Applications of Array Pointers (1A)

Young Won Lim
12/4/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.
This document was produced by using LibreOffice.

Pointer to Multi-dimensional Arrays

Integer pointer types

(int **)

a pointer to a **integer pointer**
size = 8 bytes

(int *)

a pointer to an **int**
size = 8 bytes

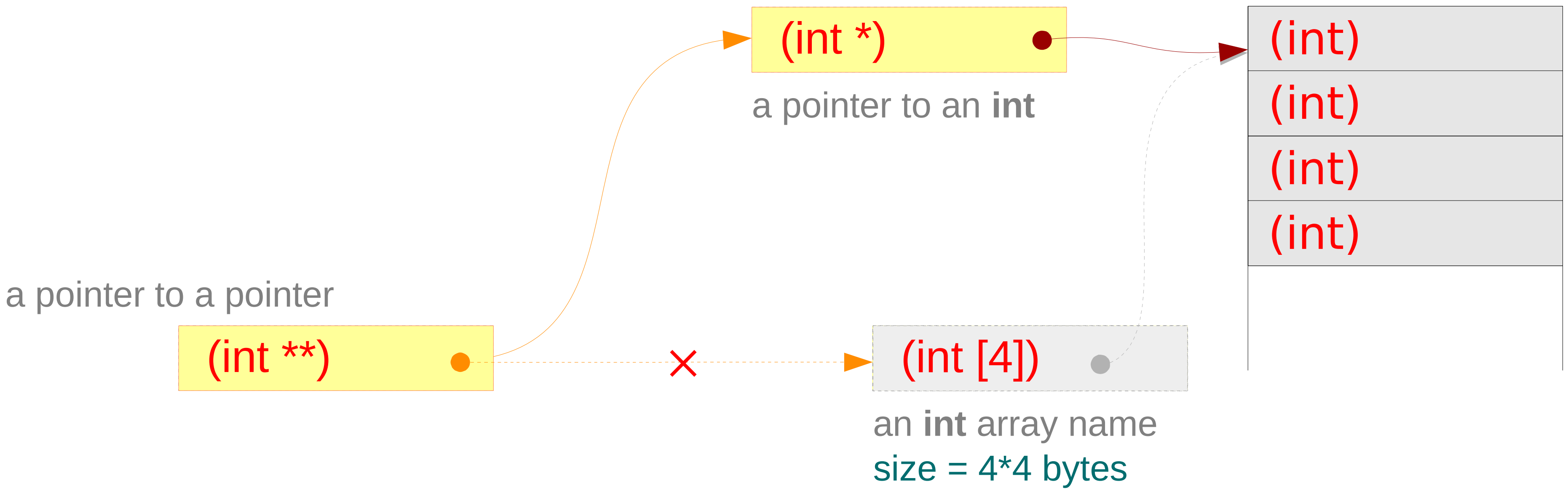
(int (*)[4])

a pointer to a **1-d array**
size = 8 bytes

(int [4])

an **int array** name
size = 4*4 bytes

Integer pointer type : (int **)



(int *)

a pointer to an **int**

(int)

(int)

(int)

(int)

a pointer to a pointer

(int **)

×

(int [4])

an **int** array name

size = 4*4 bytes

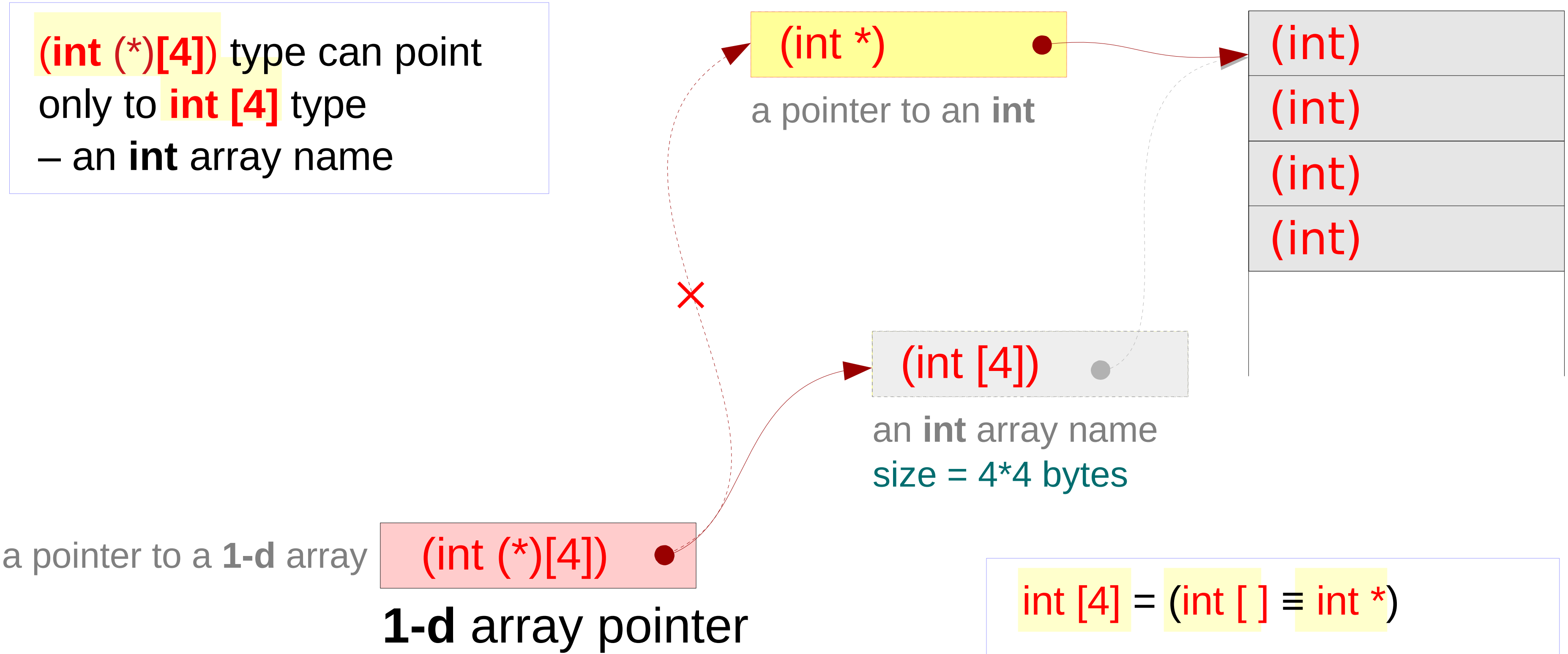
(int **) type can point only to **(int *)** type – an **int** array name

int [4] = (int [ ] ≡ int *)

each of these types points to an **int** type data

Integer pointer type : (int (*)[4])

(int (*)[4]) type can point only to **int [4]** type – an **int** array name



int [4] = (int [ ] ≡ int *)

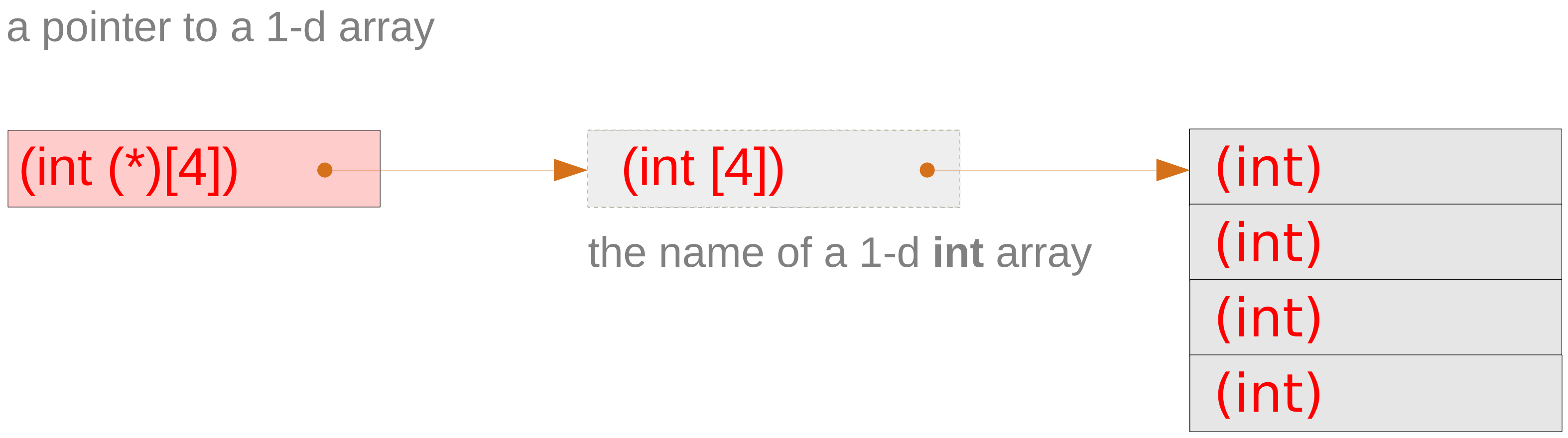
each of these types points to an **int** type data

Integer pointer types

```
#include <stdio.h>

void func(int d[])
{

}

int main(void) {
  int a[4];
  int *b;
  int **c;

  int (*p)[4];

  func(a);


}
```

```
sizeof(a)=16        // array size
sizeof(*a)=4        // int size

sizeof(b)=8         // pointer size
sizeof(*b)=4        // int size

sizeof(c)=8         // pointer size
sizeof(*c)=8        // pointer size

sizeof(d)=8         // pointer size
sizeof(*d)=4        // int size

sizeof(p)=8         // pointer size
sizeof(*p)=16       // array size
```

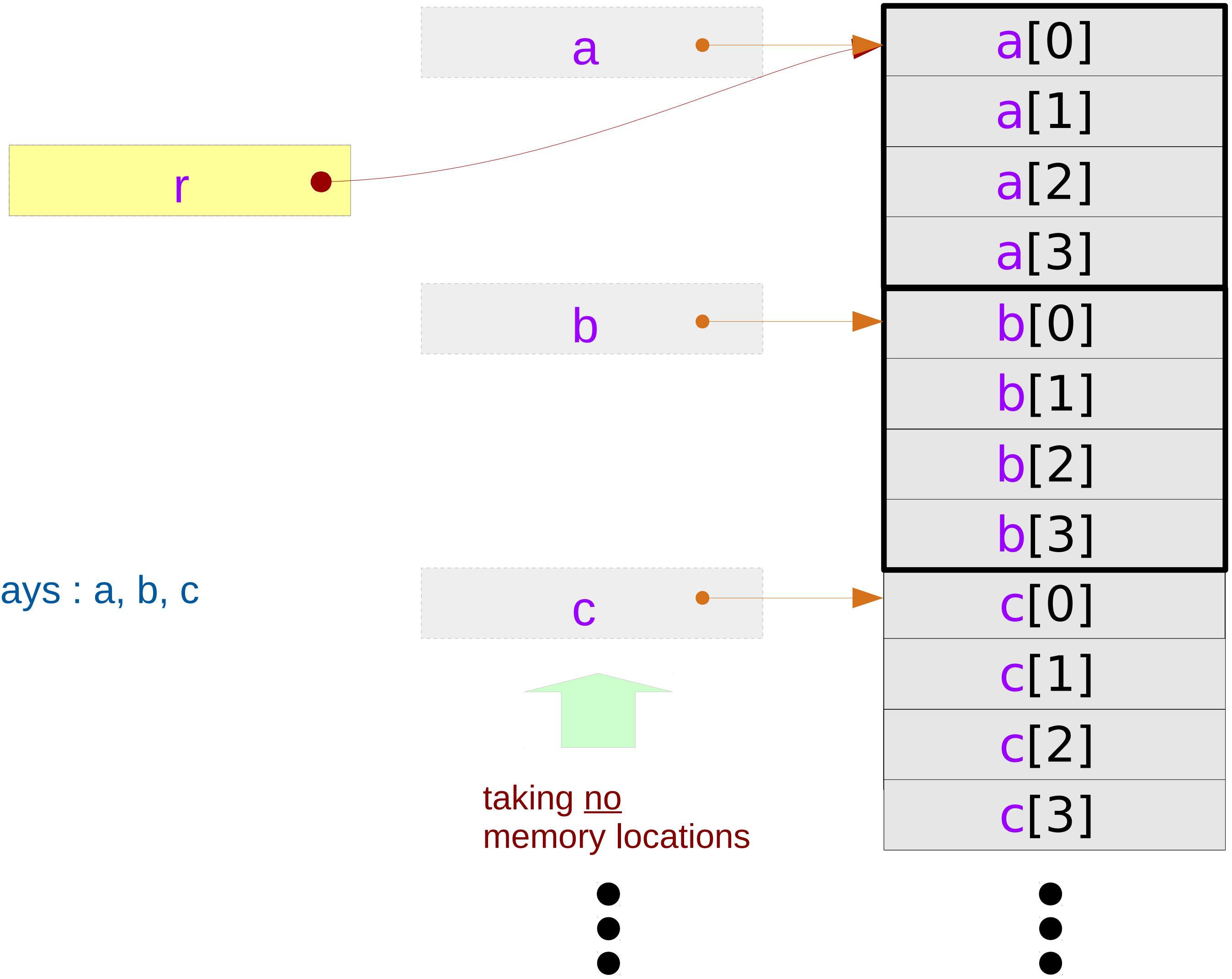
a **1-d** array pointer – a type view



Contiguous **1-d** arrays **a**, **b**, **c**

```
int a[4];   int  (*r);
int b[4];
int c[4];
```

assume contiguous 1-d arrays : a, b, c



taking <u>no</u>
memory locations

Assigning series of array pointers

```
int a[4];        int (*p1)[4];          int  (*r);        int (*q)[4][4];
int b[4];        int (*p2)[4];
int c[4];        int (*p3)[4];
```

