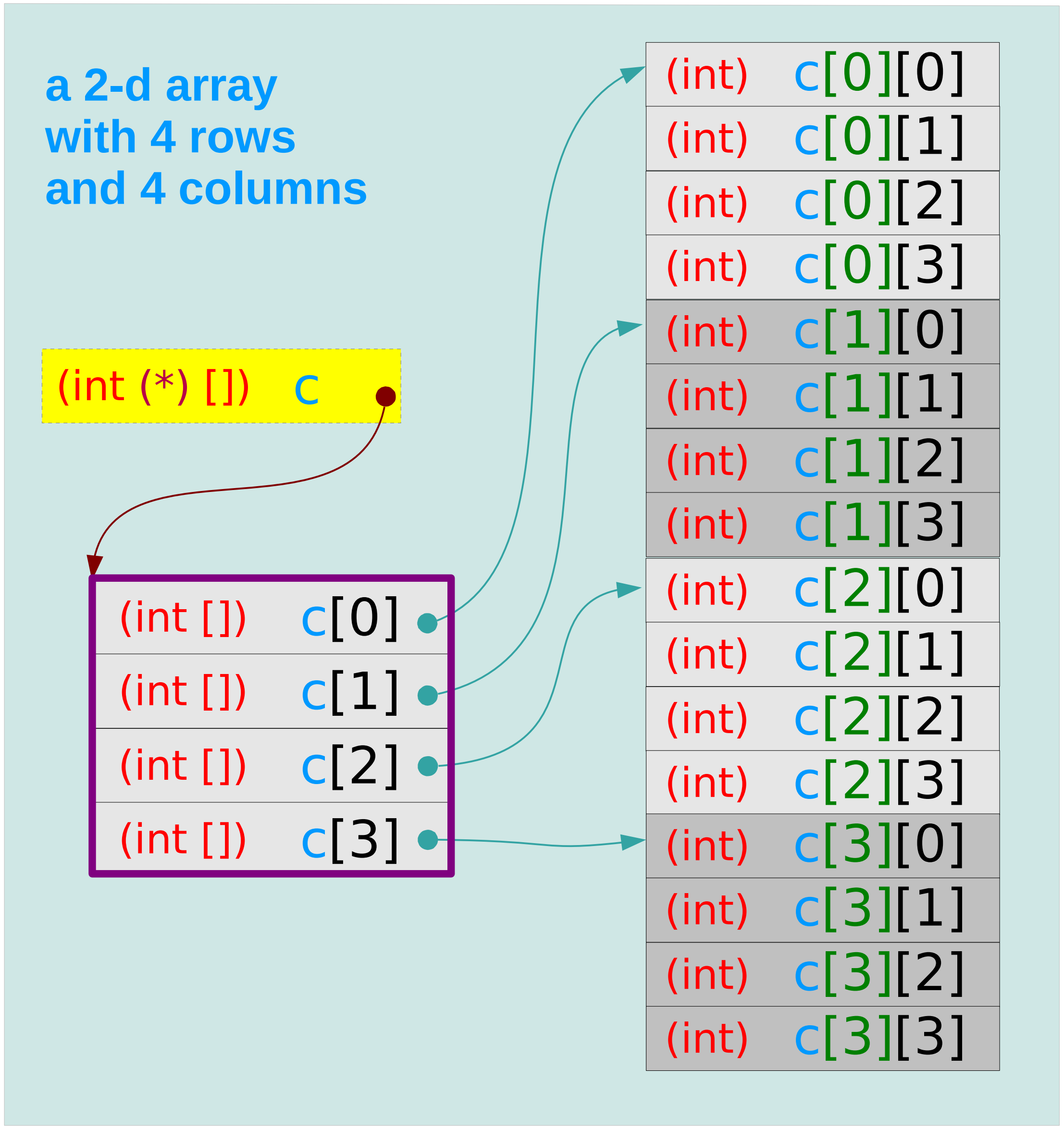


sizeof(p)= 4 bytes

sizeof(*p)= 16 bytes

Pointer to array (3)





