Array Access Methods (1A)

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

Young Won Lim
5/11/19

Contiguity constraints

int a[M][N] ;

(*(a+m))[n]; ⟺ a[m][n];
*(a[m]+n) ⟺ a[m][n]

int *b[M] ;

(*(b+m))[n]; ⟺ b[m][n];

int (*c)[N] ;

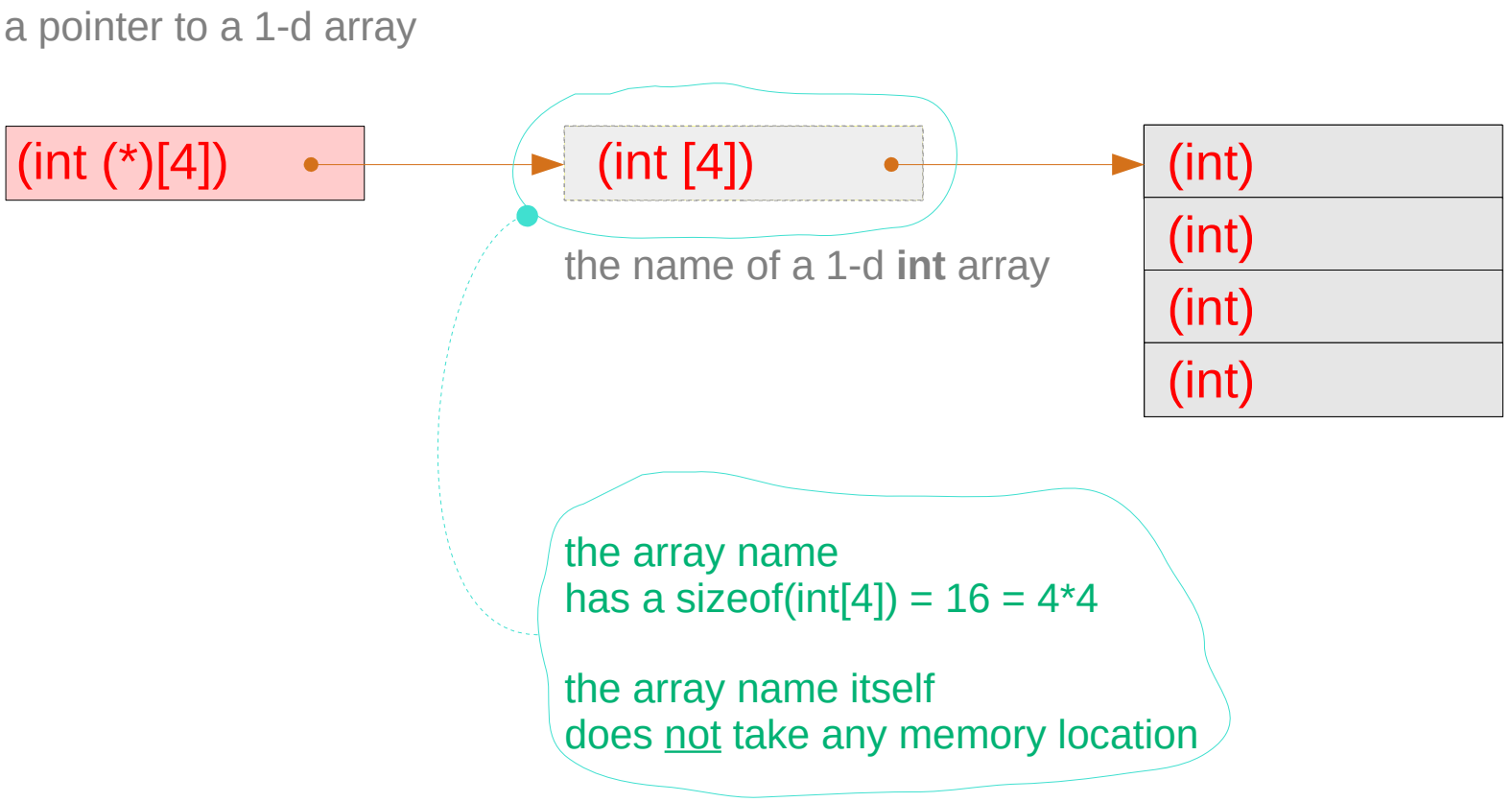
(*(c+m))[n]; ⟺ c[m][n];
*(c[m]+n) ⟺ c[m][n]

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

- assigning **1-d** array pointers **p1**, **p2**, **p3**
- type view of **1-d** array pointers **p1**, **p2**, **p3**
- variable view of **1-d** array pointers **p1**, **p2**, **p3**
- accessing **1-d** arrays via **p1**, **p2**, **p3** Contiguous and Non-contiguous

- contiguous **1-d** arrays **a**, **b**, **c** are assumed
- incrementing an array pointer **p**
- accessing **1-d** arrays by incrementing **p** Contiguous only

A **1-d** array pointer – a type view



Assigning **1-d** array pointers **p1**, **p2**, **p3**

1-d arrays

```
int a[4];
int b[4];
int c[4];
```

1-d array pointers

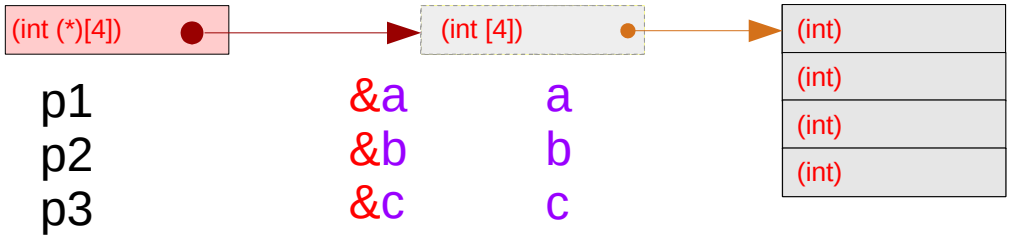
```
int (*p1)[4];
int (*p2)[4];
int (*p3)[4];
```