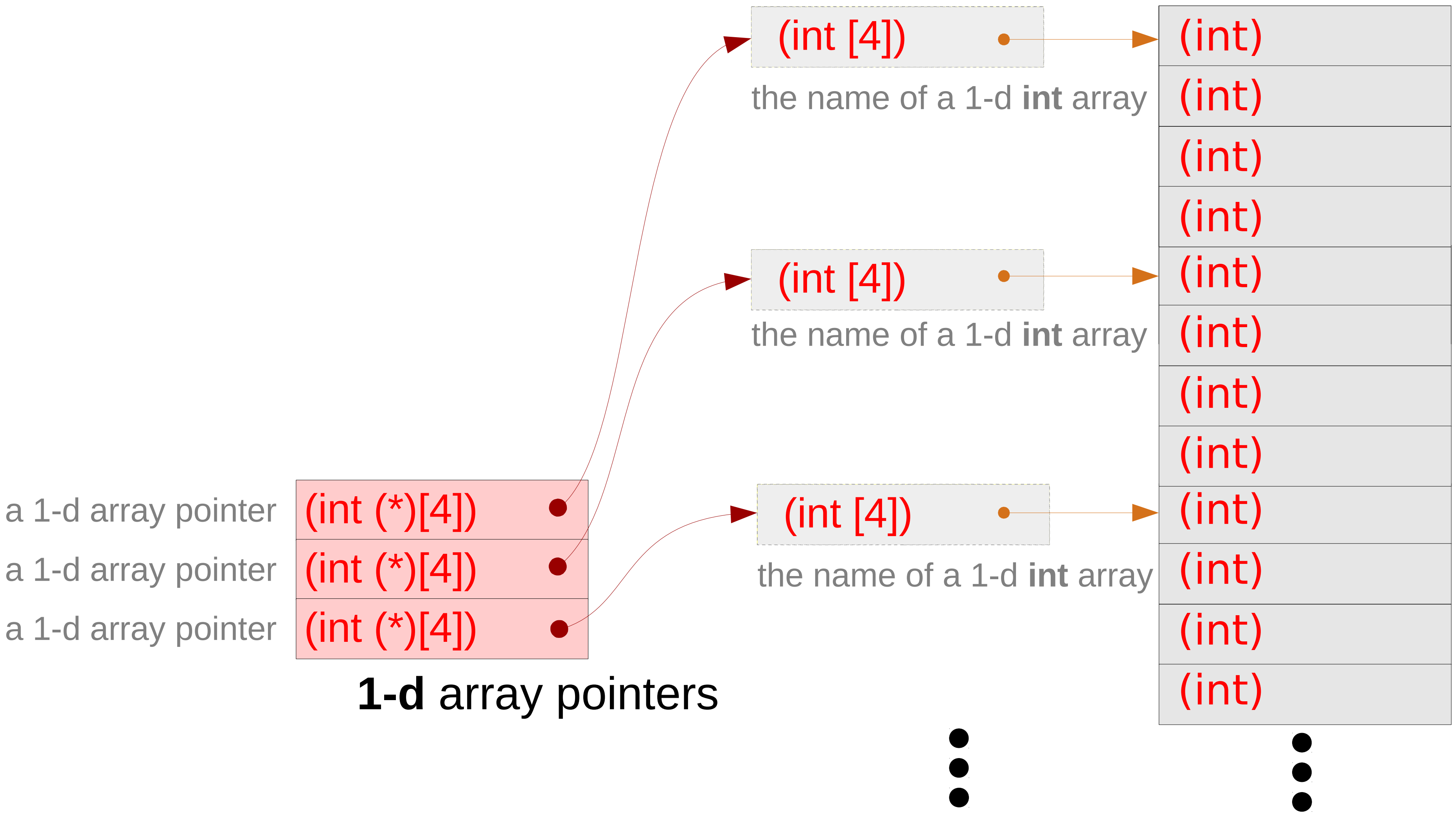
assignment

p1 = &a
p2 = &b
p3 = &c

equivalence

(*p1) ≡ p1[0] ≡ a
(*p2) ≡ p2[0] ≡ b
(*p3) ≡ p3[0] ≡ c

Series of array pointers – a type view

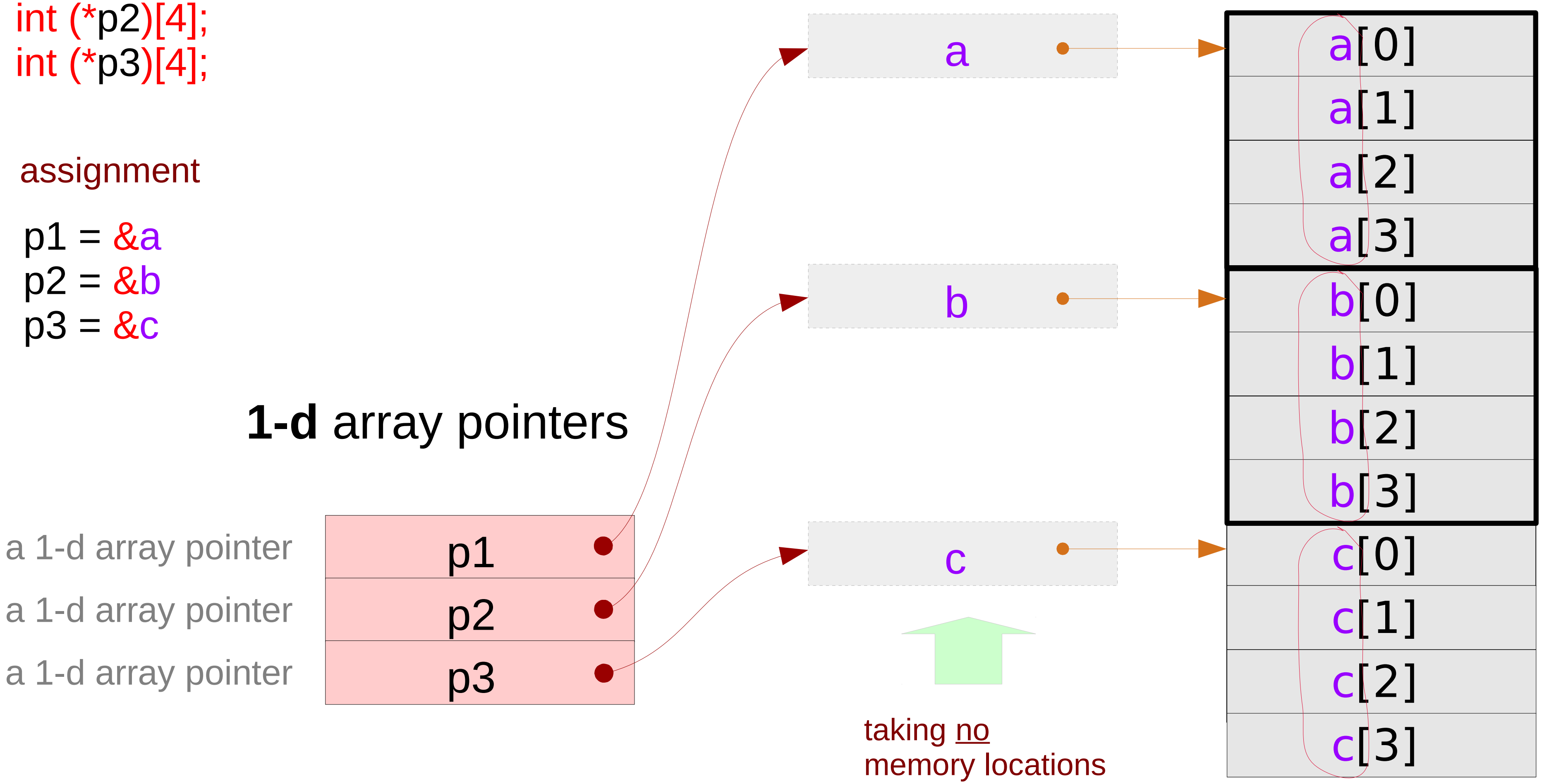


Series of array pointers – array pointers **p1**, **p2**, **p3**

```
int (*p1)[4];
int (*p2)[4];
int (*p3)[4];
```

assignment

p1 = &a
p2 = &b
p3 = &c



assume that array p1, p2, and p3 are contiguous

assume that array a, b, and c are contiguous

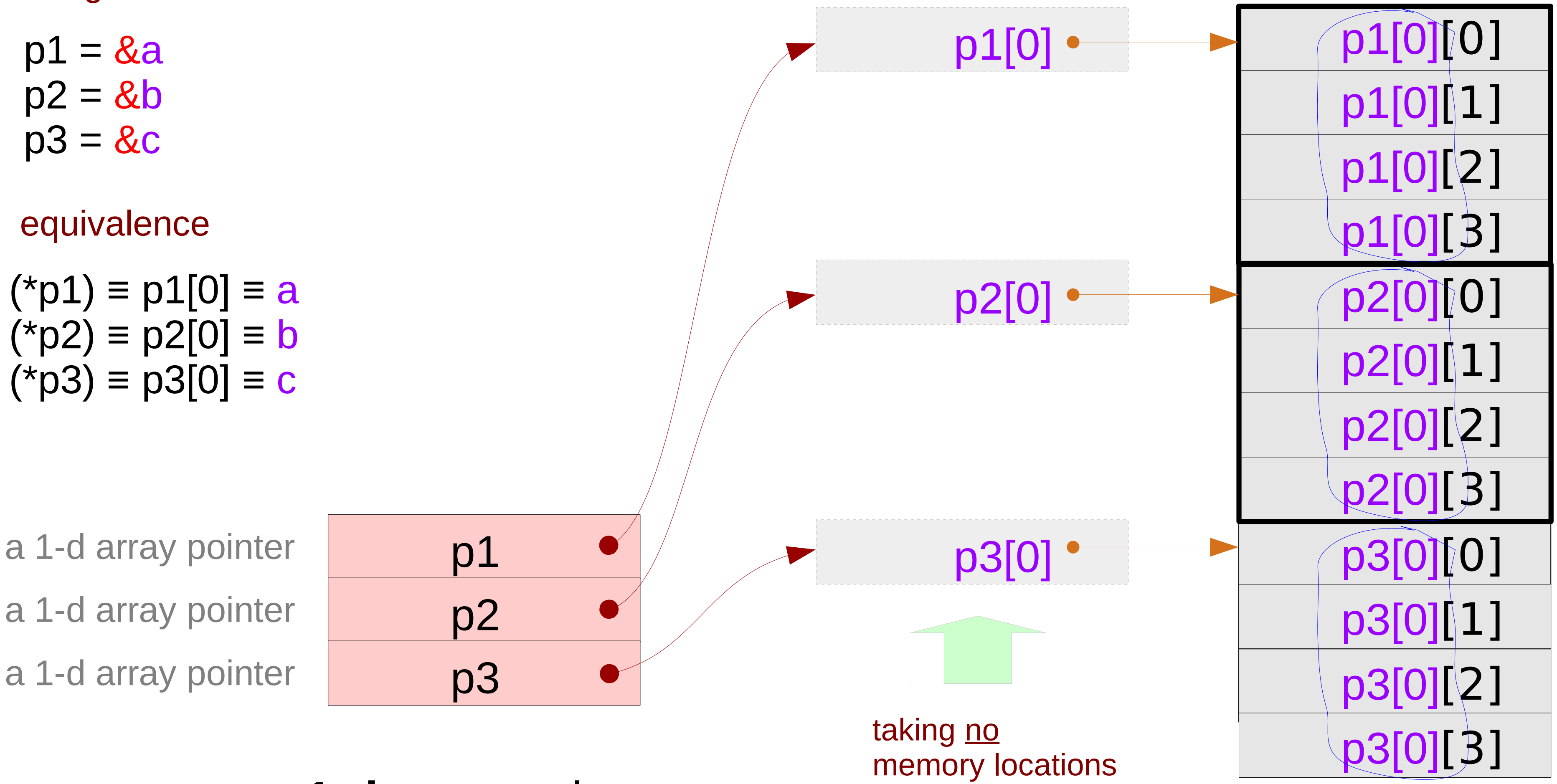
Series of array pointers – **1-d** arrays via **p1**, **p2**, **p3**