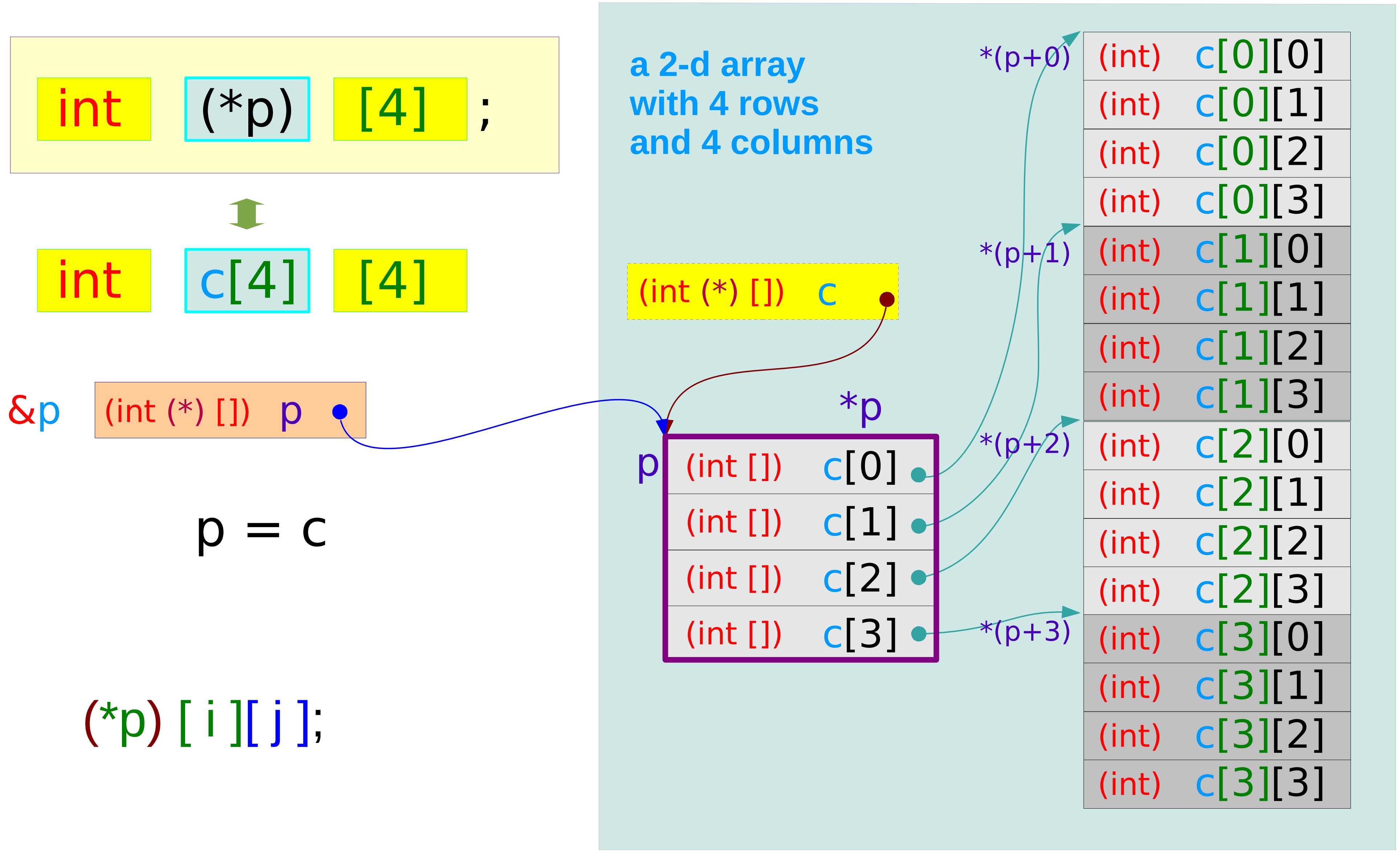
Pointer to array (3)

int (*p) [4] ;

int c[4] [4]

p = c

(*p) [i][j];



Pointer to array (4)

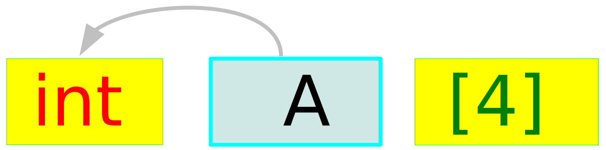
```
int c [4][4];
int (*p) [4];

p = c;

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

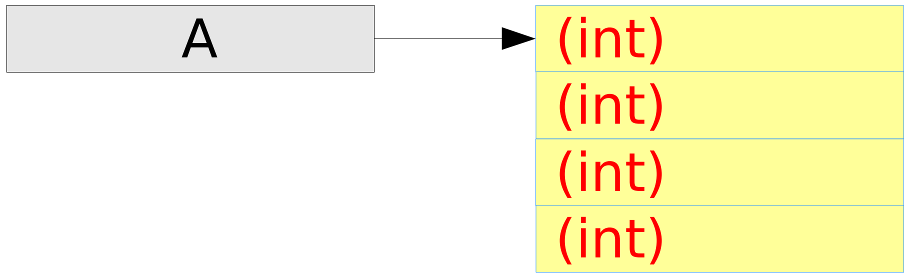
```
void func(int (*x)[4], ... )
{