assignment

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

equivalence

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



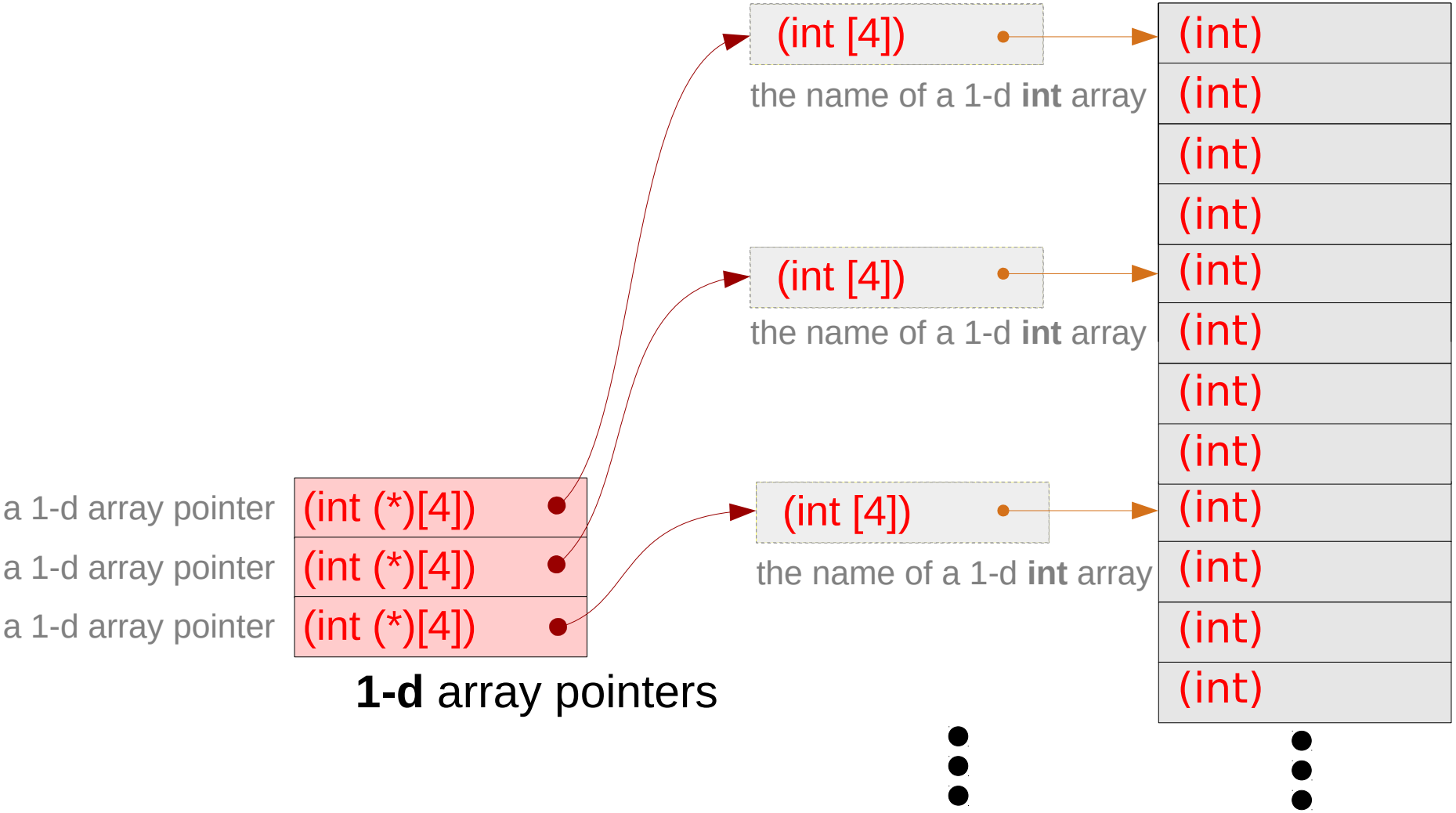
an int pointer

```
int  (*r);
```

a 2-d array pointer

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

Type view of **1-d** array pointers **p1**, **p2**, **p3**



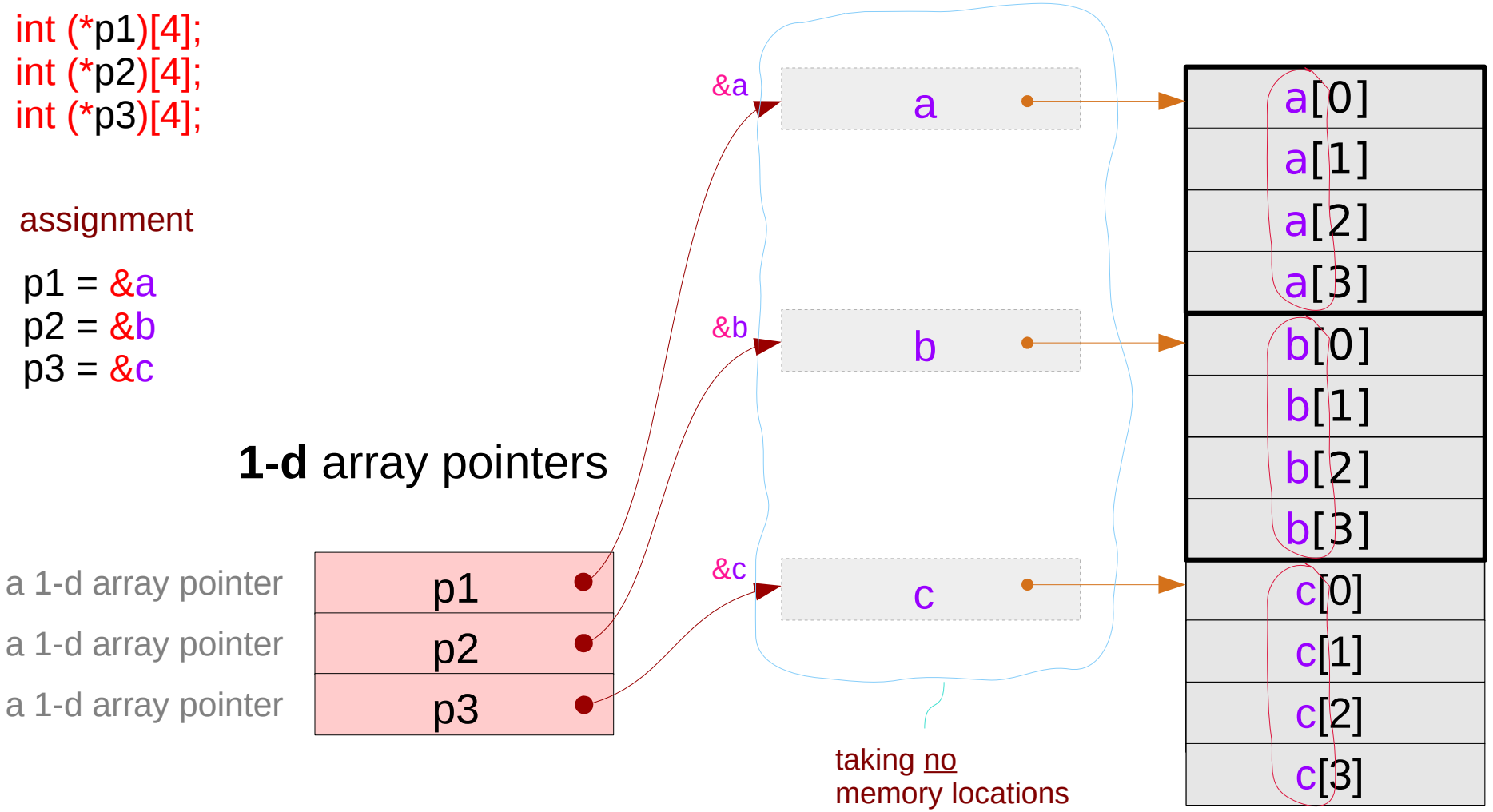
Variable view of **1-d** array pointers **p1**, **p2**, **p3**

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

assignment

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

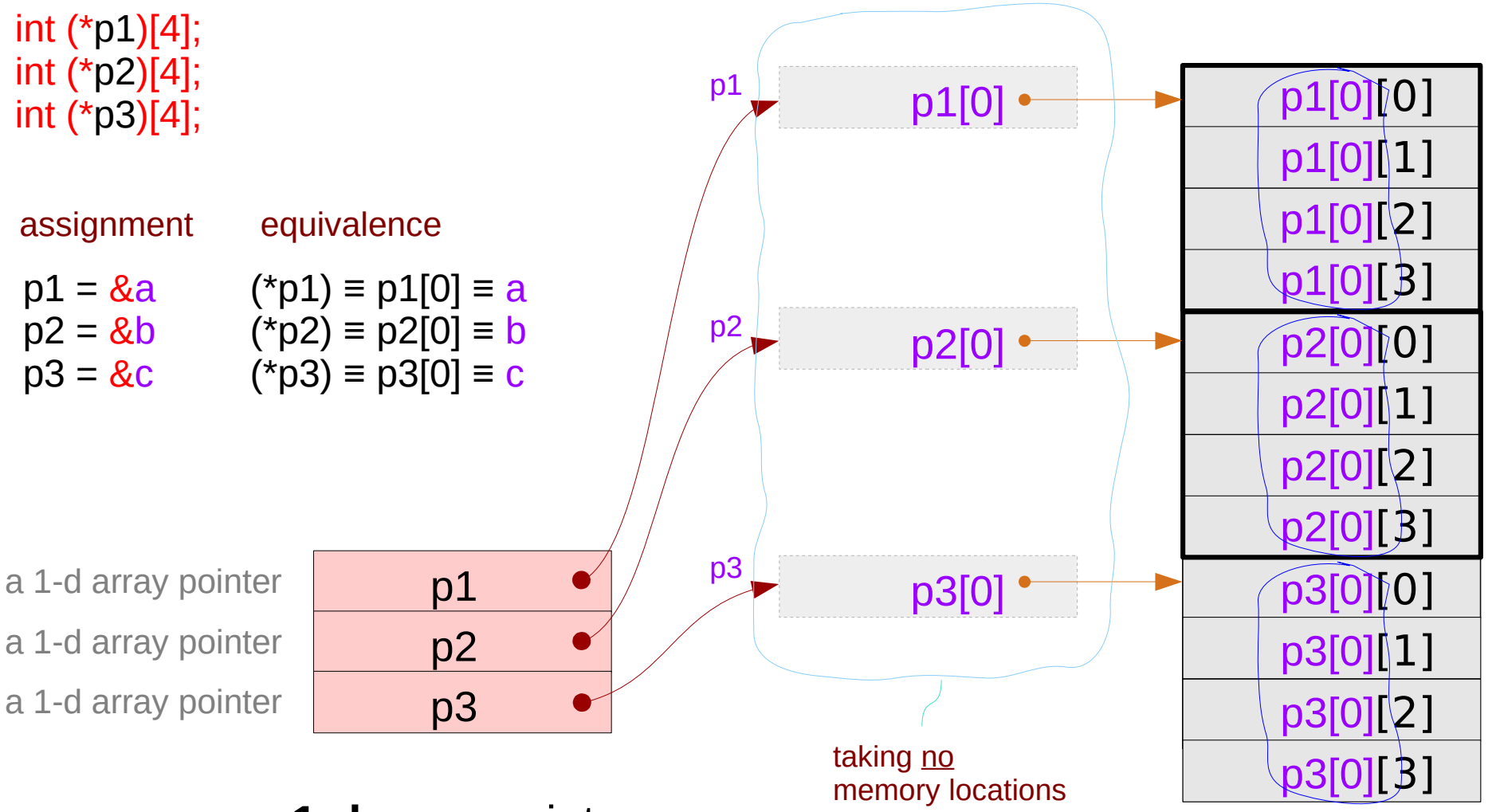
1-d array pointers



Accessing contiguous **1-d** arrays via **p1**, **p2**, **p3**

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

assignment	equivalence
p1 = &a	(*p1) ≡ p1[0] ≡ a
p2 = &b	(*p2) ≡ p2[0] ≡ b
p3 = &c	(*p3) ≡ p3[0] ≡ c