assignment

p1 = &a
p2 = &b
p3 = &c

equivalence

(*p1) ≡ p1[0] ≡ a
(*p2) ≡ p2[0] ≡ b
(*p3) ≡ p3[0] ≡ c



1-d array pointers

assume that array a, b, and c are contiguous

Series of array pointers – use **p** instead of **p1**, **p2**, **p3**

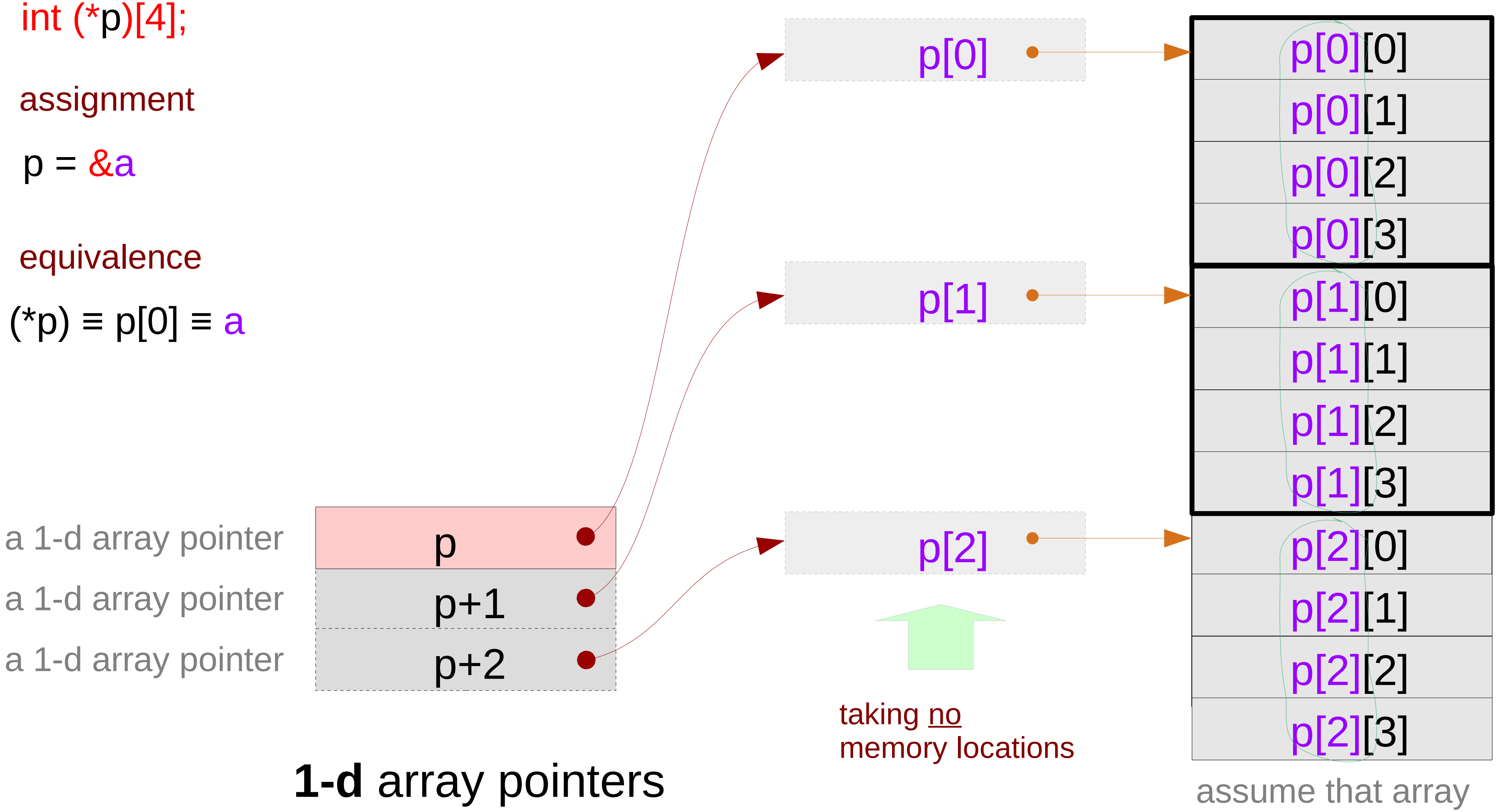
int (*p)[4];

assignment

p = &a

equivalence

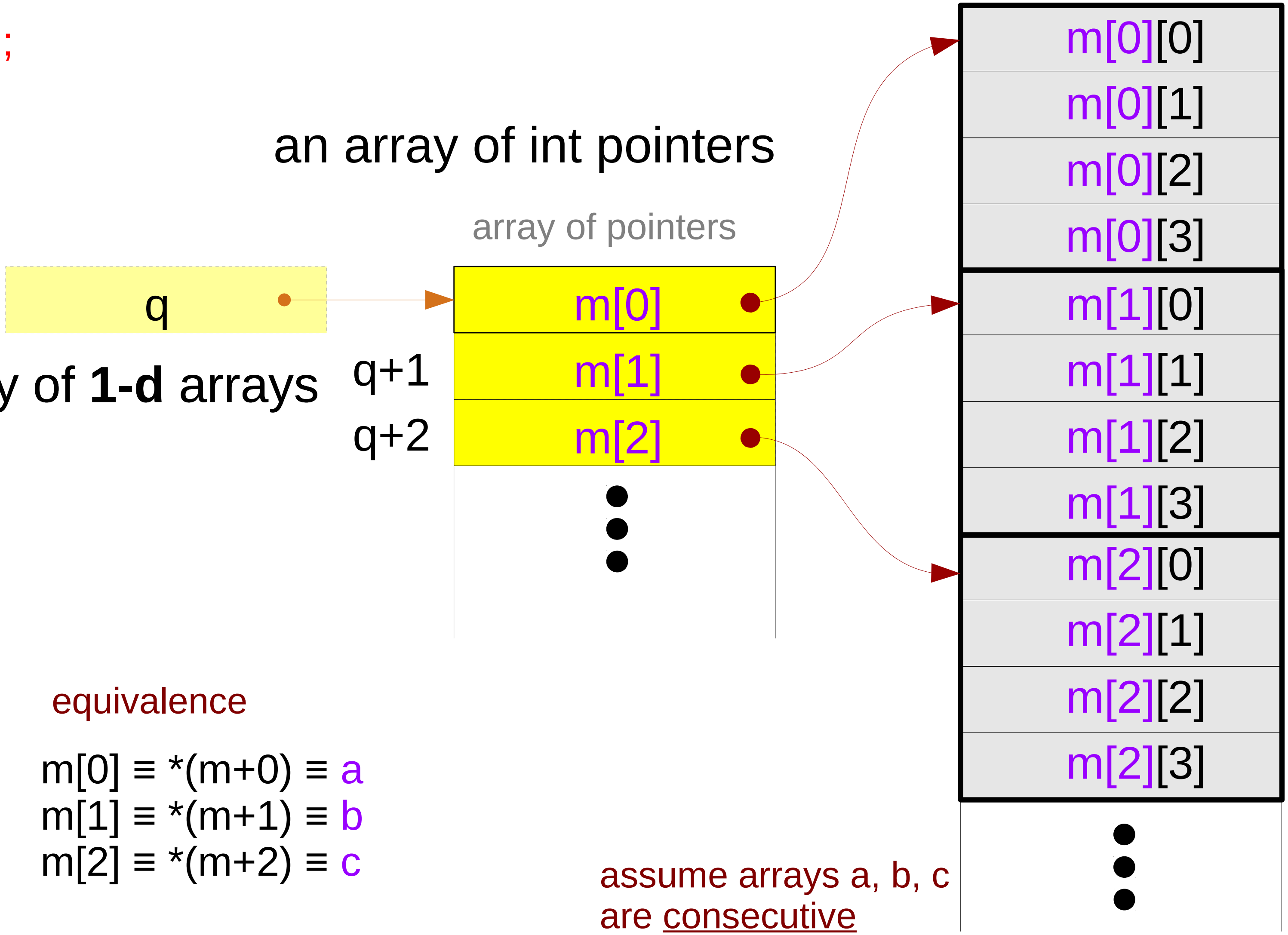
(*p) ≡ p[0] ≡ a



assume that array a, b, and c are contiguous

Series of array pointers – use **m** and **q**

```
int (*q)[4][4]  = m;

int *m[4];
```



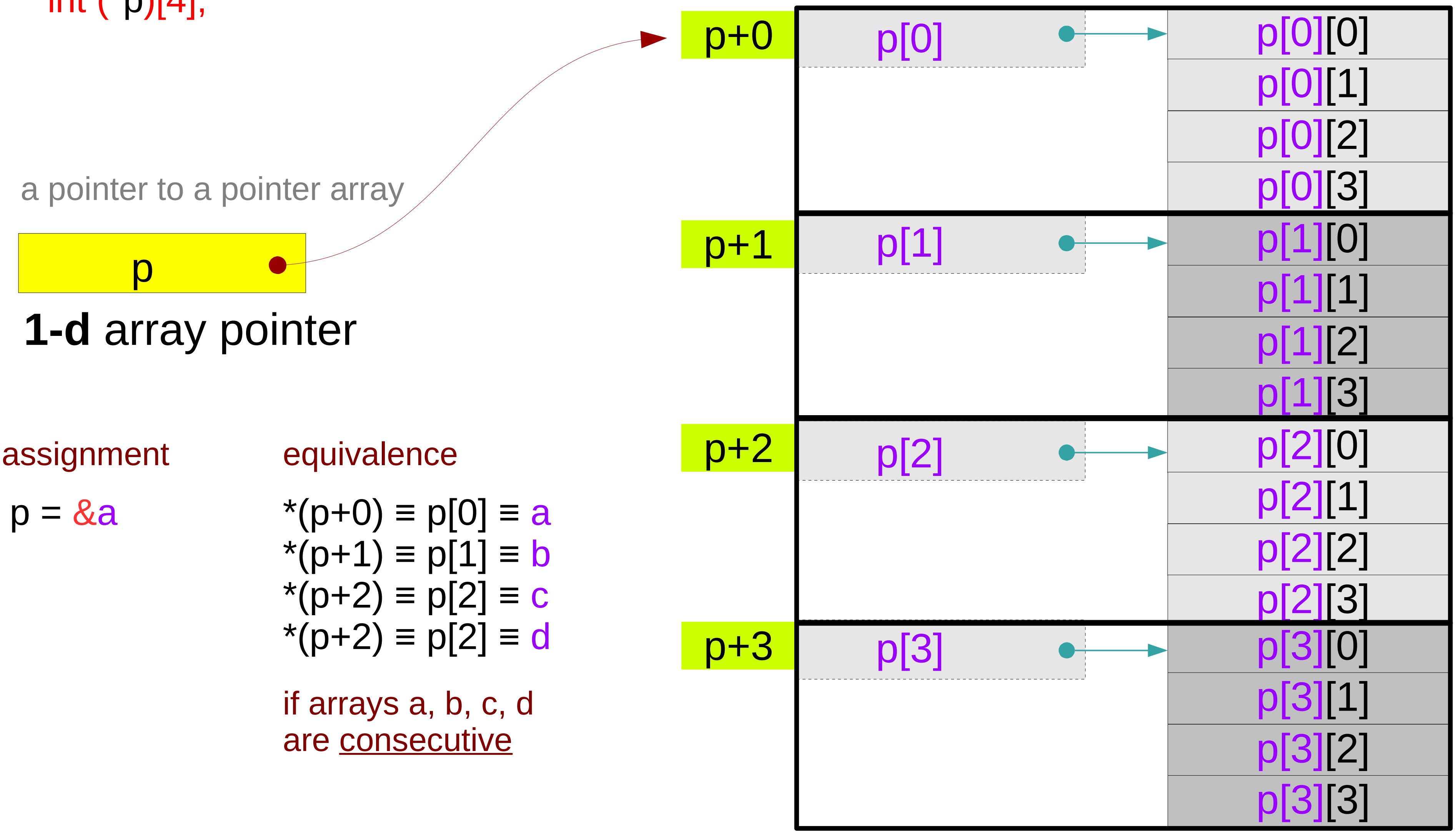
an array of **1-d** arrays

assignment	equivalence
m[0] = a	m[0] ≡ *(m+0) ≡ a
m[1] = b	m[1] ≡ *(m+1) ≡ b
m[2] = c	m[2] ≡ *(m+2) ≡ c

assume arrays a, b, c
are consecutive

1-d array pointer to consecutive **1-d** arrays

int (*p)[4];