    x[r][c] =

}
```

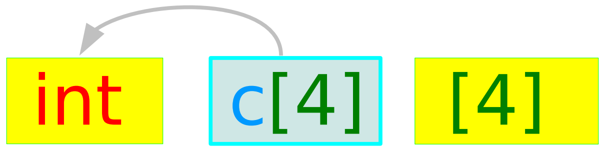
```
void func(int x[ ][4], ... )
{

    x[r][c] =

}
```

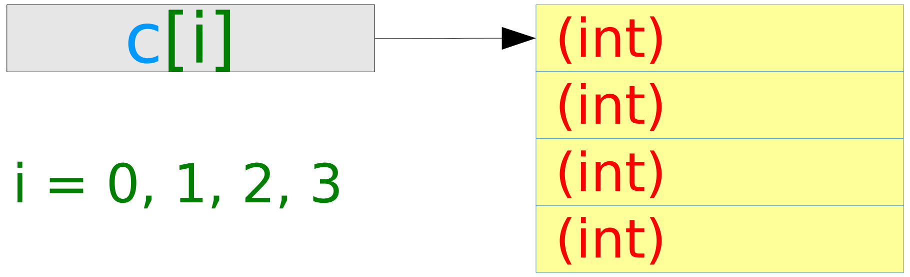
2-d Arrays

Addresses of 4 element integer arrays





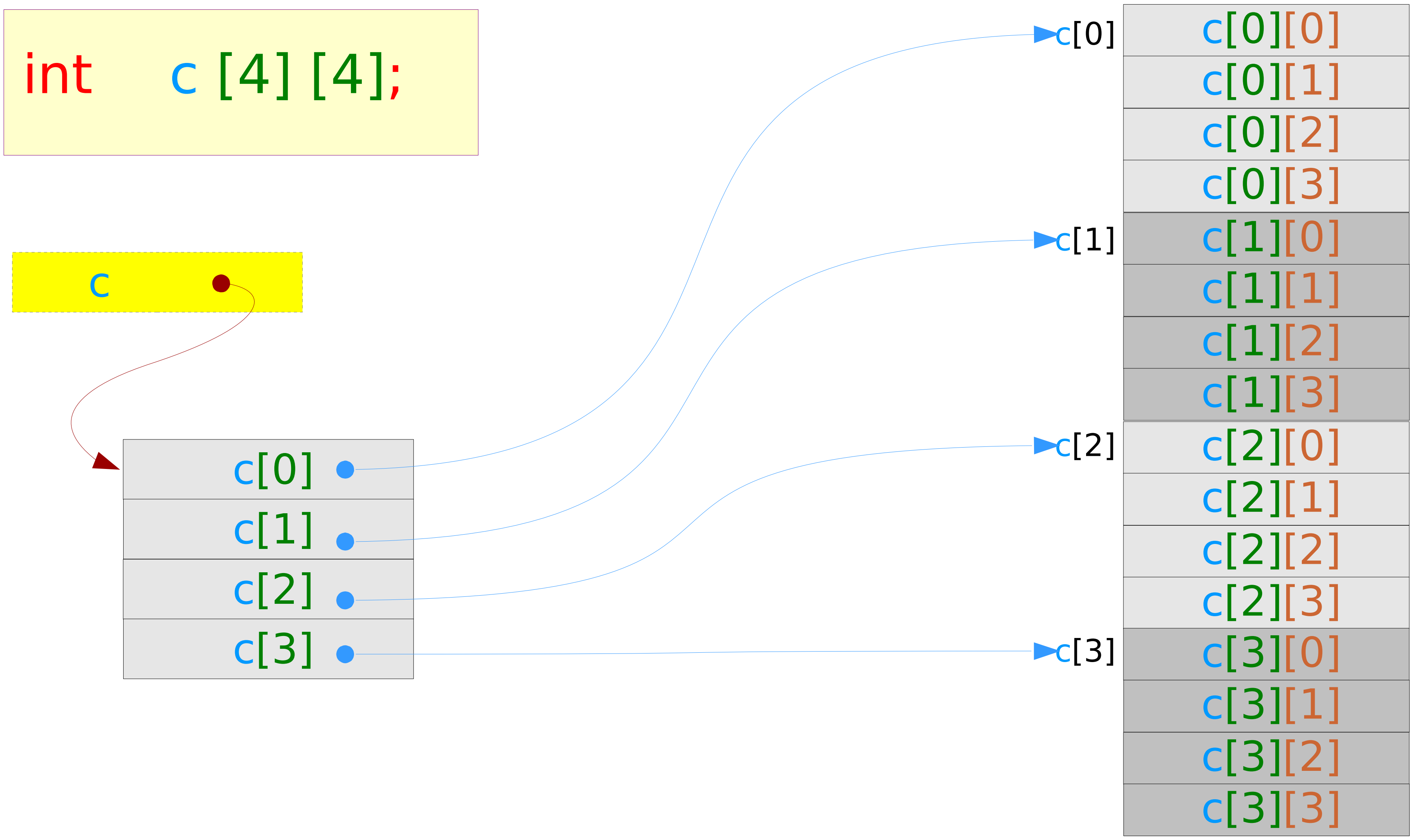