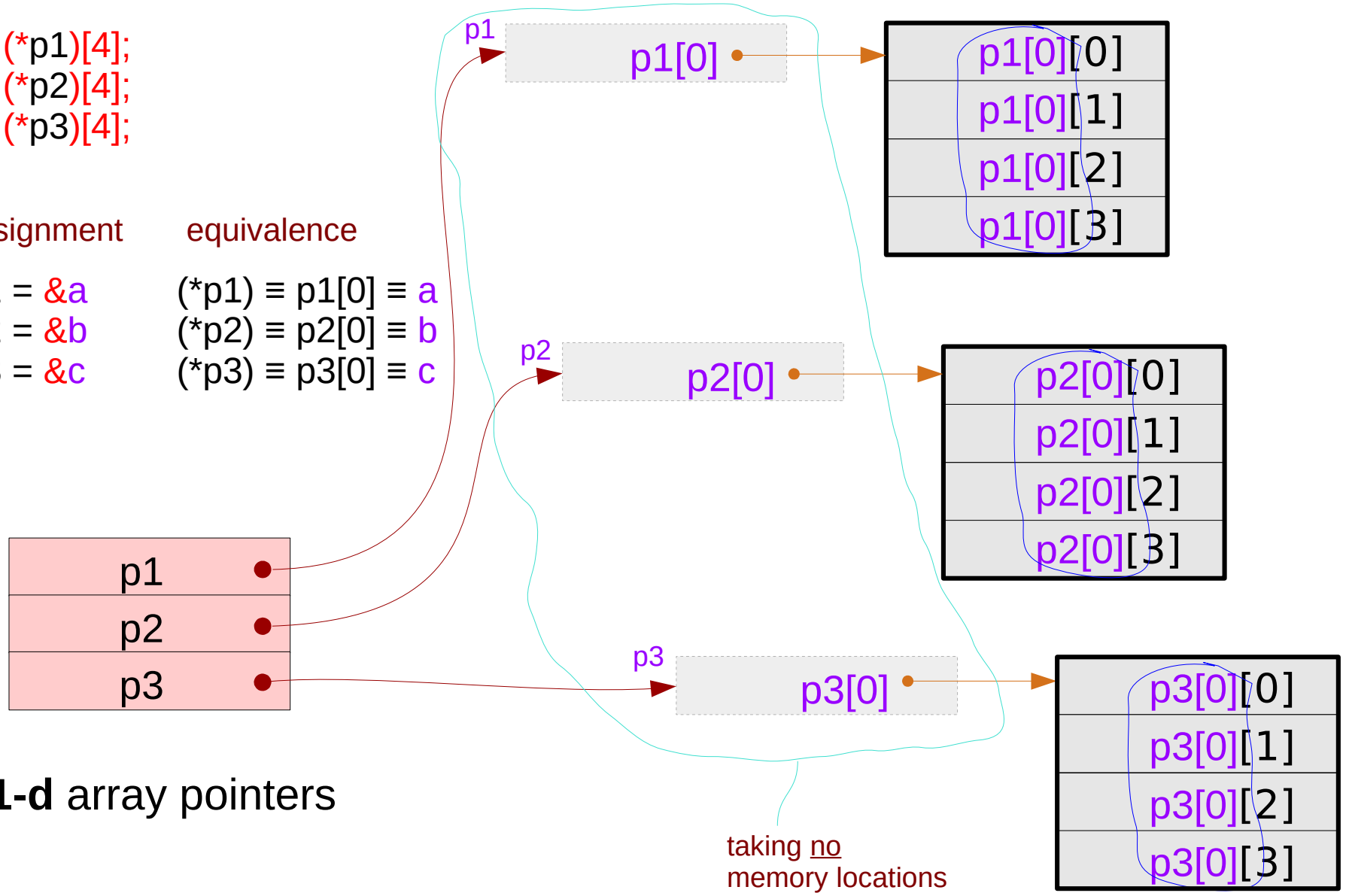
1-d array pointers

a, b, c : either contiguous or non-contiguous

Accessing non-contiguous **1-d** arrays via **p1**, **p2**, **p3**

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

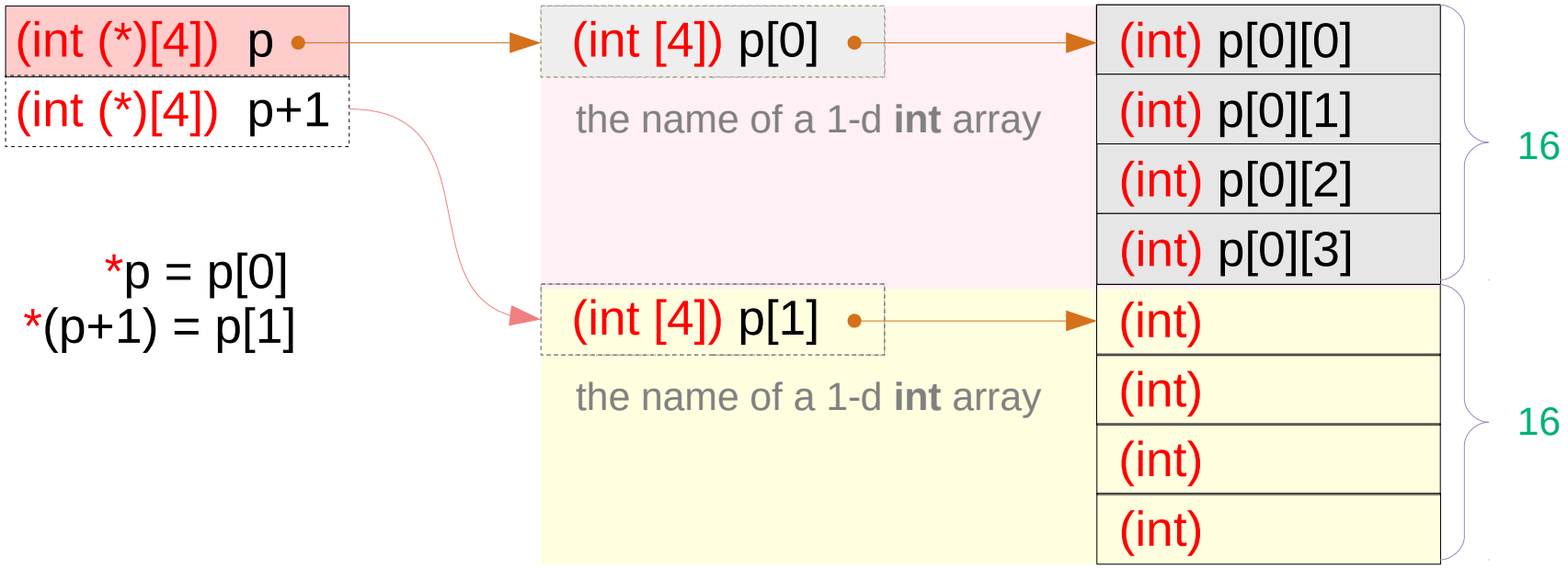
assignment	equivalence
p1 = &a	(*p1) ≡ p1[0] ≡ a
p2 = &b	(*p2) ≡ p2[0] ≡ b
p3 = &c	(*p3) ≡ p3[0] ≡ c



1-d array pointers

Incrementing an array pointer **p**

a pointer to a 1-d array



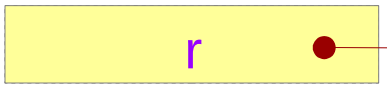
sizeof(*p) = sizeof(p[0]) = 16 = 4*4

(long)(p+1) – (long)p = 16

Contiguous **1-d** array **a**, **b**, **c** are assumed

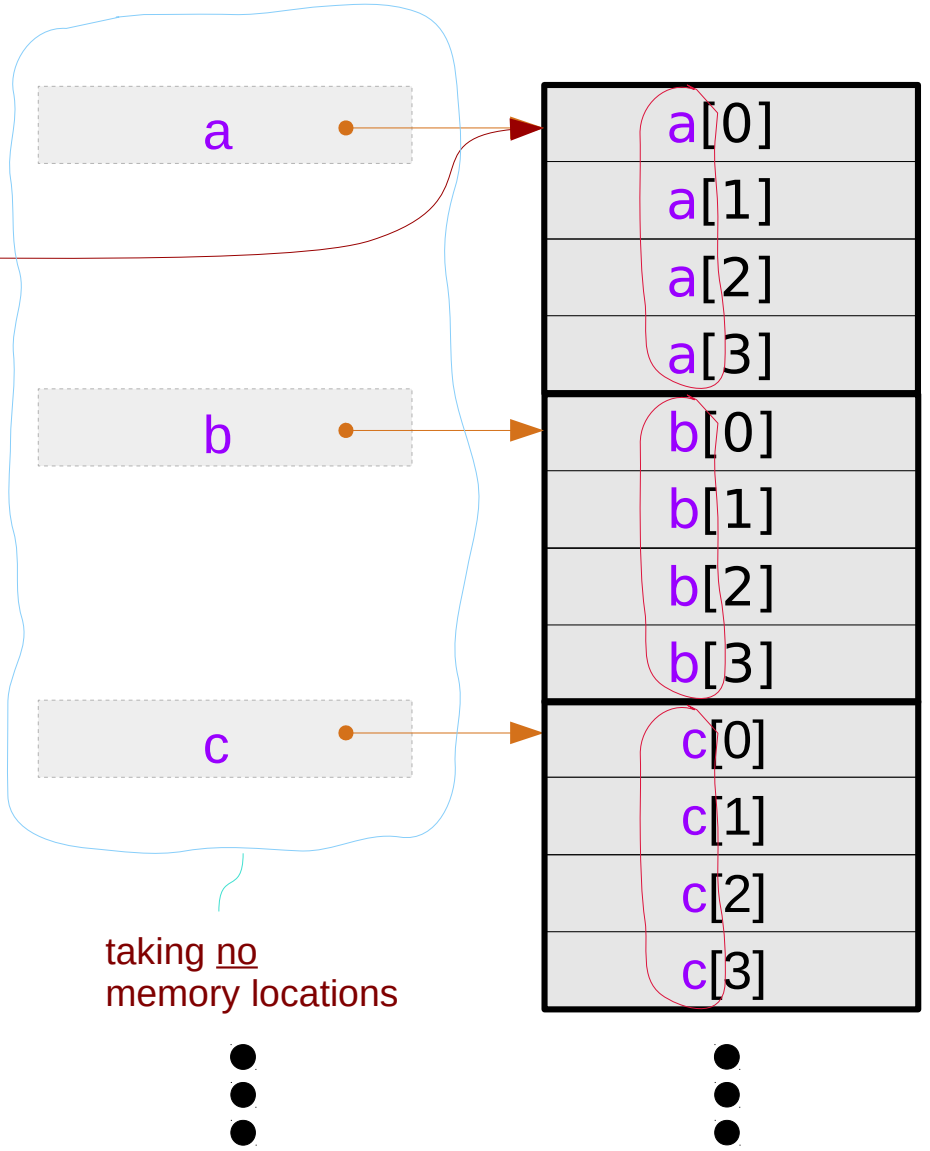
```
int a[4];
int b[4];
int c[4];

int  (*r) = a;
```