1-d array pointer

assignment	equivalence
p = &a	*(p+0) ≡ p[0] ≡ a *(p+1) ≡ p[1] ≡ b *(p+2) ≡ p[2] ≡ c *(p+2) ≡ p[2] ≡ d

if arrays a, b, c, d
are consecutive

A **2-d** array and its sub-arrays – array name

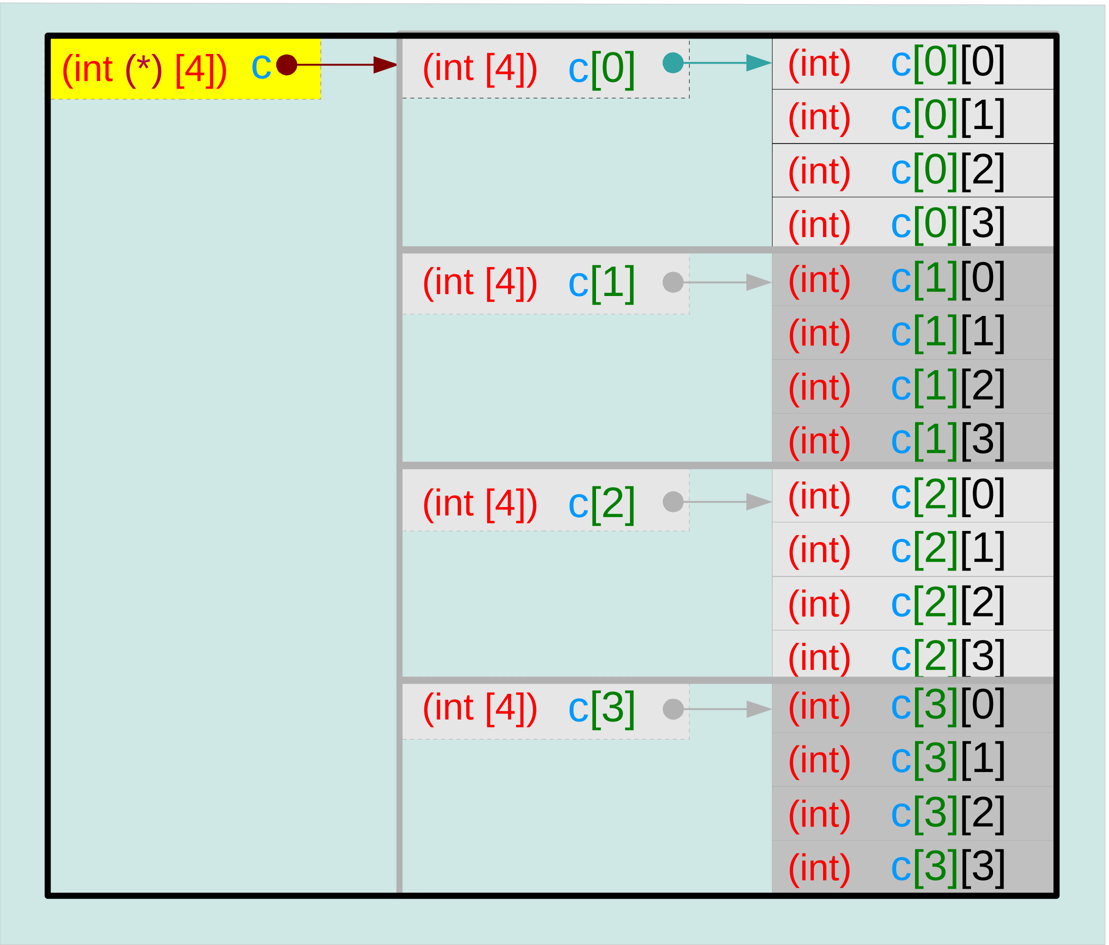
int c[4][4];

c :

- the **2-d** array name
- the **2-d** array starting address
- the **1-d** array pointer

  points to its 1st **1-d** sub-array

compilers do not allocate
c's memory location



A **2-d** array and its sub-arrays – subarray names

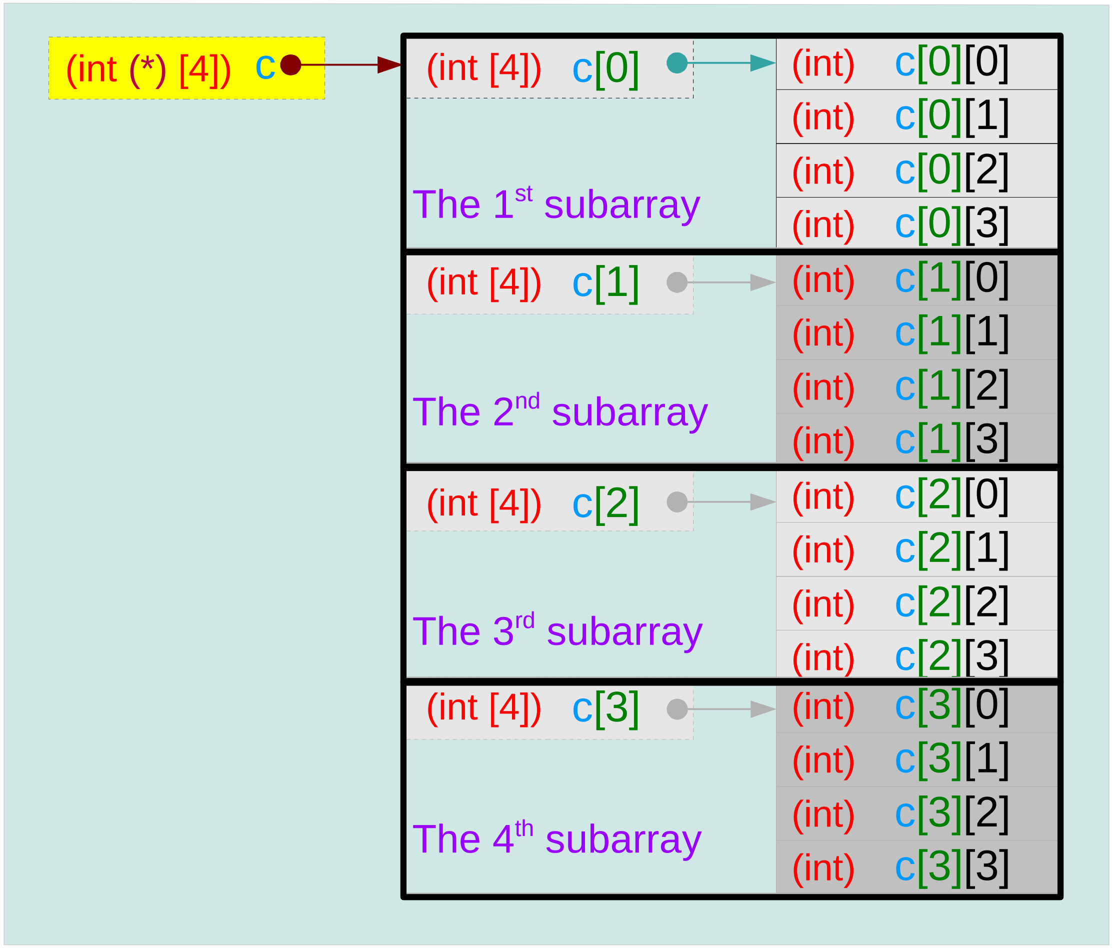
int c[4][4];

c[i]

- the **1-d** array name
- the **1-d** array starting address
- the **0-d** array pointer
  points to its scalar integer

c[0]	the 1st	**1-d** subarray name
c[1]	the 2nd	**1-d** subarray name
c[2]	the 3rd	**1-d** subarray name
c[3]	the 4th	**1-d** subarray name

compilers do not allocate
c[i]’s memory location



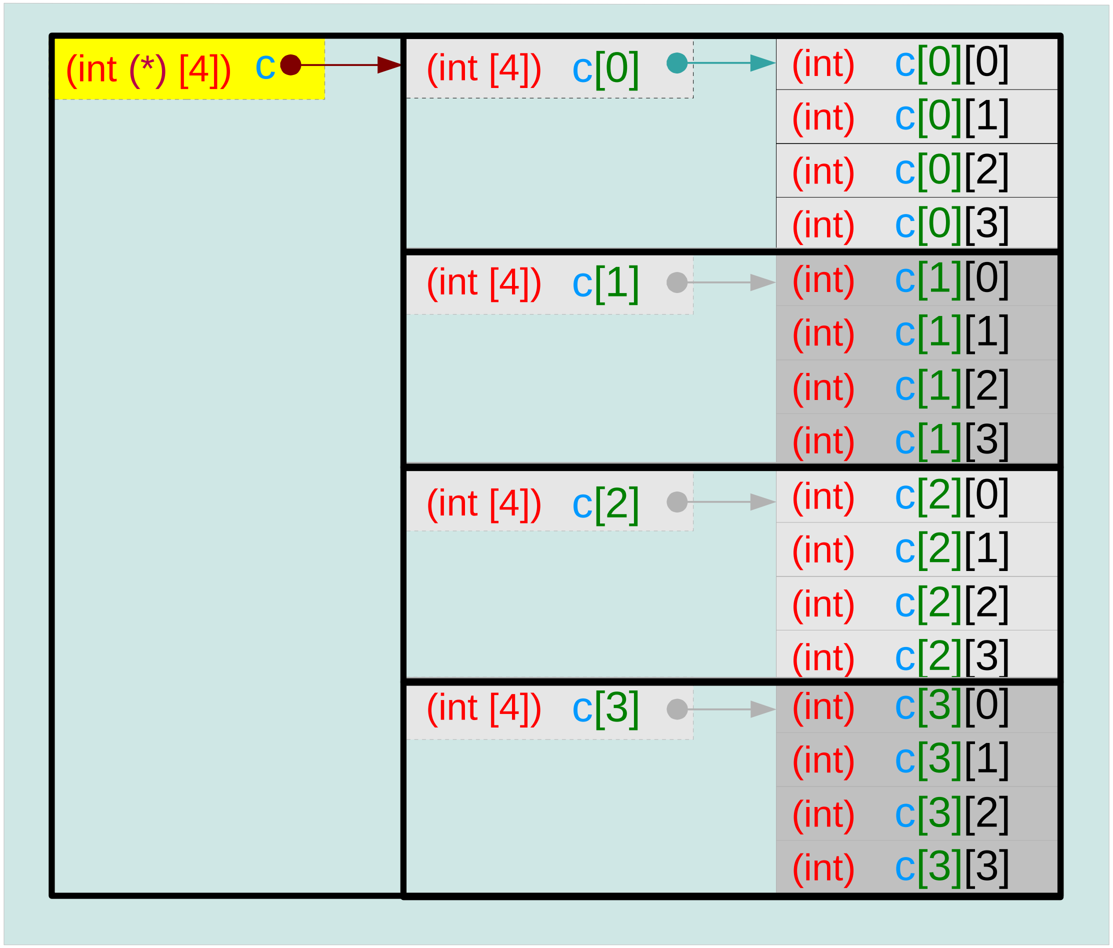
A **2-d** array and its sub-arrays – type sizes

sizeof(c) = 4*4*4 bytes

sizeof(c[i]) = 4*4 bytes

sizeof(c[i][j]) = 4 bytes

c : the **2-d** array name
c[i] : the **1-d** array name
c[i][j] : the **0-d** array name
(a scalar integer)



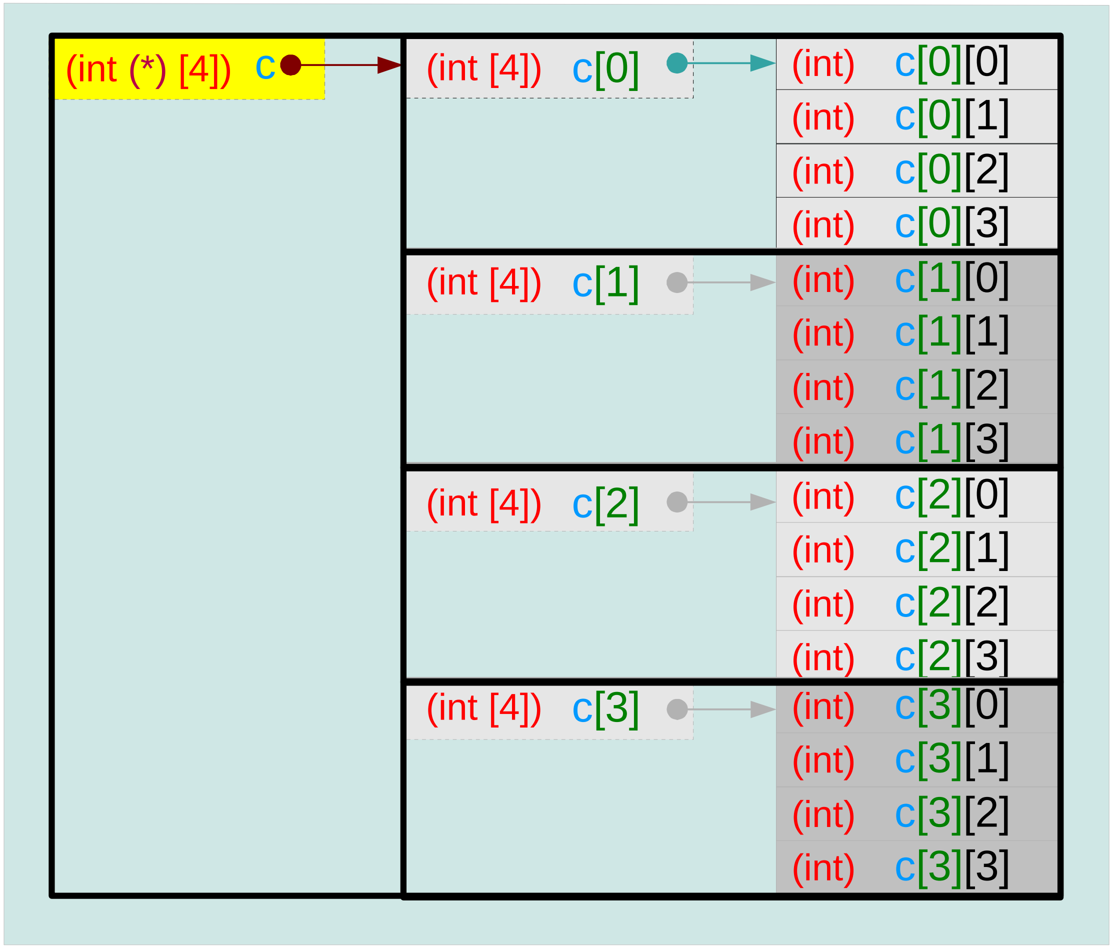
A **2-d** array and its **1-d** sub-arrays – a type view