```
int    c [4] [4];
```

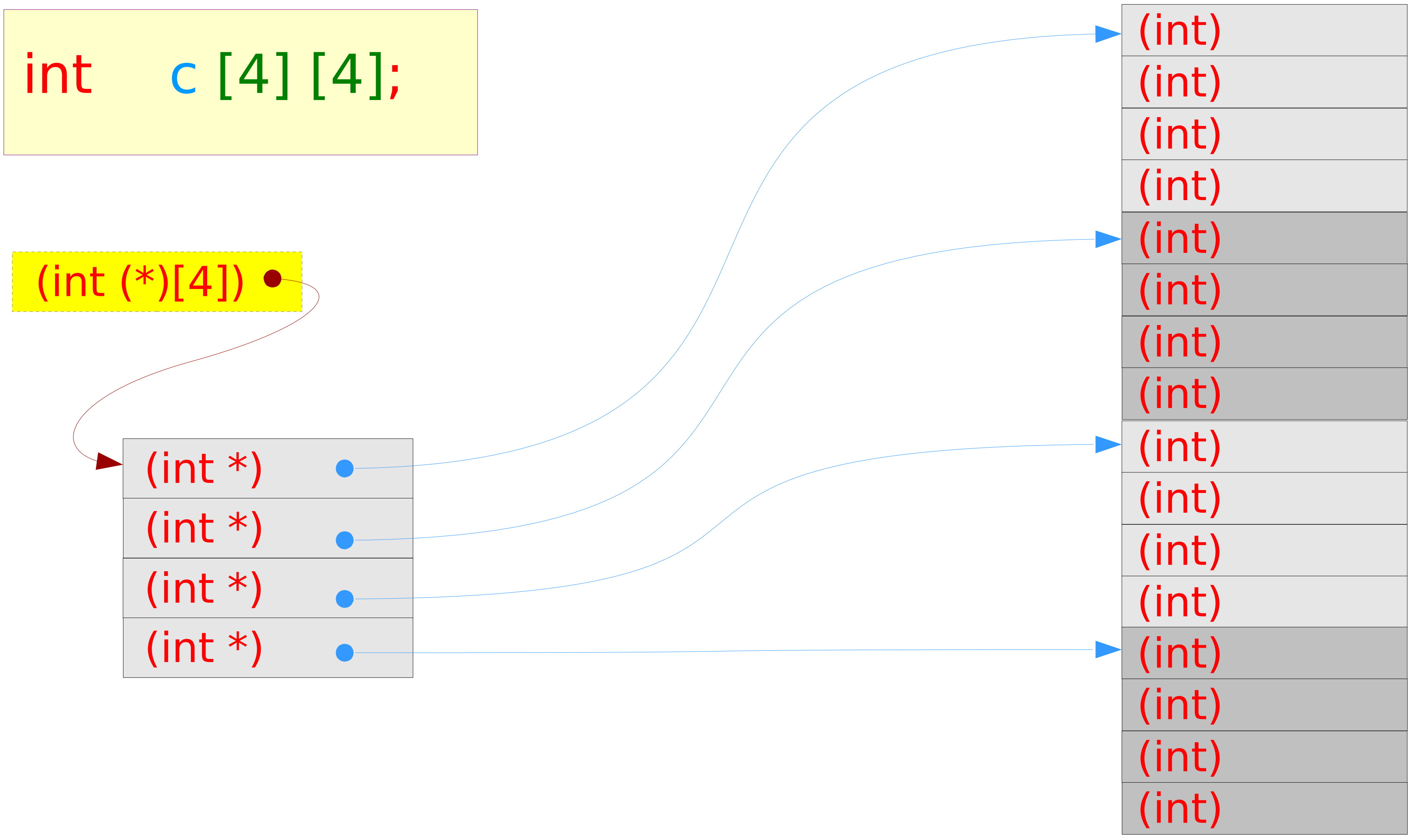
A 2-D Array – a variable view

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



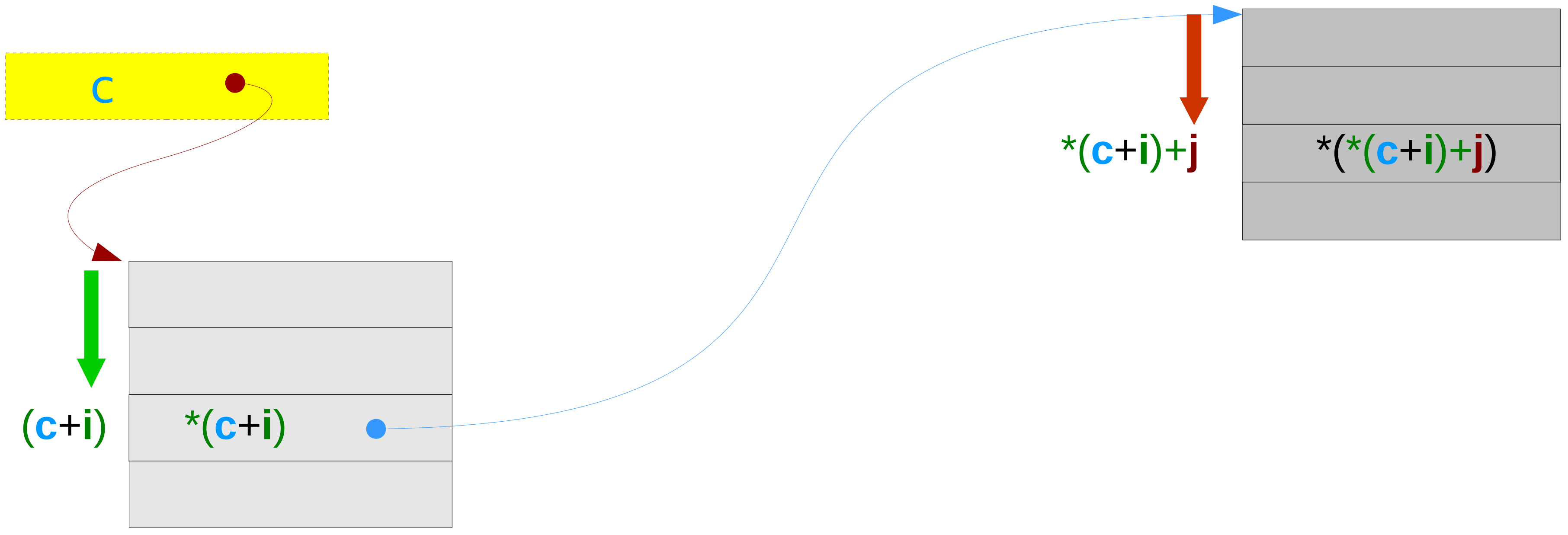
A 2-D Array – a type view

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



A 2-D Array – an index view

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



A 2-D Array via a double indirection

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

int c[4] [4]

$$\begin{tikzcd} \texttt{(c [i])[j]} \arrow[r] & \texttt{(*(c+i))[j]} \arrow[r] & \boxed{\texttt{*(*(c+i)+j)}} \end{tikzcd}$$

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

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

A 2-D Array via an array pointer

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

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

(c [i])[j] ➡ (*(c+i))[j] ➡ *(*(c+i)+j)

p = c

(p [i])[j] ➡ (*(p+i))[j] ➡ *(*(p+i)+j)

A 2-D Array via a double pointer

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