an integer pointer

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



taking no
memory locations

Accessing **1-d** arrays by incrementing pointer **p**

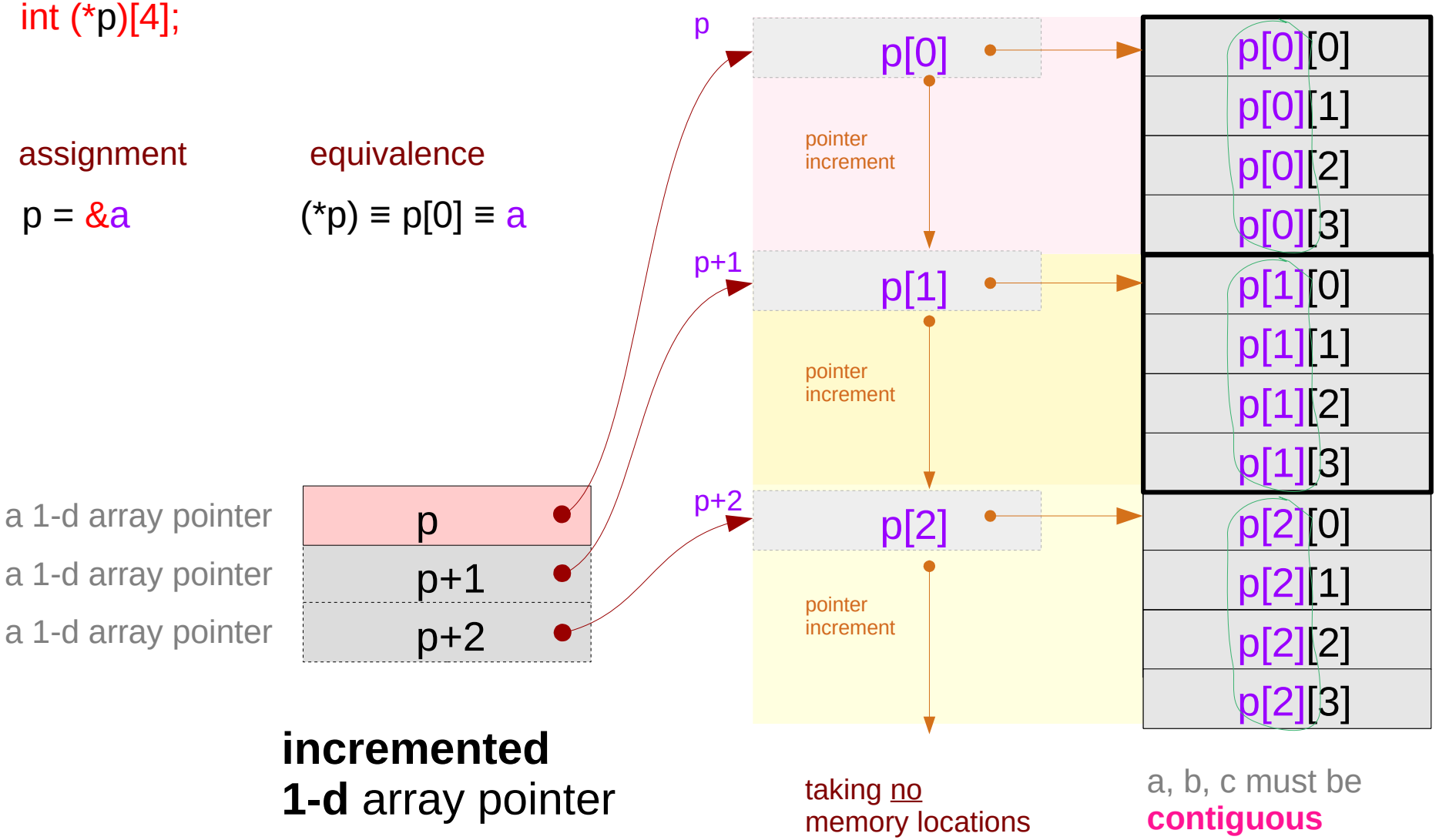
int (*p)[4];

assignment

p = &a

equivalence

(*p) ≡ p[0] ≡ a



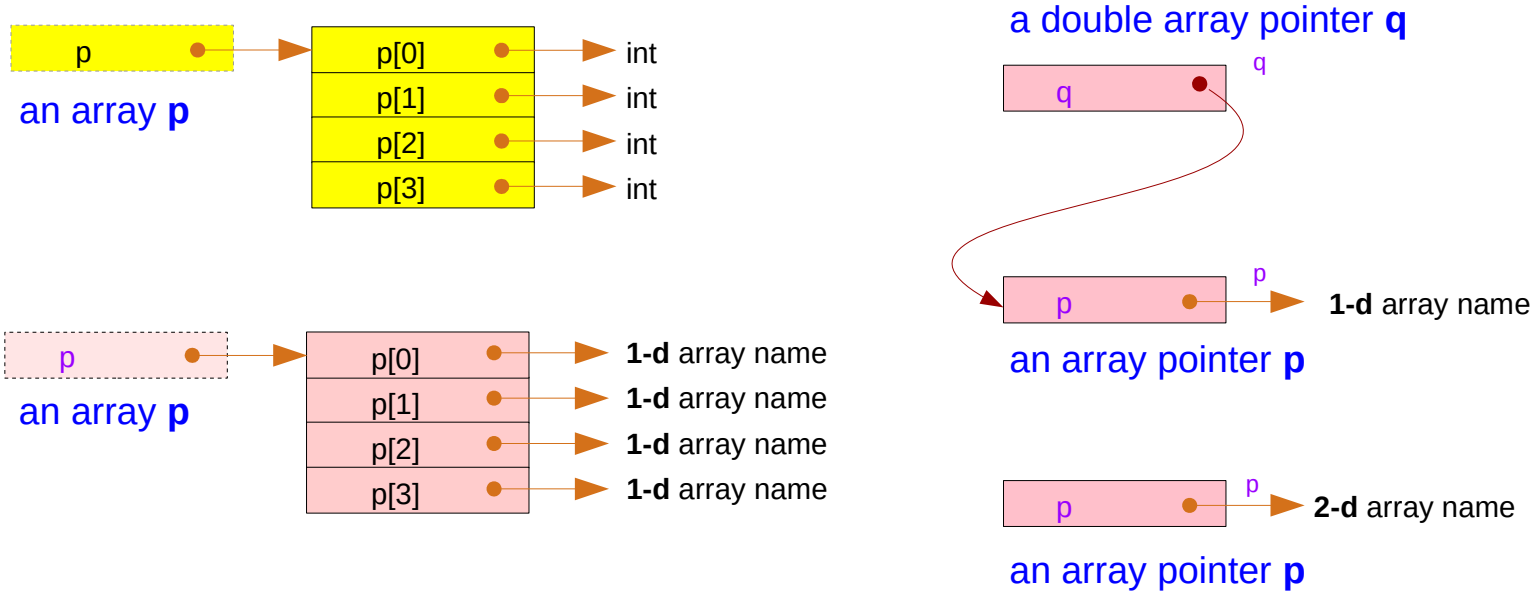
incremented
1-d array pointer

taking no
memory locations

a, b, c must be
contiguous

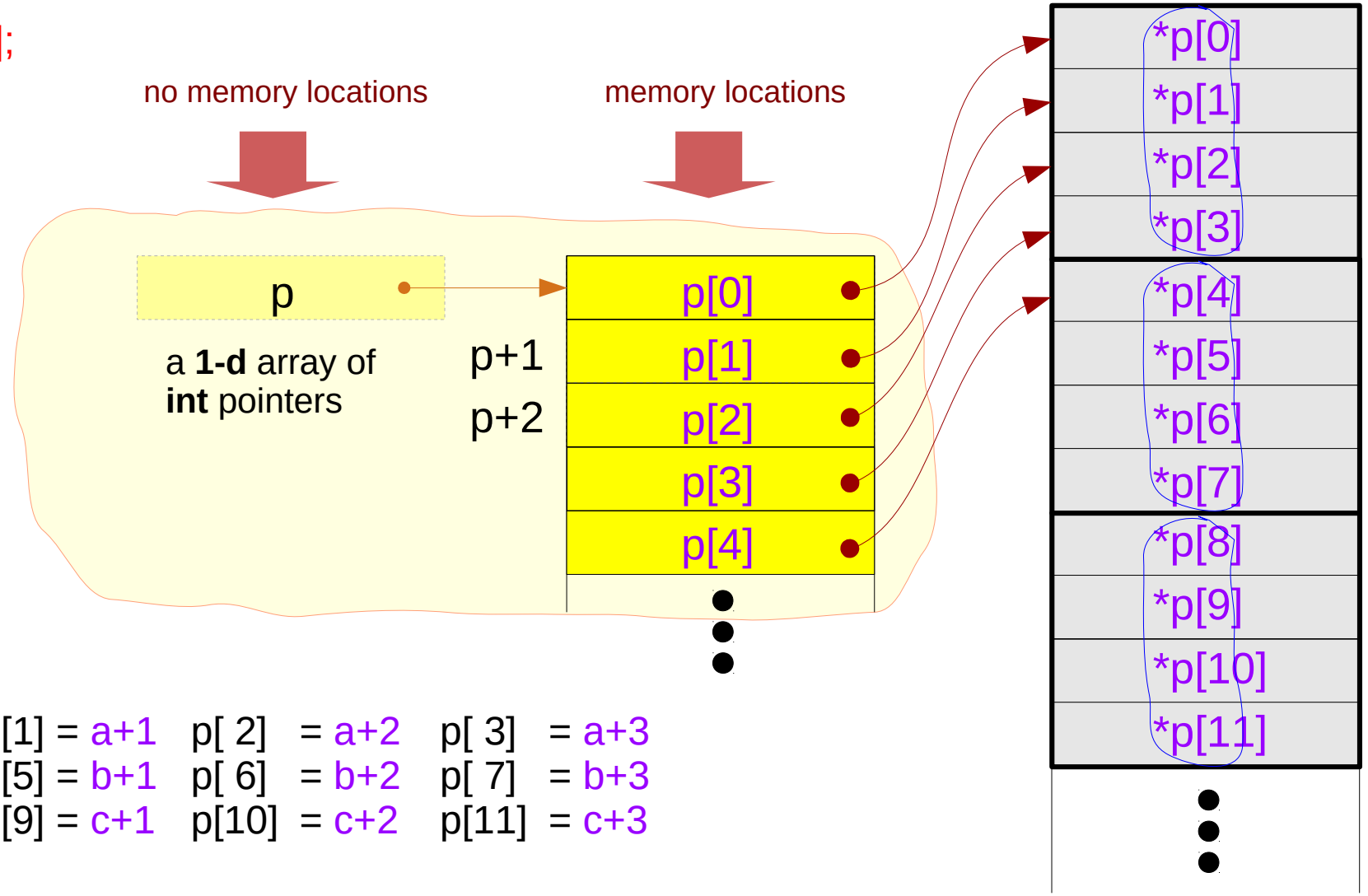
Other types of pointers

- **1-d** array **p** of integer pointers
- an array **p** of array pointers
- a double array pointer **q**
- **1-d** array pointer to consecutive **1-d** arrays
- **2-d** array pointer to consecutive **2-d** arrays