2-d array name c	int (*) [4]
1-d array pointer c	int (*) [4]

1-d subarray name c[0]	int [4]
1-d subarray name c[1]	int [4]
1-d subarray name c[2]	int [4]
1-d subarray name c[3]	int [4]

c and c[0]

- different types
- the same address of the starting element



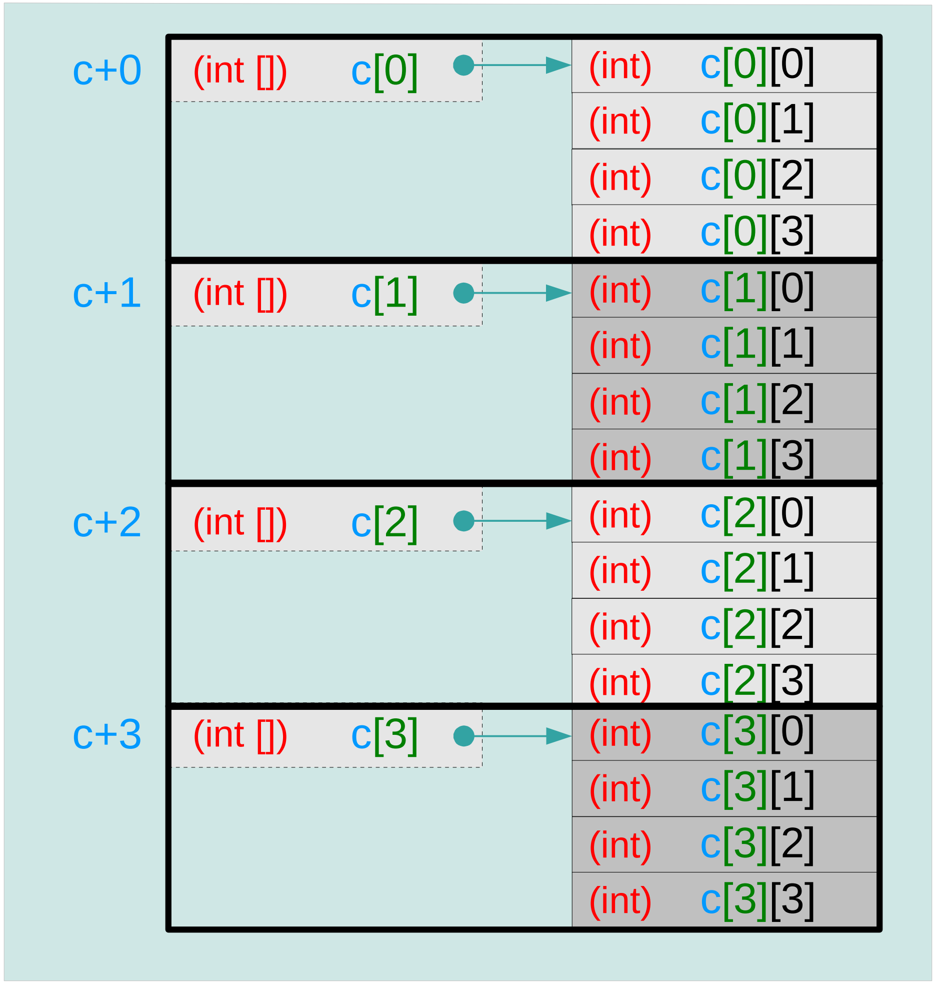
1-d subarray aggregated data type

The 1st subarray c[0] (=subarray name)
sizeof(c[0]) = 4*4 bytes
(c+0) : start address

The 2nd subarray c[1] (=subarray name)
sizeof(c[1]) = 4*4 bytes
(c+1) : start address

The 3rd subarray c[2] (=subarray name)
sizeof(c[2]) = 4*4 bytes
(c+2) : start address

The 4th subarray c[3] (=subarray name)
sizeof(c[3]) = 4*4 bytes
(c+3) : start address



2-d array name as a pointer to a **1-d** subarray

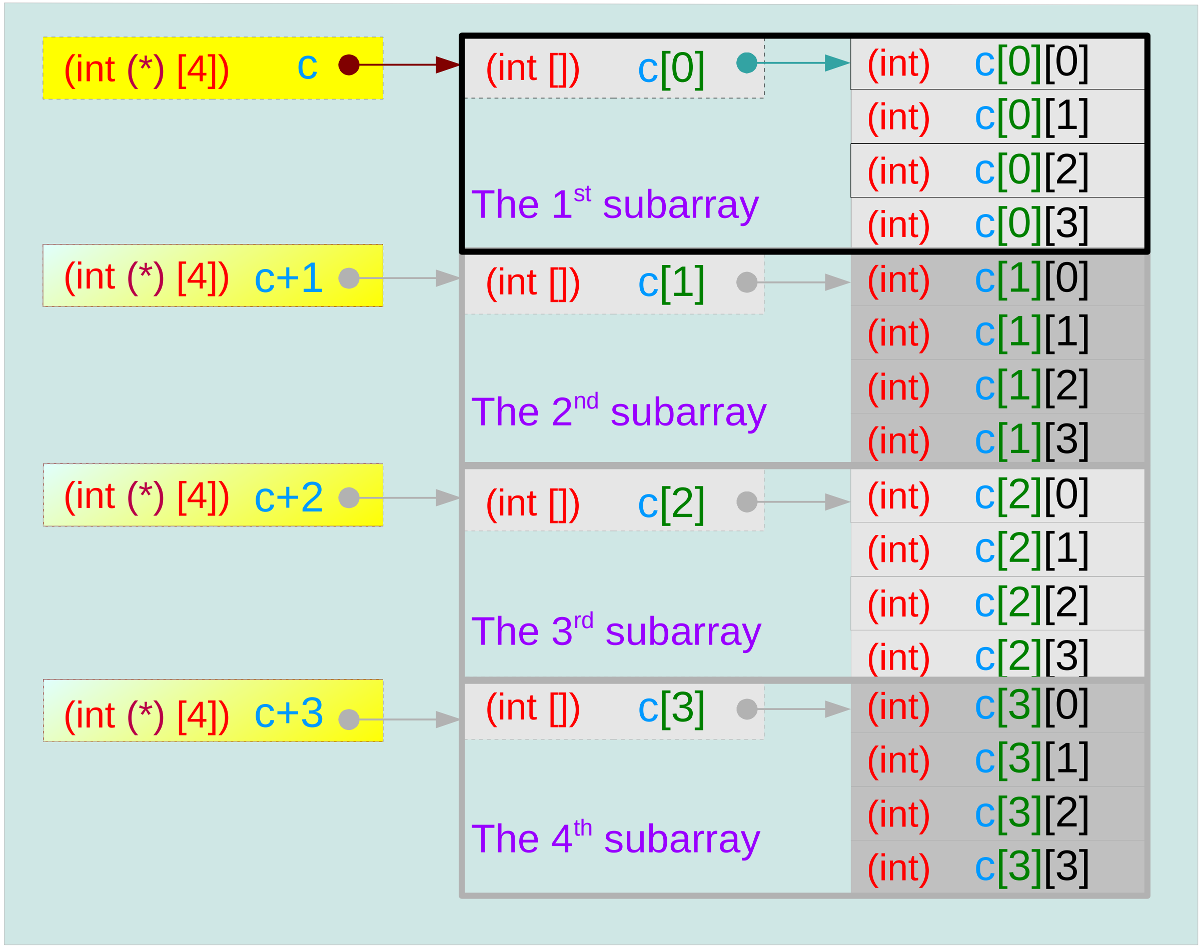
2-d array name **c**

1-d array pointer **c**

1-d array pointer **c+1**

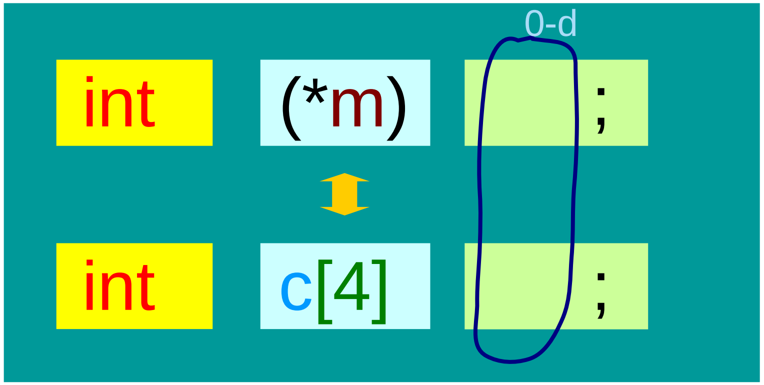
1-d array pointer **c+2**

1-d array pointer **c+3**



1-d array and **0-d** and **1-d** array pointers

0-d array pointer : int pointer



0-d

int (*m) ;

int c[4] ;

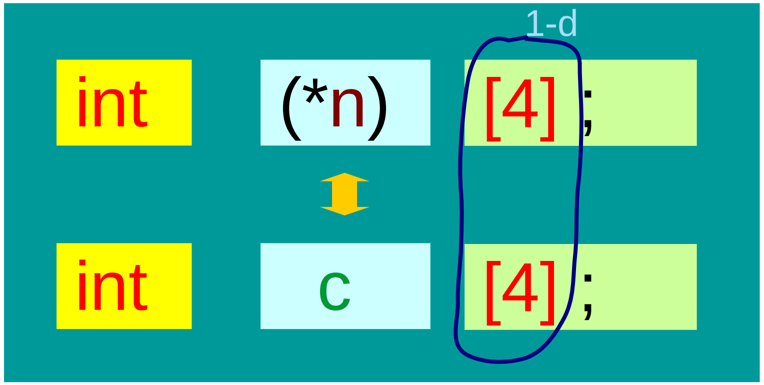
(int (*))

m = c;

m = &c[0];

m[**i**] ≡ c[**i**]

1-d array pointer



1-d

int (*n) [4];

int c [4];

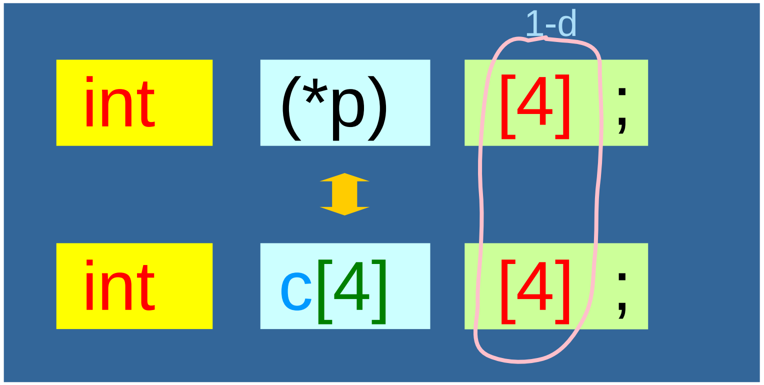
(int(*)[4])

n = &c;

(*n)[**i**] ≡ n[0][**i**] ≡ c[**i**]

2-d array and **1-d** and **2-d** array pointers

1-d array pointer



1-d

int (*p) [4] ;

int c[4] [4] ;