```
int    **p, *q[4];
```

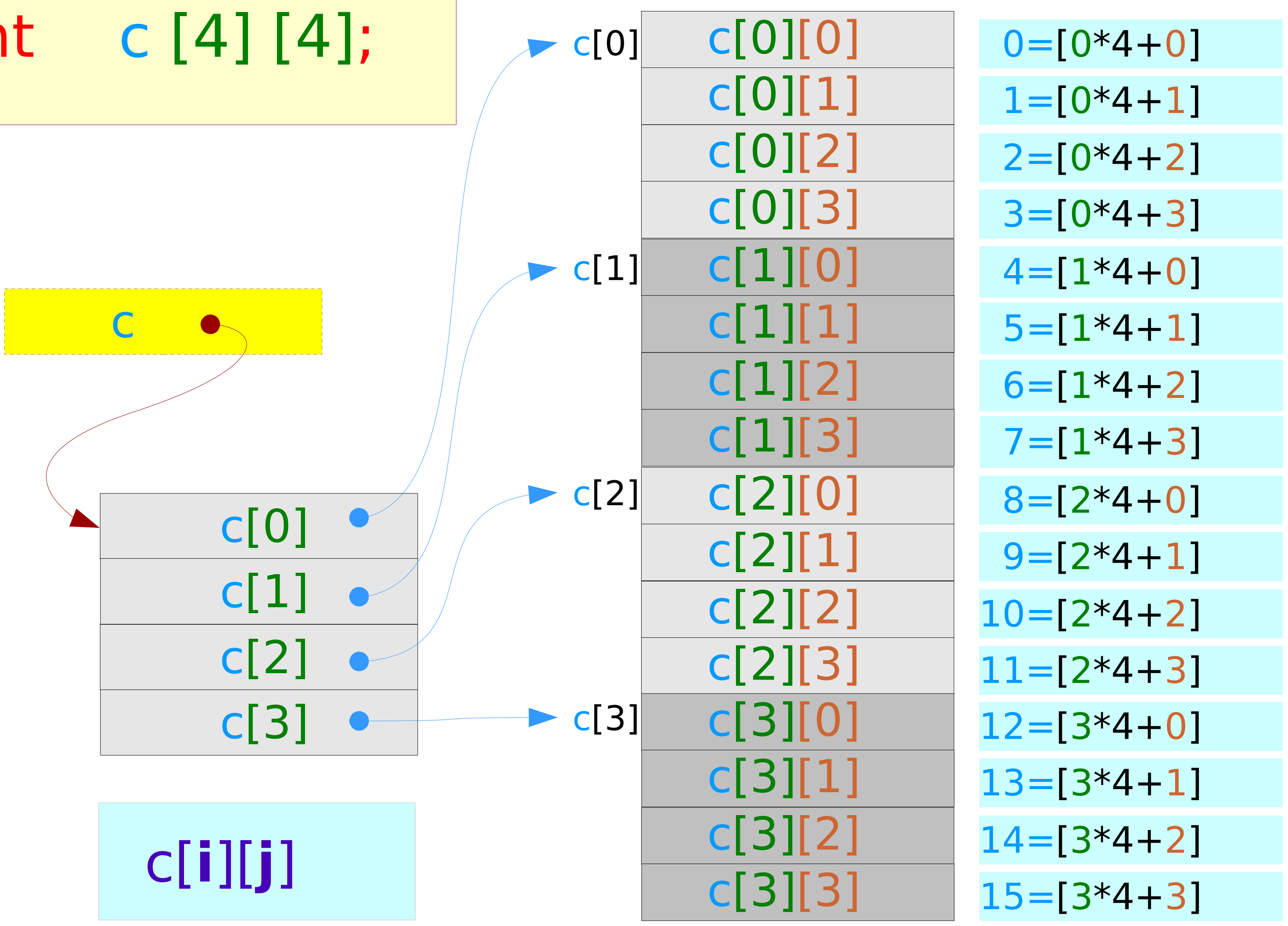
$$\begin{tikzcd}
\texttt{(c [i])[j]} \arrow[r] & \texttt{(*(c+i))[j]} \arrow[r] & \boxed{\texttt{*(*(c+i)+j)}}
\end{tikzcd}$$