1-d array p of integer pointers (1)

int *p[4*4];

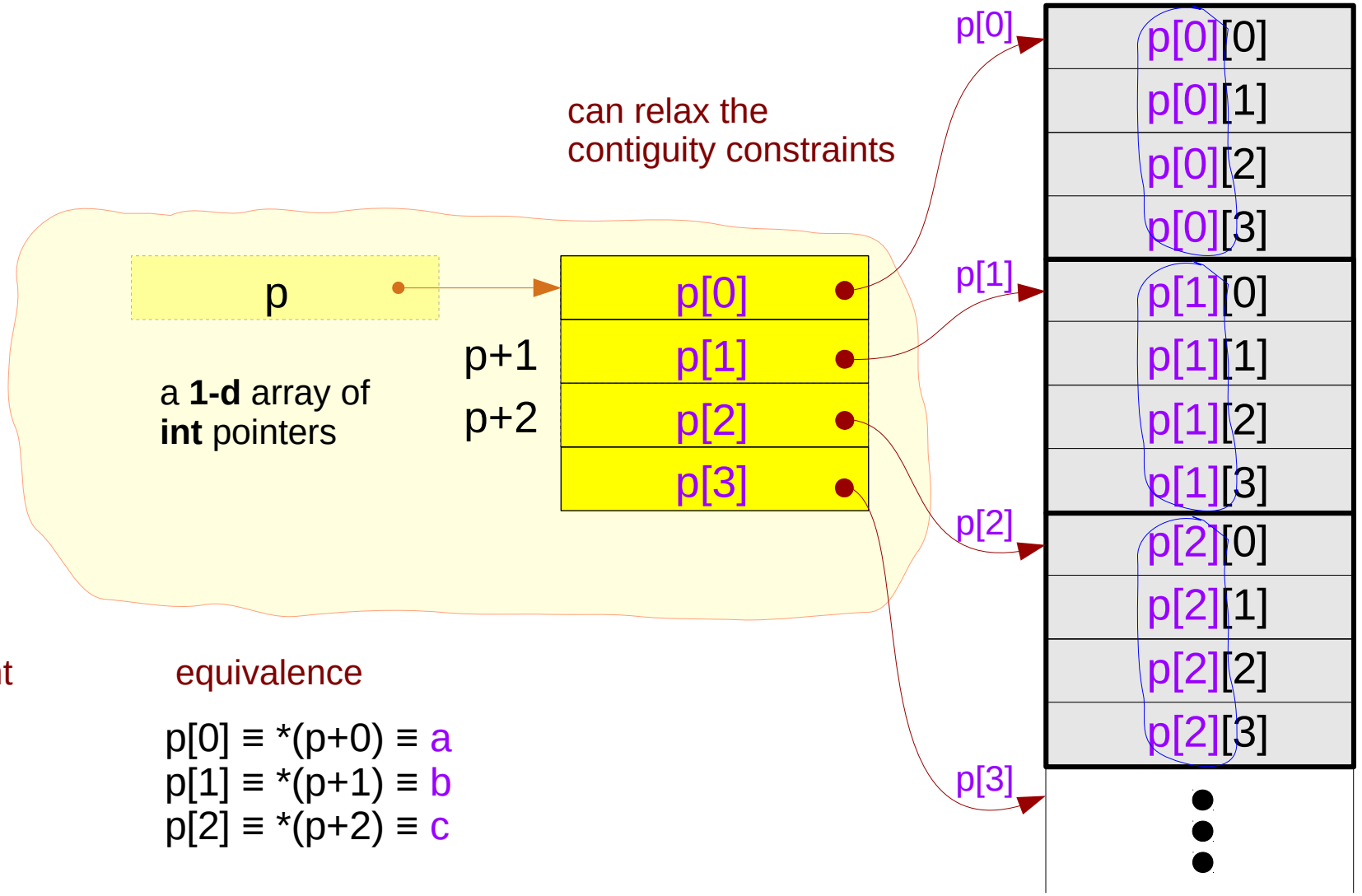


assignment

p[0] = a p[1] = a+1 p[2] = a+2 p[3] = a+3
p[4] = b p[5] = b+1 p[6] = b+2 p[7] = b+3
p[8] = c p[9] = c+1 p[10] = c+2 p[11] = c+3

1-d array p of integer pointers (2)

int *p[4];



assignment

p[0] = a
p[1] = b
p[2] = c

equivalence

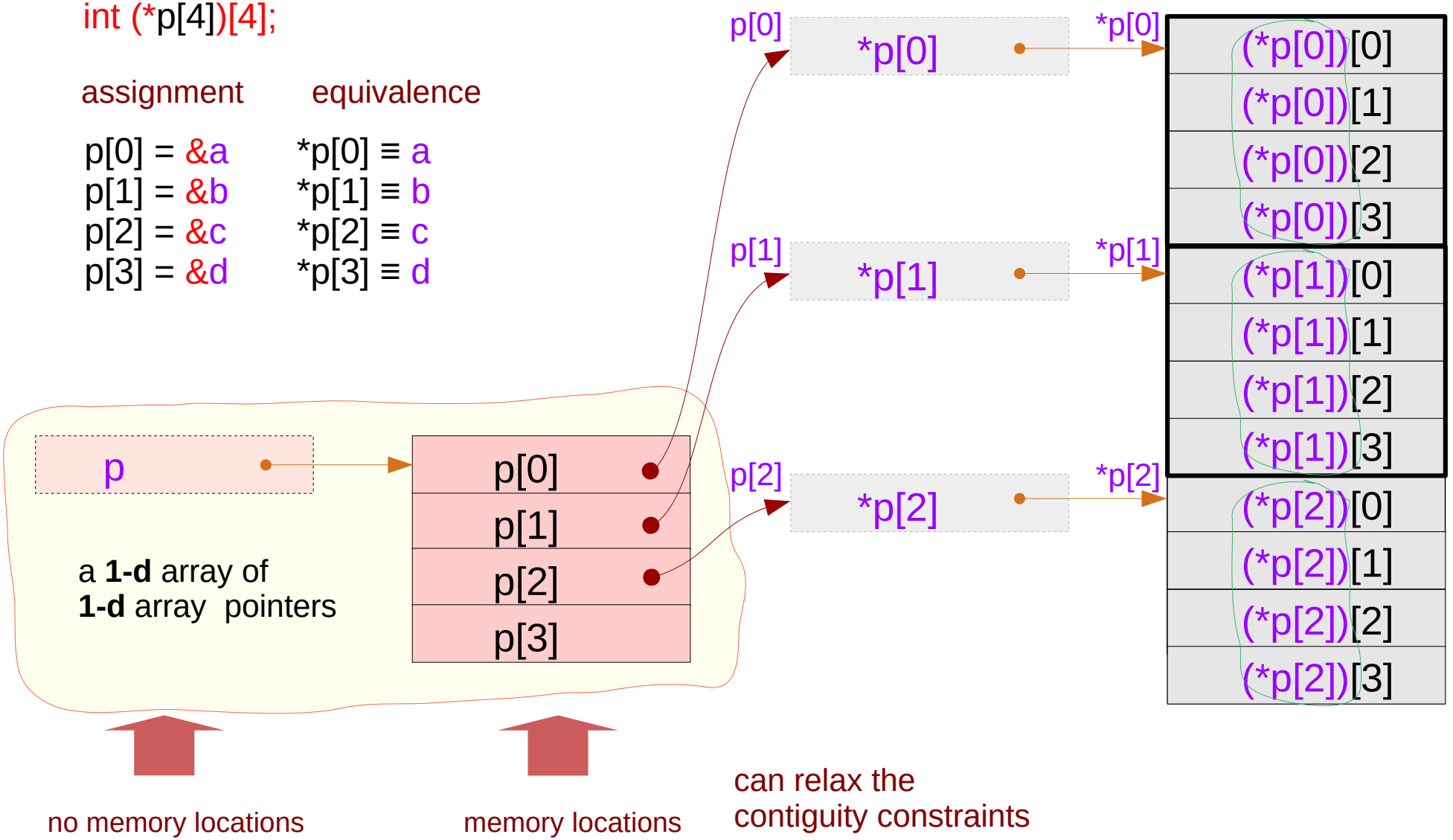
p[0] ≡ *(p+0) ≡ a
p[1] ≡ *(p+1) ≡ b
p[2] ≡ *(p+2) ≡ c