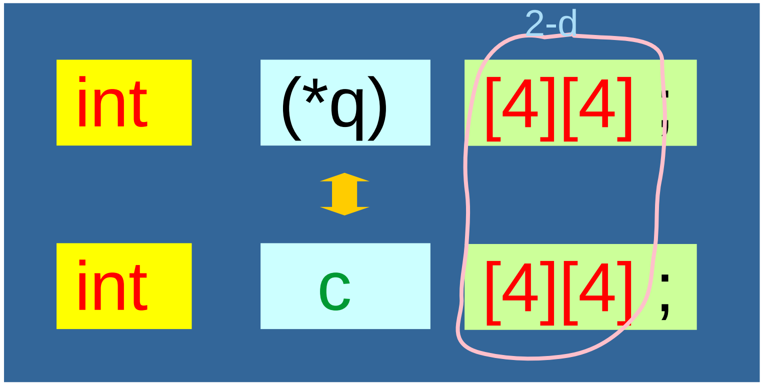
(int (*) [4])

p = c;

p = &c[0];

p[**i**] ≡ c[**i**]

2-d array pointer



2-d

int (*q) [4][4] ;

int c [4][4] ;

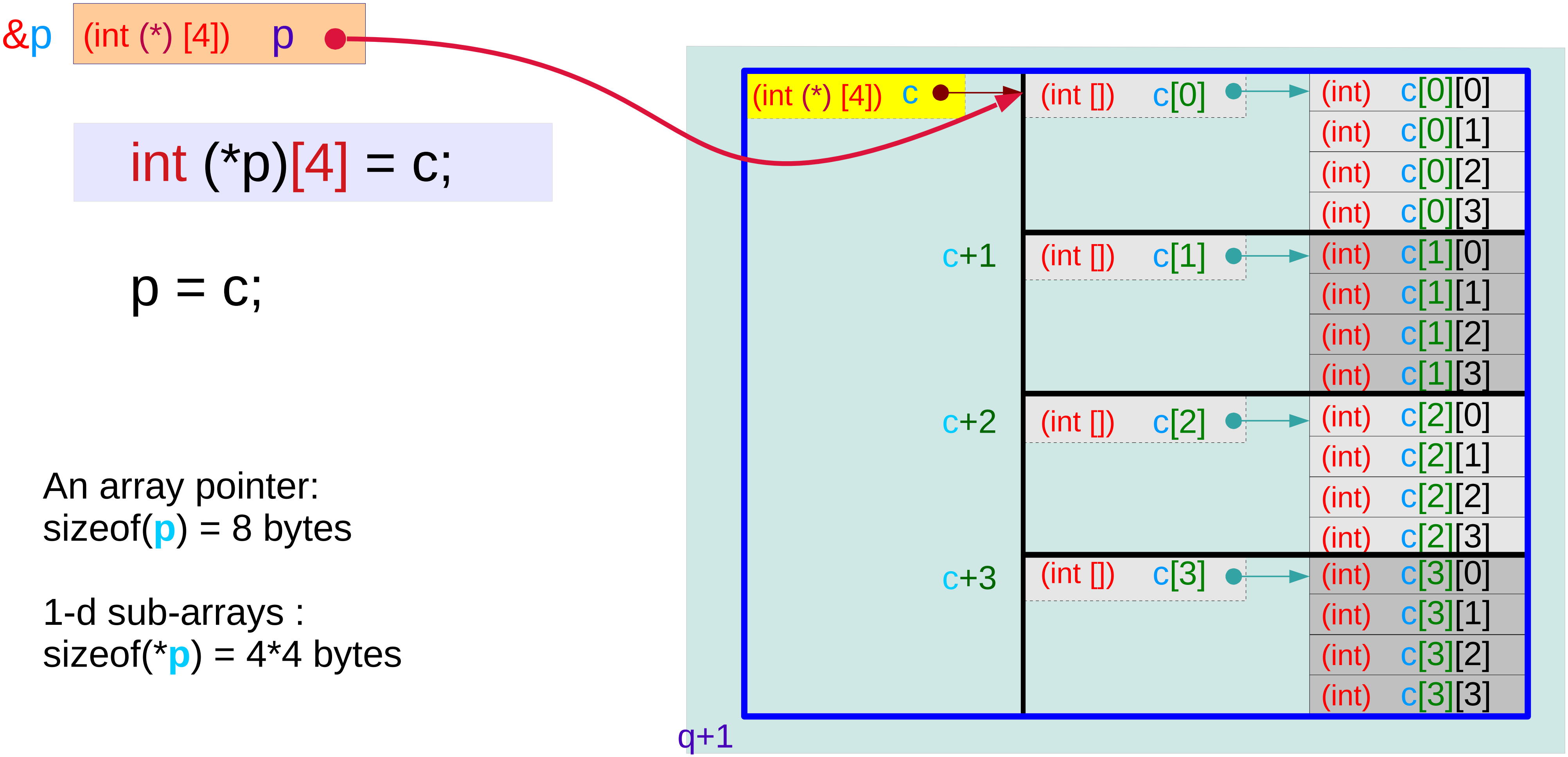
(int(*)[4][4])

q = &c;

(*q)[**i**][**j**] ≡ q[0][**i**][**j**] ≡ c[**i**][**j**]

1-d array pointer to the **1-d** subarray of a **2-d** array

1-d array pointer



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

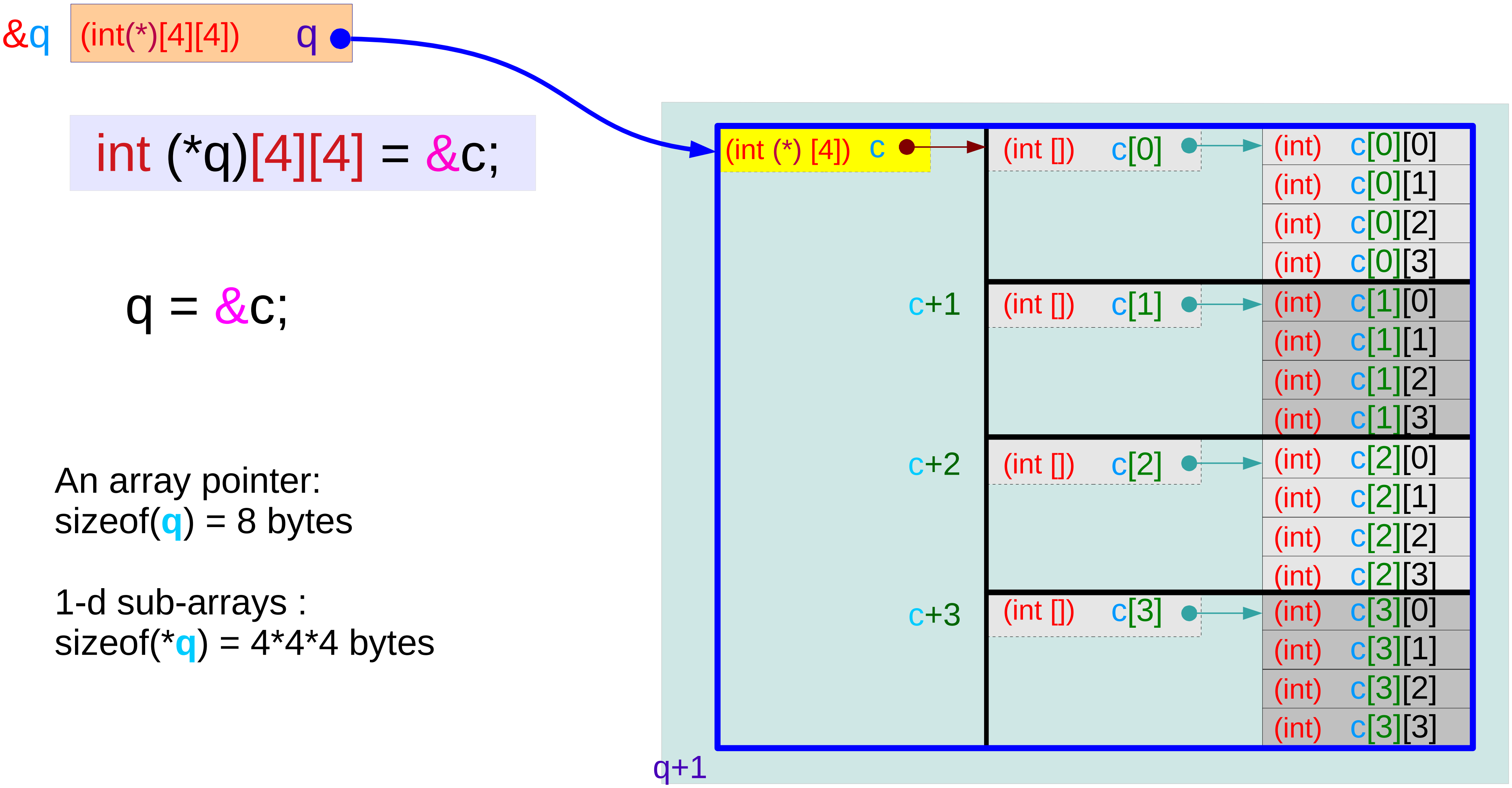
```
p = c;
```

An array pointer:
sizeof(p) = 8 bytes

1-d sub-arrays :
sizeof(*p) = 4*4 bytes

2-d array pointer to a **2-d** array

2-d array pointer



```
int (*q)[4][4] = &c;
```

```
q = &c;
```

An array pointer:
sizeof(**q**) = 8 bytes

1-d sub-arrays :
sizeof(***q**) = 4*4*4 bytes

Using a **1-d** array pointer to a **2-d** array

1-d

int (*p) [4] ;

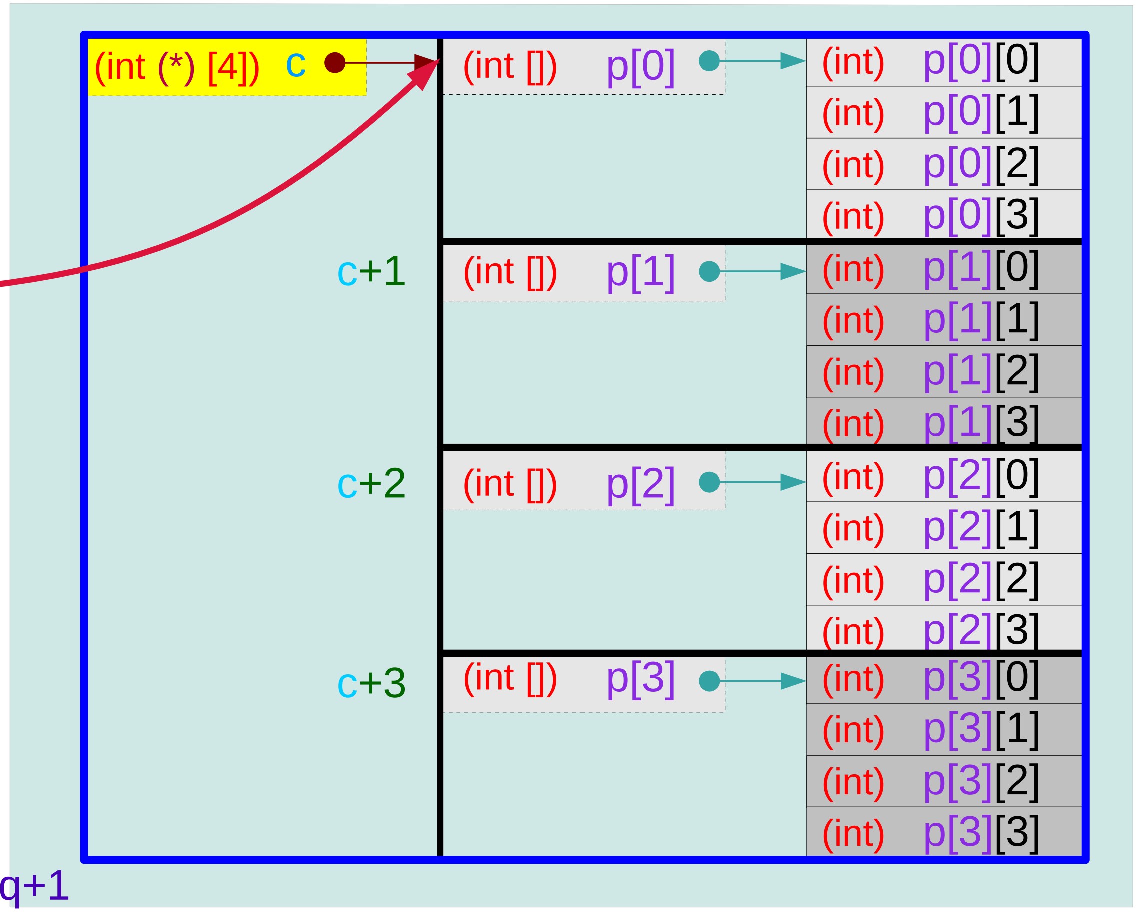
int c[4] [4]

1-d array pointer

&p (int (*) [4]) p

p = c;

p[i] ≡ c[i]



Using a **2-d** array pointer to a **2-d** array

2-d

int (*q) [4][4] ;