```
p = q;    q[0]=c[0], q[1]=c[1], q[2]=c[2], q[3]=c[3];
```

$$\begin{tikzcd}
\texttt{(p [i])[j]} \arrow[r] & \texttt{(*(p+i))[j]} \arrow[r] & \boxed{\texttt{*(*(p+i)+j)}}
\end{tikzcd}$$

2-D array as a 1-D array

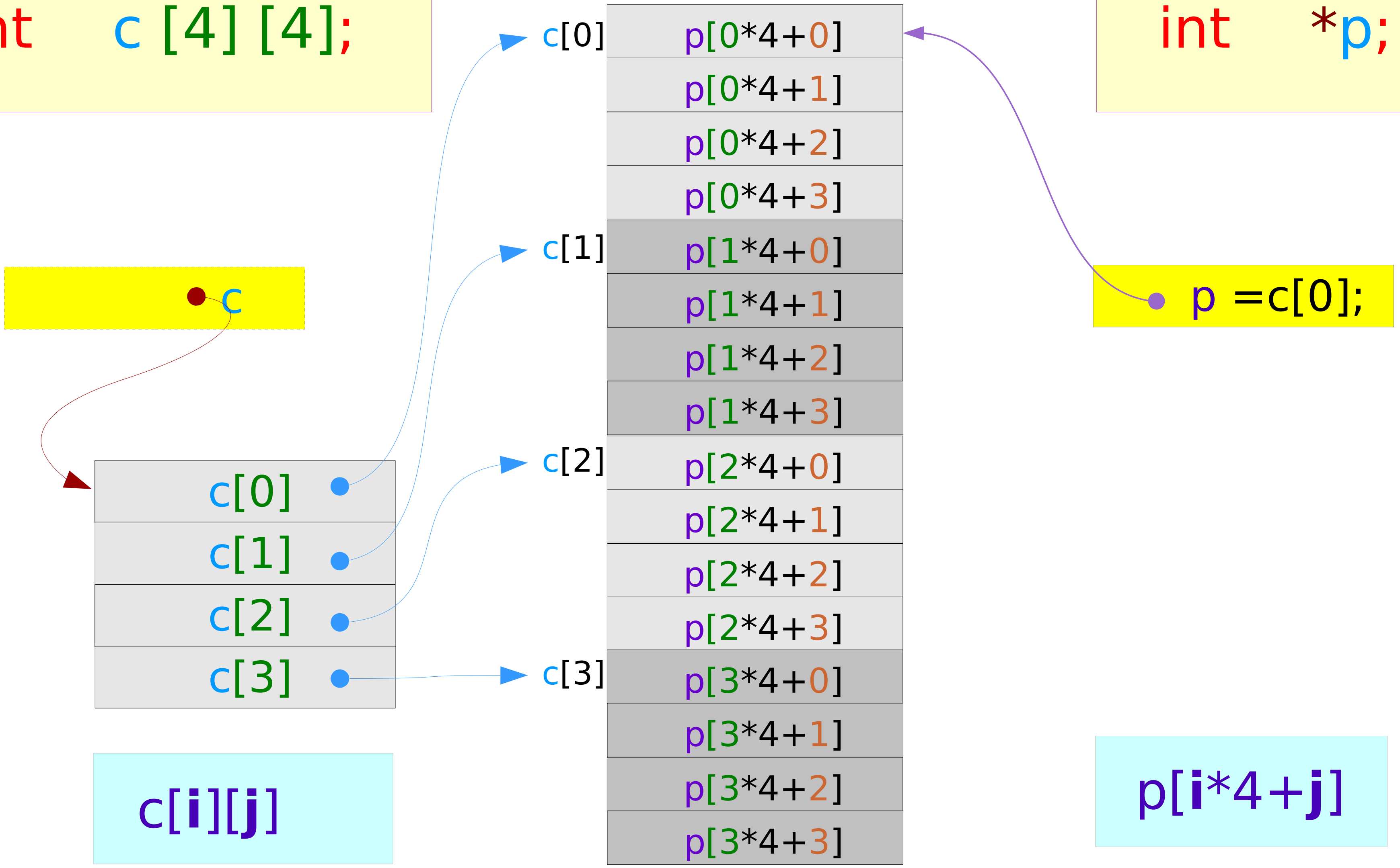
```
int    c [4] [4];
```



Accessing a 2-D array via a single pointer

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