An array **p** of array pointers

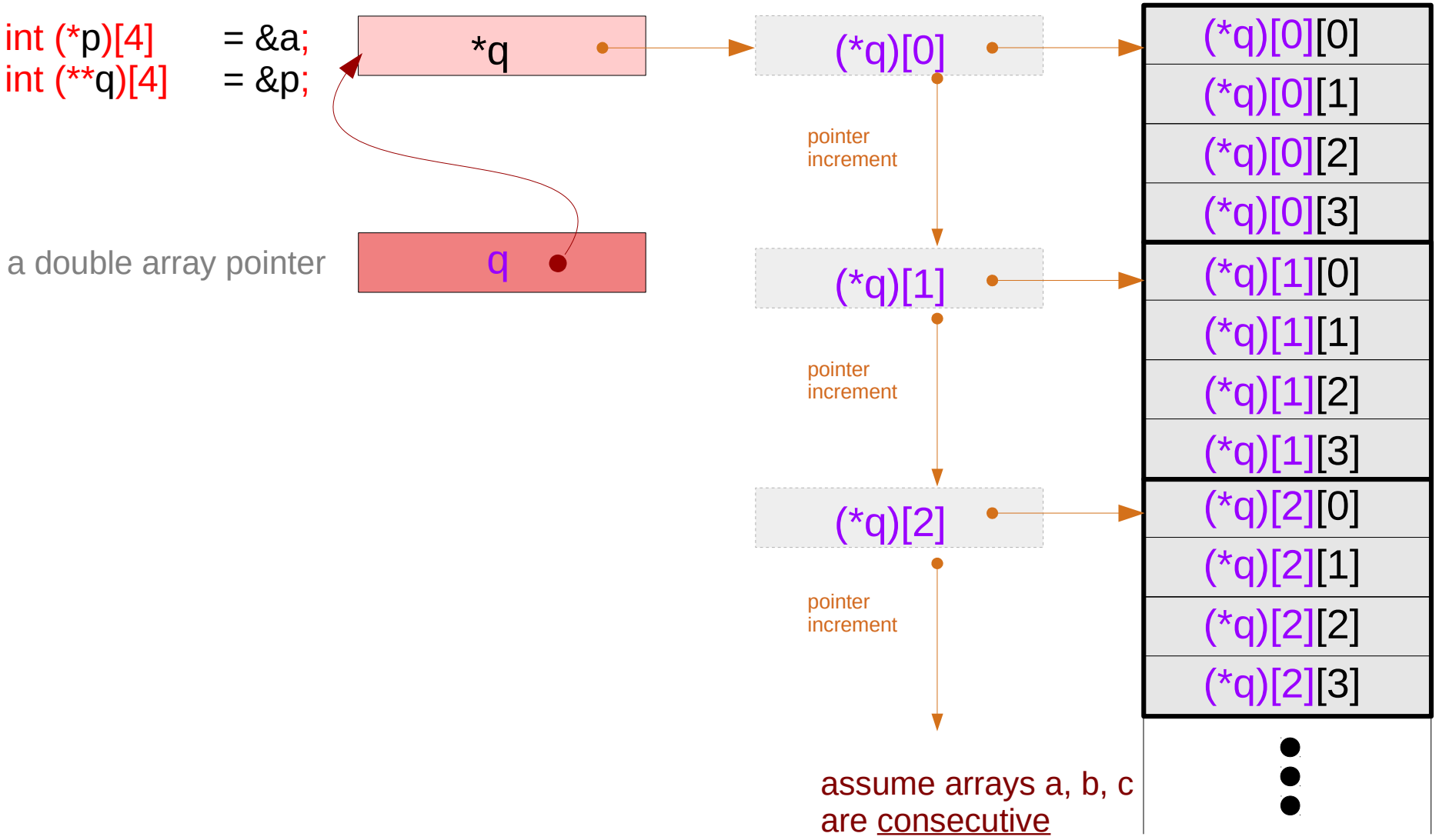
int (*p[4])[4];

assignment | equivalence

p[0] = &a | *p[0] ≡ a
p[1] = &b | *p[1] ≡ b
p[2] = &c | *p[2] ≡ c
p[3] = &d | *p[3] ≡ d



a double array pointer **q**



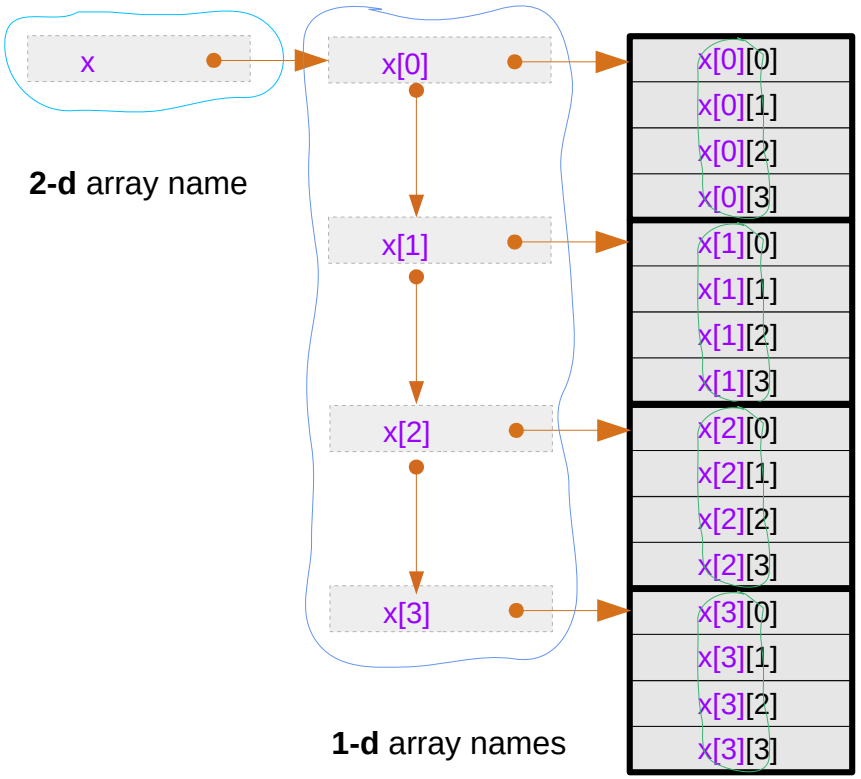
2-d array access using various pointers

- a **1-d** array of integer pointers **int *p[4];**
- a **1-d** array pointer **int (*p)[4];**
- a **2-d** array pointer **int (*p)[4][4];**
- an array of **1-d** array pointers **int (*p[4])[4];**
- a **2-d** array of integer pointers **int *p[4][4];**

Accessing a 2-d array using pointers

int x[4][4];

A **2-d** array



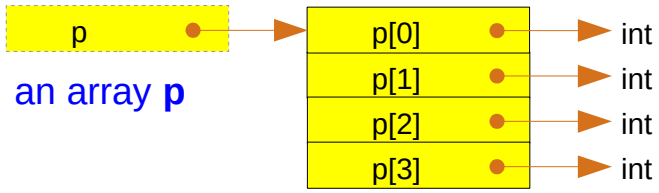
2-d array name

1-d array names

Array **p** and Pointer **p**

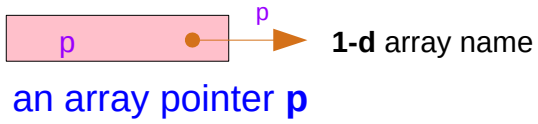
A **1-d** array **p** of integer pointers

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



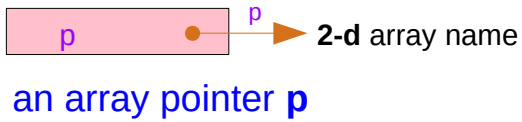
A **1-d** array pointer **p**

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



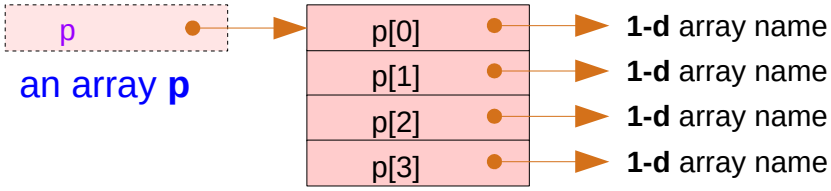
A **2-d** array pointer **p**

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



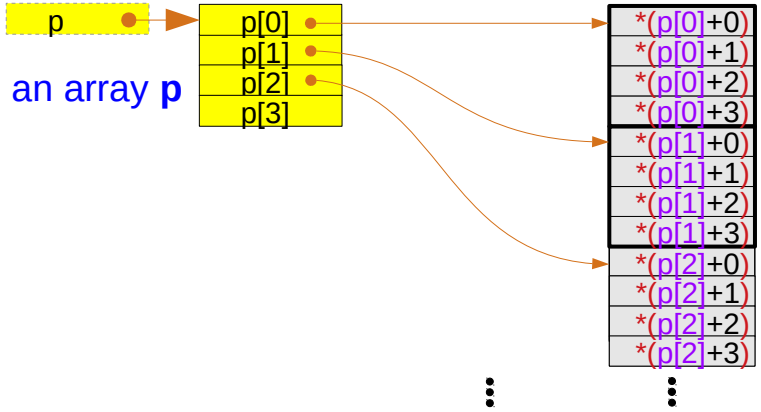
An array **p** of **1-d** array pointers

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

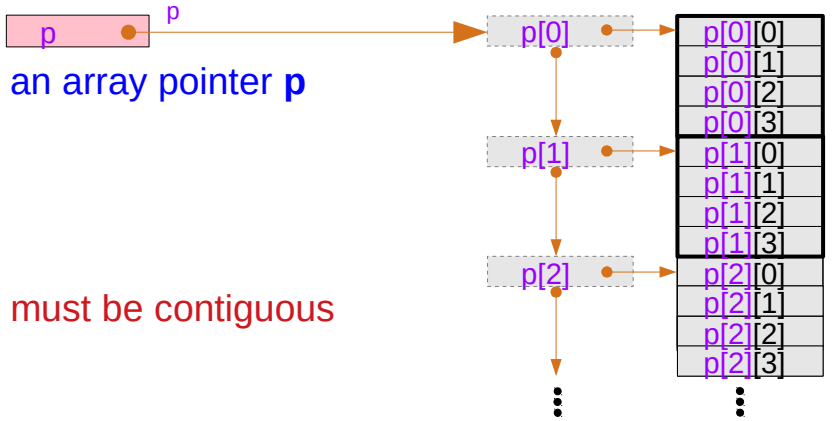


Arrays **p** and Pointers **p** for accessing a **2-d** array

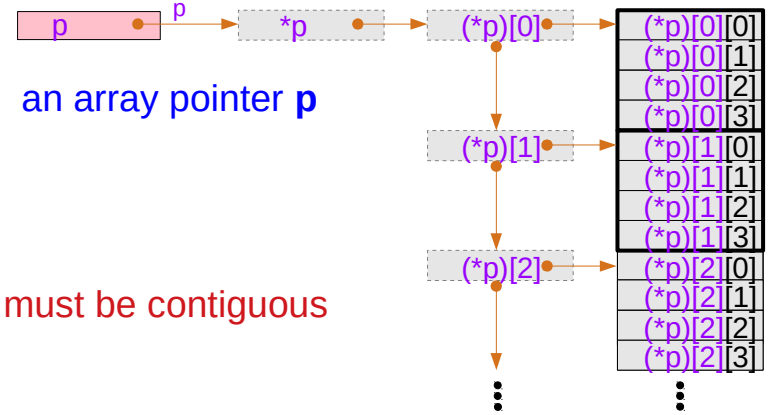
A **1-d** array **p** of integer pointers