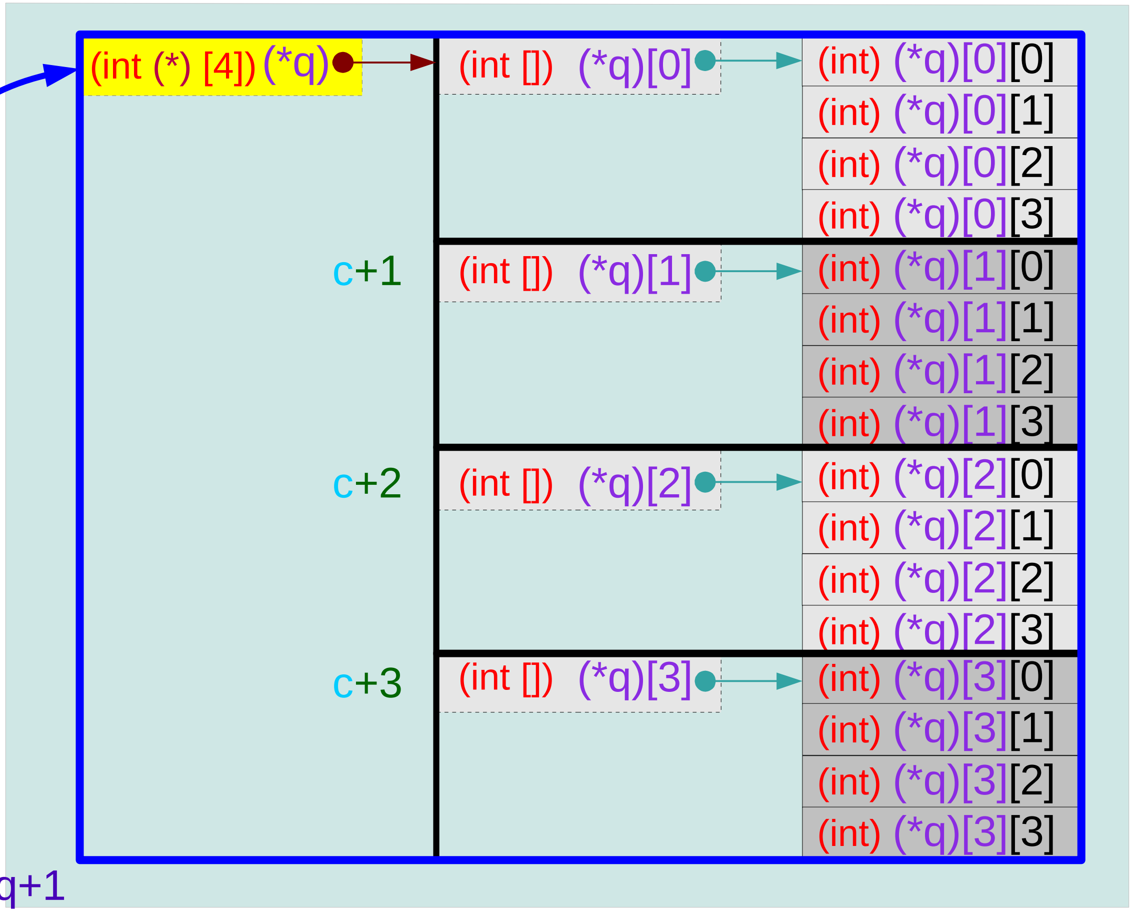
int c [4][4] ;

2-d array pointer

&p (int(*)[4][4]) q

q = &c;

(*q)[i] ≡ c[i]



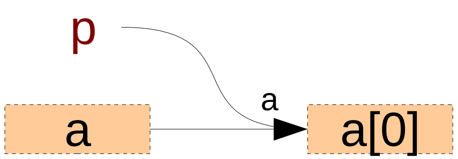
(*n-1*)-d array pointer to a *n*-d array

`int a[4] ;`	**1-d** array	
`int (*p) ;`	**0-d** array pointer	(p = a)
`int b[4] [2];`	**2-d** array	
`int (*q) [2];`	**1-d** array pointer	(q = b)
`int c[4] [2][3];`	**3-d** array	
`int (*r) [2][3];`	**2-d** array pointer	(r = c)
`int d[4] [2][3][4];`	**4-d** array	
`int (*s) [2][3][4];`	**3-d** array pointer	(s = d)

***n*-d array name and (*n-1*)-d array pointer**

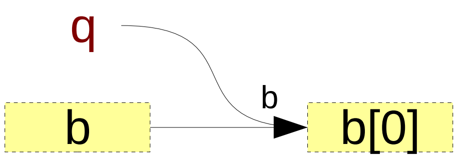
```
int a[4] ;
int (*p) ;
```

```
p = &a[0];
p = a;
```



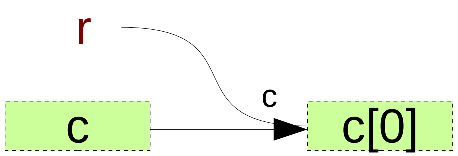
```
int b[4]  [2];
int (*q)  [2];
```

```
q = &b[0];
q = b;
```



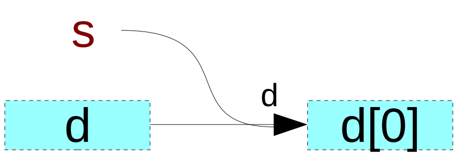
```
int c[4]  [2][3];
int (*r)  [2][3];
```

```
r = &c[0];
r = c;
```



```
int d[4]  [2][3][4];
int (*s)  [2][3][4];
```

```
s = &d[0];
s = d;
```



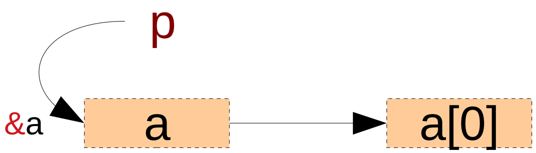
***n*-d** array pointer to a ***n*-d** array

`int a [4] ;`	**1-d** array	
`int (*p) [4];`	**1-d** array pointer	(p = &a)
`int b [4][2];`	**2-d** array	
`int (*q) [4][2];`	**2-d** array pointer	(q = &b)
`int c [4][2][3];`	**3-d** array	
`int (*r) [4][2][3];`	**3-d** array pointer	(r = &c)
`int d [4][2][3][4];`	**4-d** array	
`int (*s) [4][2][3][4];`	**4-d** array pointer	(s = &d)

***n*-d** array name and ***n*-d** array pointer

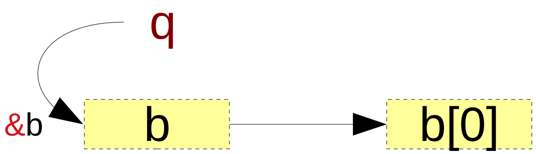
```
int  a   [4] ;
int (*p) [4];
```

```
p = &a;
```



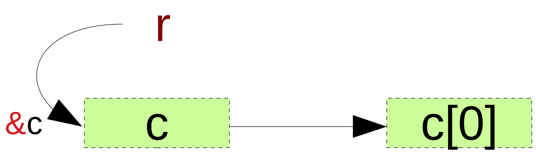
```
int  b   [4][2];
int (*q) [4][2];
```

```
q = &b;
```



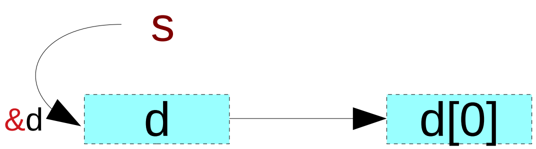
```
int  c   [4][2][3];
int (*r) [4][2][3];
```

```
r = &c;
```

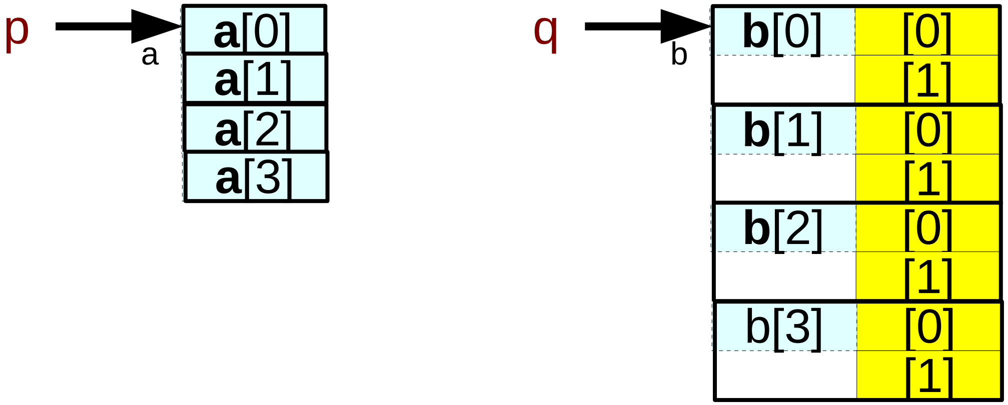


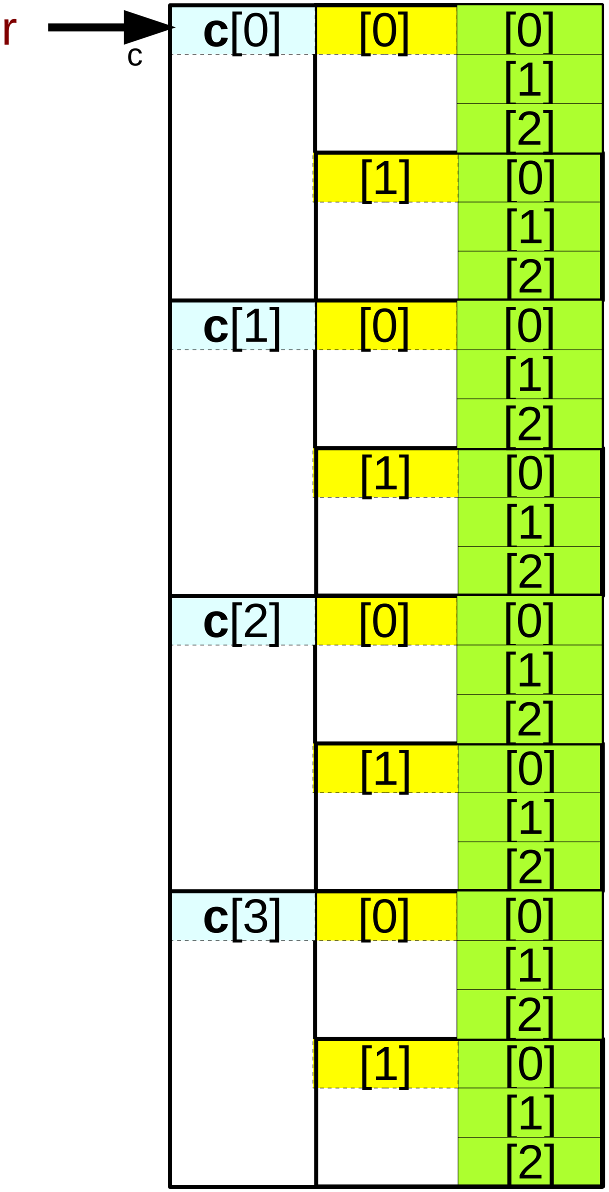
```
int  d   [4][2][3][4];
int (*s) [4][2][3][4];
```

```
s = &d;
```



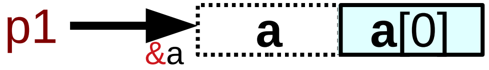
multi-dimensional array pointers



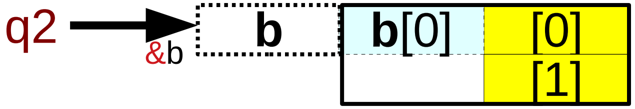


int a[4] ;		**1-d** array
int (*p) ;		**0-d** array pointer
int b[4]	[2];	**2-d** array
int (*q)	[2];	**1-d** array pointer
int c[4]	[2][3];	**3-d** array
int (*r)	[2][3];	**2-d** array pointer
int d[4]	[2][3][4];	**4-d** array
int (*s)	[2][3][4];	**3-d** array pointer