```
int    *p;
```



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

p[**i***4+**j**]

2-D array index vs 1-D array index

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

```
int    *p=c[0];
```

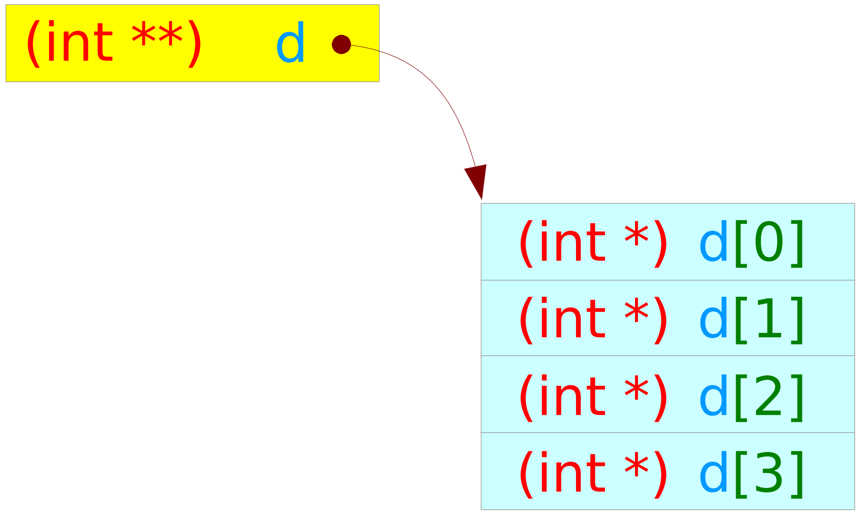
```
c[i][j]
```