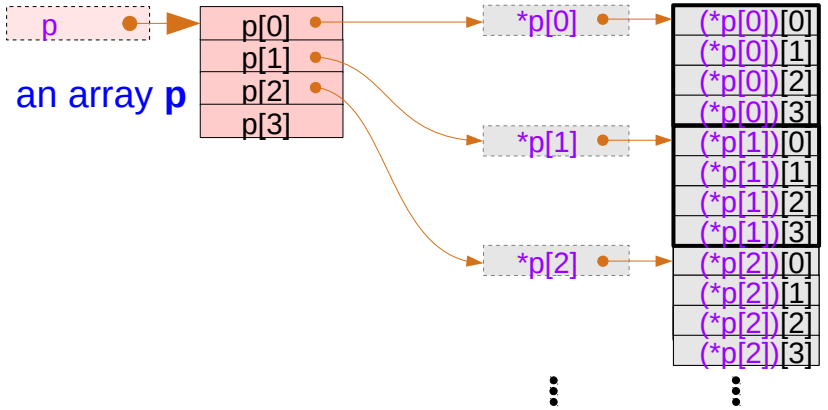
A **1-d** array pointer **p**



A **2-d** array pointer **p**



An array **p** of **1-d** array pointers

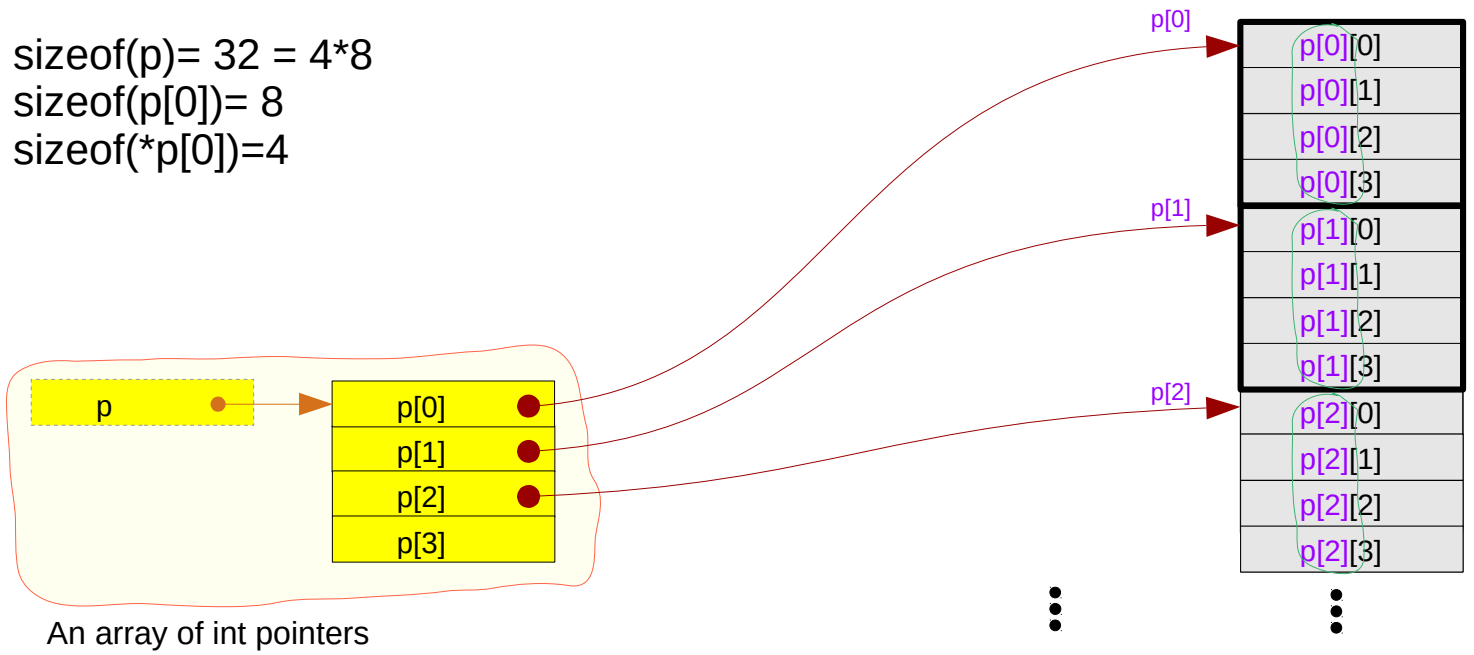


Using a **1-d** array of integer pointer **: int *p[4]**

int *p[4]; **Type Definition**

*(p[m]+n) ≡ p[m][n] **Access Method**

sizeof(p)= 32 = 4*8
sizeof(p[0])= 8
sizeof(*p[0])=4



assignment

p[0]=x[0]
p[1]=x[1]
p[2]=x[2]
p[3]=x[3]