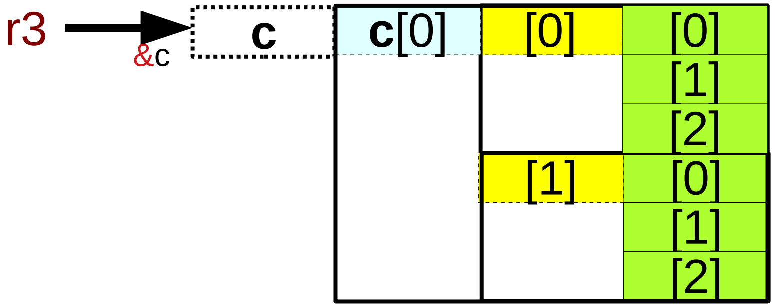
Initializing ***n-d*** array pointers



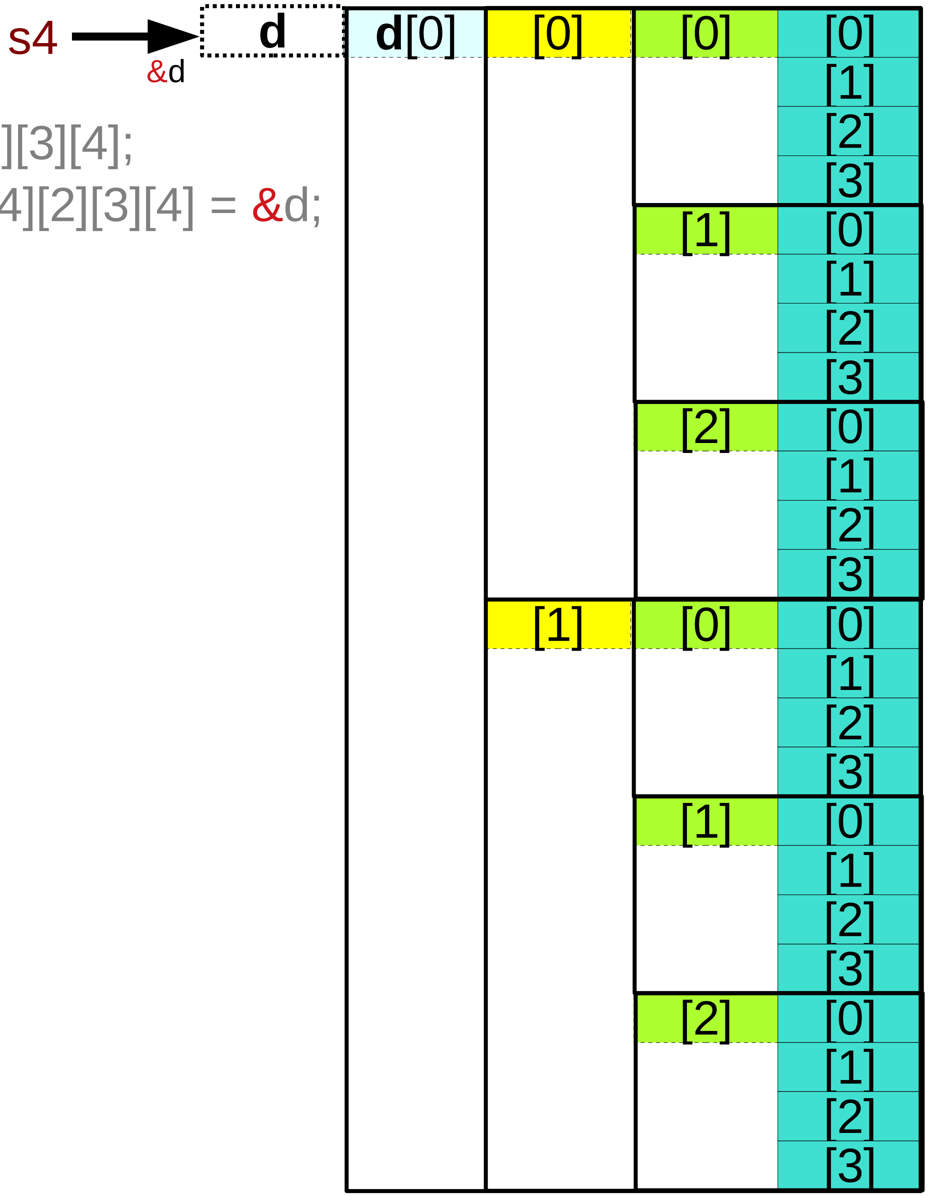
```
int a[4] ;
int (*p1)[4] = &a ;
```



```
int b[4][2];
int (*q2)[4][2] = &b;
```

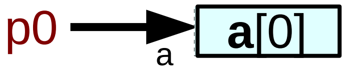


```
int c[4][2][3];
int (*r3)[4][2][3] = &c;
```

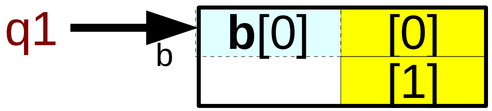


```
int d[4][2][3][4];
int (*s4)[4][2][3][4] = &d;
```

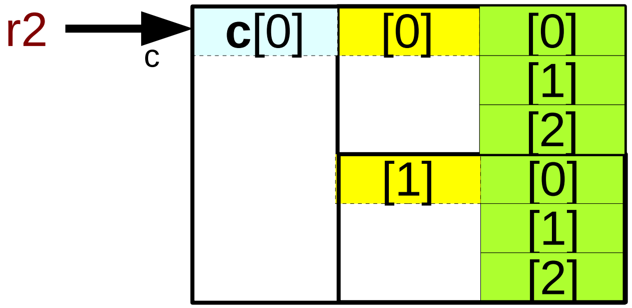
Initializing ***(n-1)-d*** array pointers



int a[4] ;
int (*p0) = a ;

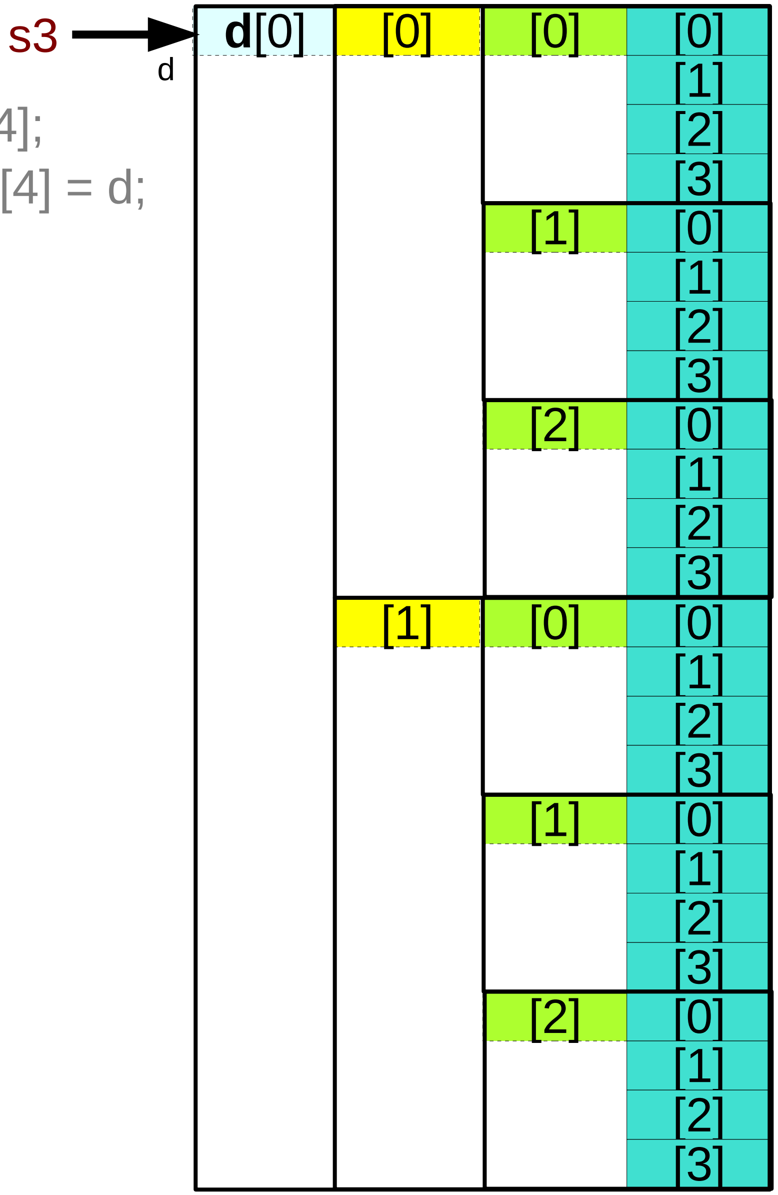


int b[4][2];
int (*q1)[2] = b;



int c[4][2][3];
int (*r2)[2][3] = c;

int d[4][2][3][4];
int (*s3)[2][3][4] = d;



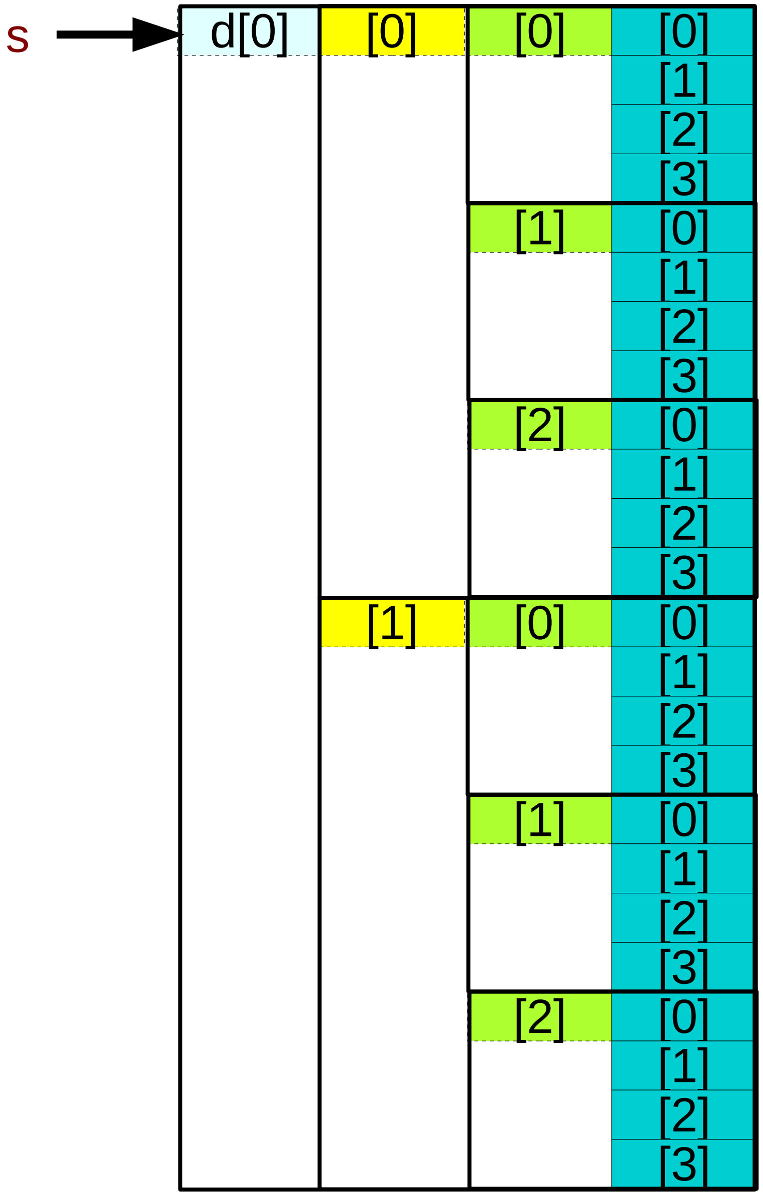
array pointers to multi-dimensional subarrays

```
int d[4] [2][3][4];
int (*s) [2][3][4];
```

d	4-d array name	d[4][2][3][4]
	3-d array pointer	(*p)[2][3][4]
d[i]	3-d array name	d[i][2][3][4]
	2-d array pointer	(*q)[3][4]
d[i][j]	2-d array name	d[i][j][3][4]
	1-d array pointer	(*r)[4]
d[i][j][k]	1-d array name	d[i][j][k][4]
	0-d array pointer	(*s)

i,j,k are specific index values

i =[0..3], j = [0..1], k= [0..2]



Initializing array pointers to multi-dimensional subarrays

```
int d[4] [2][3][4];
int (*s) [2][3][4];
```

d	4-d array name	d[4][2][3][4]	p[i][j][k][l]
	3-d array pointer	(*p)[2][3][4]	int (*p)[2][3][4] = d;
d[i]	3-d array name	d[i][2][3][4]	q[j][k][l]
	2-d array pointer	(*q)[3][4]	int (*q)[3][4] = d[i];
d[i][j]	2-d array name	d[i][j][3][4]	r[k][l]
	1-d array pointer	(*r)[4]	int (*r)[4] = d[i][j];
d[i][j][k]	1-d array name	d[i][j][k][4]	s[l]
	0-d array pointer	(*s)	int (*s) = d[i][j][k];

i =[0..3], j = [0..1], k= [0..2]

Passing multidimensional array names

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

call

```
funa(a, …);
```

prototype

```
void funa(int (*p), …);
```

```
int b[4]  [2];
int (*q)  [2];
```

call

```
funb(b, …);
```

prototype

```
void funb(int (*q)[2], …);
```

```
int c[4]  [2][3];
int (*r)  [2][3];
```

call

```
func(c, …);
```

prototype

```
void func(int (*r)[2][3], …);
```

```
int d[4]  [2][3][4];
int (*s)  [2][3][4];
```

call

```
fund(d, …);
```

prototype

```
void fund(int (*s)[2][3][4], …);
```

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