```
p[i*4+j]
```

c[0]	c[0][0]	p[0*4+0]
	c[0][1]	p[0*4+1]
	c[0][2]	p[0*4+2]
	c[0][3]	p[0*4+3]
c[1]	c[1][0]	p[1*4+0]
	c[1][1]	p[1*4+1]
	c[1][2]	p[1*4+2]
	c[1][3]	p[1*4+3]
c[2]	c[2][0]	p[2*4+0]
	c[2][1]	p[2*4+1]
	c[2][2]	p[2*4+2]
	c[2][3]	p[2*4+3]
c[3]	c[3][0]	p[3*4+0]
	c[3][1]	p[3*4+1]
	c[3][2]	p[3*4+2]
	c[3][3]	p[3*4+3]

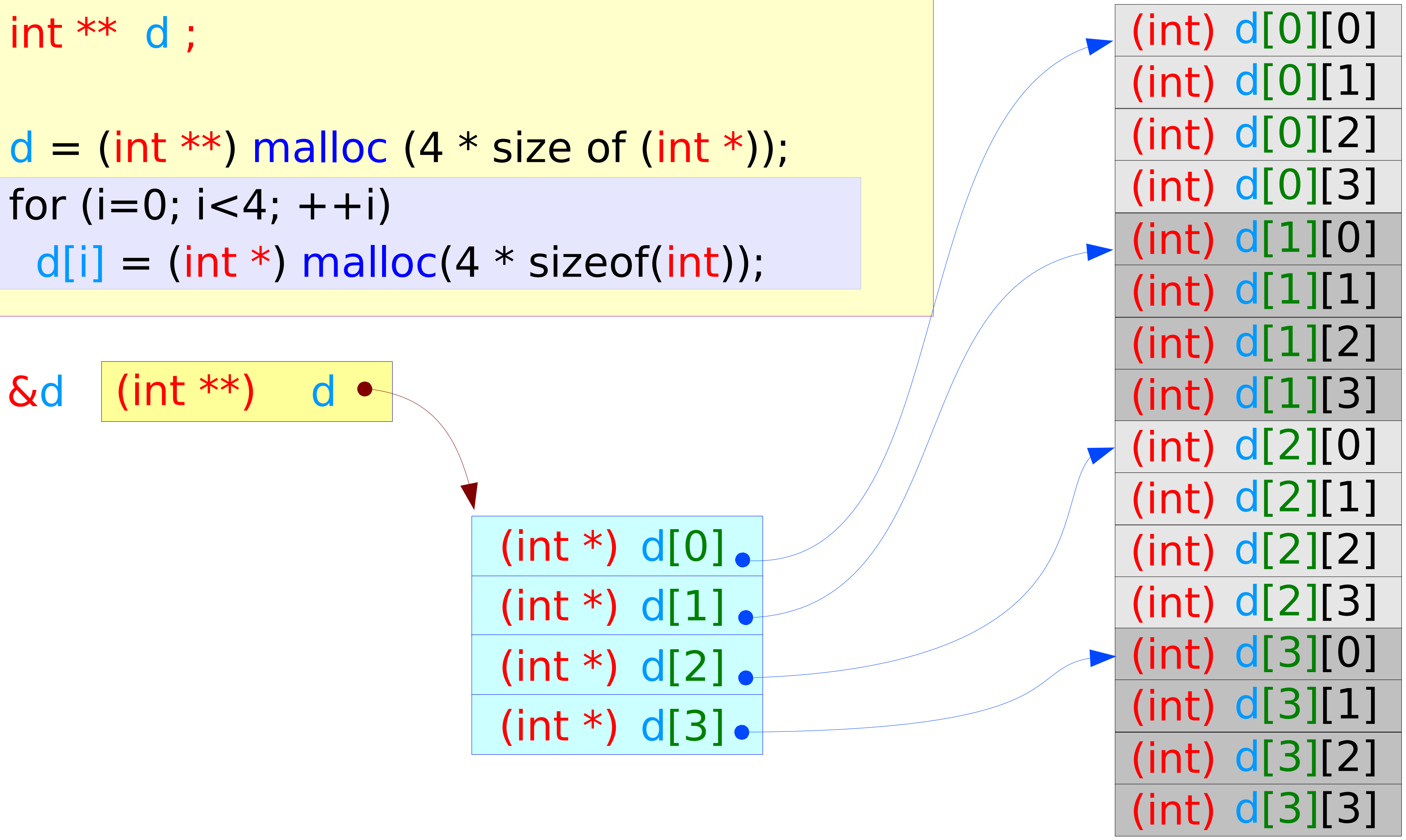
2-D Array Dynamic Memory Allocation (1)

```
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));
```



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