equivalence

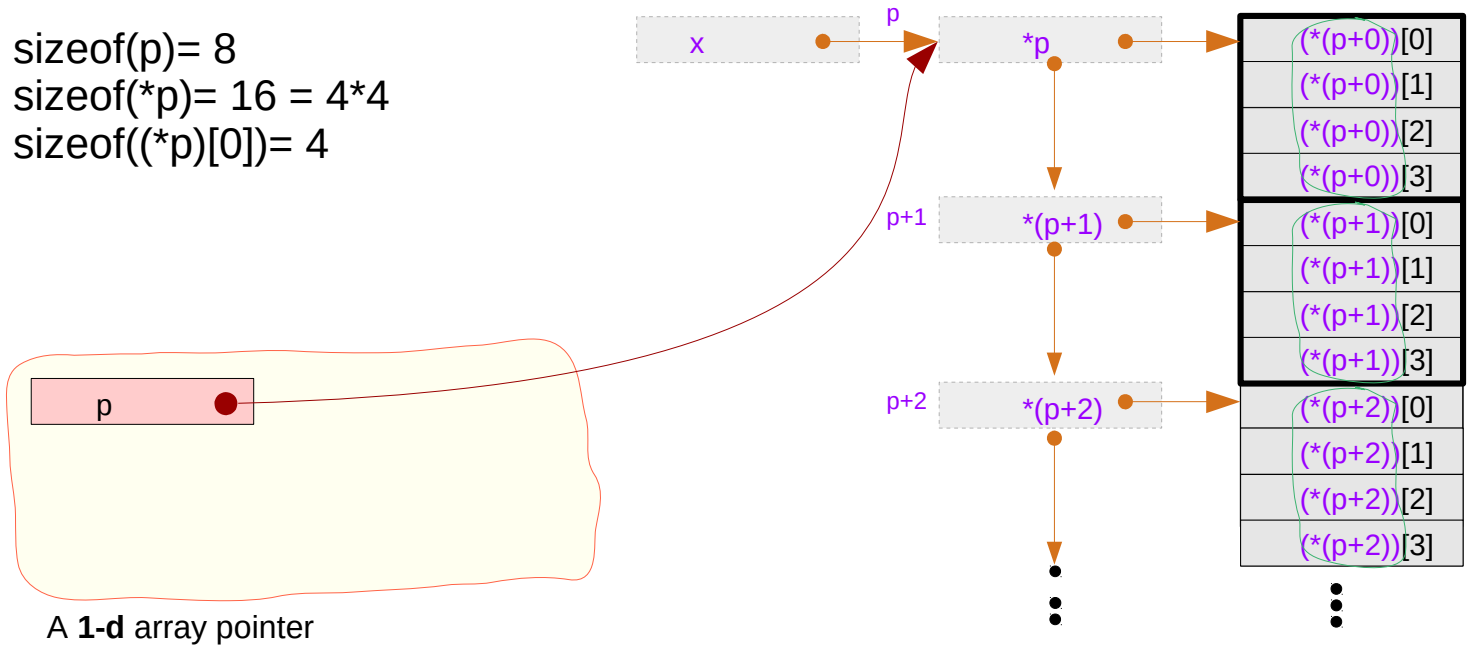
p=x

Using a **1-d** array pointer : **int (*p)[4]**

int (*p)[4]; **Type Definition**

(*(p+m))[n]; ≡ p[m][n]; **Access Method**

sizeof(p)= 8
sizeof(*p)= 16 = 4*4
sizeof((*p)[0])= 4



A **1-d** array pointer

assignment

p=x

equivalence

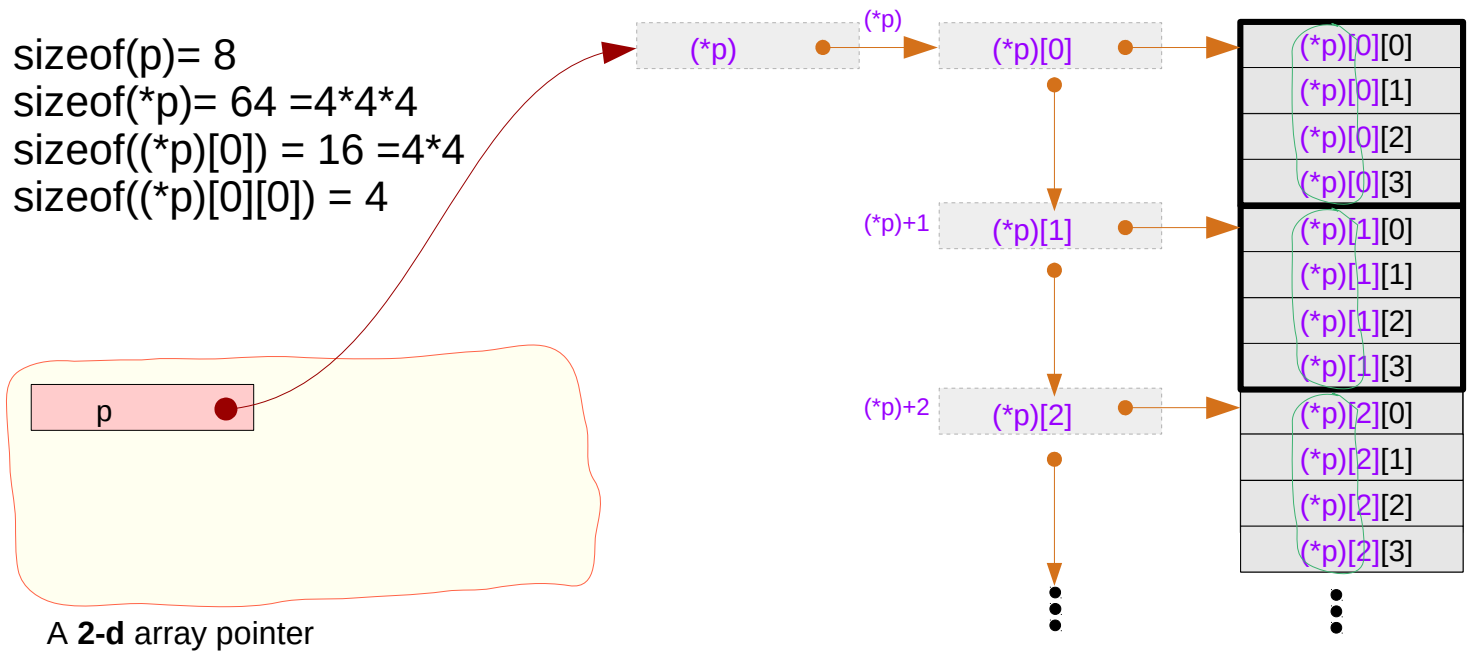
p[0]=x[0]
p[1]=x[1]
p[2]=x[2]
p[3]=x[3]

Using a **2-d** array pointer : **int (*p)[4][4]**

int (*p)[4][4]; **Type Definition**

(*p)[m][n]; **Access Method**

sizeof(p)= 8
sizeof(*p)= 64 =4*4*4
sizeof((*p)[0]) = 16 =4*4
sizeof((*p)[0][0]) = 4



A **2-d** array pointer

assignment

p=&x

equivalence

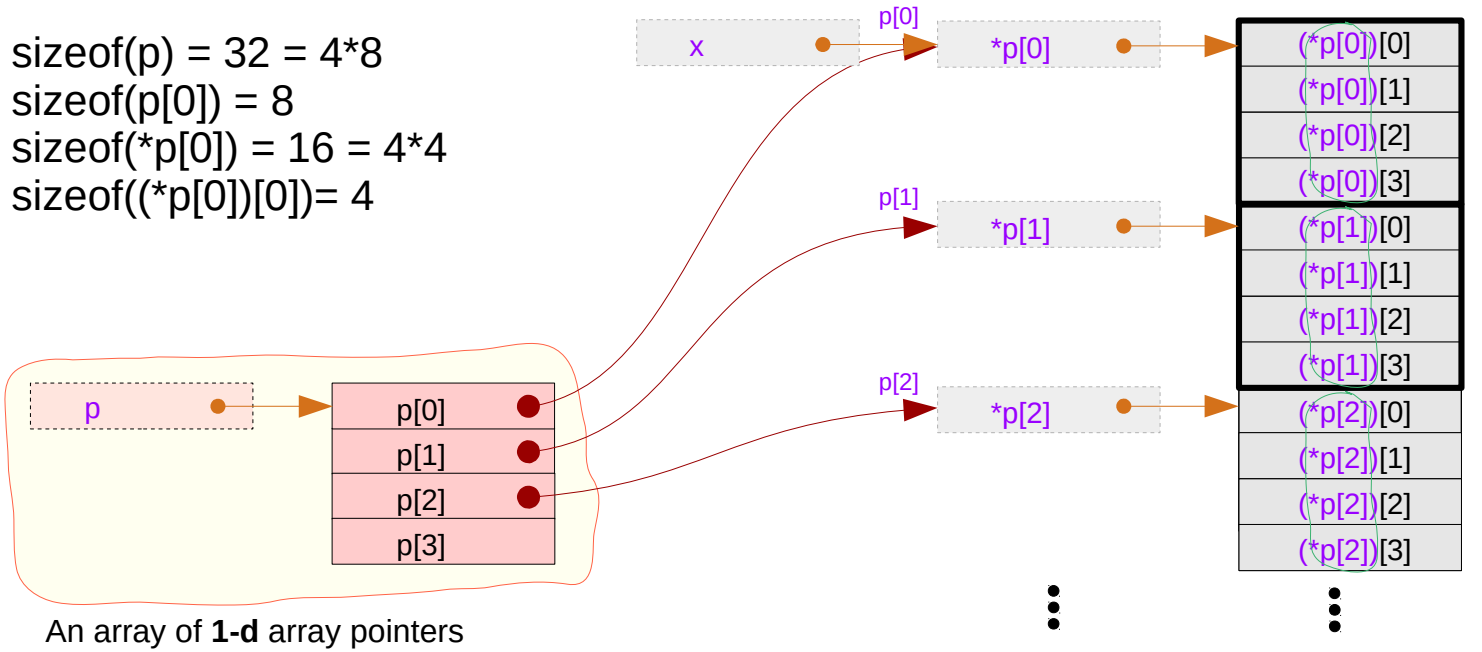
(*p)[0]=x[0]
(*p)[1]=x[1]
(*p)[2]=x[2]
(*p)[3]=x[3]

Using an array of **1-d** array pointers : **int (*p[4])[4]**

int (*p[4])[4]; **Type Definition**

(*p[m])[n]; **Access Method**

sizeof(p) = 32 = 4*8
sizeof(p[0]) = 8
sizeof(*p[0]) = 16 = 4*4
sizeof((*p[0])[0])= 4



An array of **1-d** array pointers

assignment

p[0]=&x[0]
p[1]=&x[1]
p[2]=&x[2]
p[3]=&x[3]

equivalence

*p=x

Access methods in a type view

A **1-d** array **p** of integer pointers

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

```
p[m][n]
```

A **1-d** array pointer **p**

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

```
p[m][n]
```

A **2-d** array pointer **p**

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

```
(*p)[m][n]
```

An array **p** of **1-d** array pointers

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

```
(*p[m])[n]
```

Sizes of (*p[m]), (*p[m])[n], ((*p)[m]), ((*p)[m])[n]

A **1-d** array **p** of integer pointers

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

sizeof(p)=32 (4*8) an array of int pointers
sizeof(p[0])=8 an int pointer
sizeof(*p[0])=4 an integer

An array **p**

A **1-d** array pointer **p**

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

sizeof(p)=8 a 1-d array pointer
sizeof(*p)=16 (4*4) a 1-d array
sizeof((*p)[0])=4 an integer

An array pointer **p**

A **2-d** array pointer **p**

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

sizeof(p)=8 a 2-d array pointer
sizeof(*p)=64 a 2-d array (4*4*4)
sizeof((*p)[0])=16 a 1-d array (4*4)
sizeof((*p)[0][0])=4 an integer

An array pointer **p**

An array **p** of **1-d** array pointers

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

sizeof(p)=32 (4*8) a 1-d array of 1-d array pointers
sizeof(p[0])=8 a 1-d array pointer
sizeof(*p[0])=16 (4*4) a 1-d array
sizeof((*p[0])[0])=4 an integer

An array **p**

Initialization

A **1-d** array **p** of integer pointers

```
int *p[4] = {x[0], x[1], x[2], x[3]};

p[0] = x[0];        // an integer pointer (int *)
p[1] = x[1];        // an integer pointer (int *)
p[2] = x[2];        // an integer pointer (int *)
p[3] = x[3];        // an integer pointer (int *)
```

An array **p**

A **1-d** array pointer **p**

```
int (*p)[4] = &x[0];

p = &x[0];          // a 1-d array pointer (int (*)[4])

p = x;              // a 1-d array pointer (int (*)[4])
```

An array pointer **p**

A **2-d** array pointer **p**

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

p = &x;             // a 2-d array pointer (int (*)[4][4])
```

An array pointer **p**

An array **p** of **1-d** array pointers

```
int (*p[4])[4] = {&x[0], &x[1], &x[2], &x[3]};

p[0] = &x[0];       // a 1-d array pointer (int (*)[4])
p[1] = &x[1];       // a 1-d array pointer (int (*)[4])
p[2] = &x[2];       // a 1-d array pointer (int (*)[4])
p[3] = &x[3];       // a 1-d array pointer (int (*)[4])
```

An array **p**

Equivalent access methods in a type view

A **1-d** array **p** of integer pointers

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

*(p[m]+n)

p[m][n] = (p[m])[n]

A **1-d** array pointer **p**

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

(*(p+m))[n]

p[m][n] = (p[m])[n]

A **2-d** array pointer **p**

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

(*p)[m][n] = ((*p)[m])[n]

An array **p** of **1-d** array pointers

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

(*p[m])[n] = (*(p[m]))[n]

(*p)[m][n] = ((*p)[m])[n]

Types in an access method view

p[m][n] an integer

- int *p[M]; an integer pointer
- int (*p)[N] a 1-d array pointer

(*p)[m][n] an integer

- int (*p)[M][N]; a 2-d array pointer
- int (*p[M])[N]; an array of 1-d array pointers

(*p[m])[n] an integer

- int (*p[M])[N]; an array of 1-d array pointers

*p[m][n] an integer

- int *p[M][N]; a 2-d array of integer pointers

2-d array access using array pointers

- **2-d** array pointer — **int (*p)[4][4];** **(*p)[i][j];**
- array of **1-d** array pointers — **int (*p[4])[4];** **(*p[i])[j];**

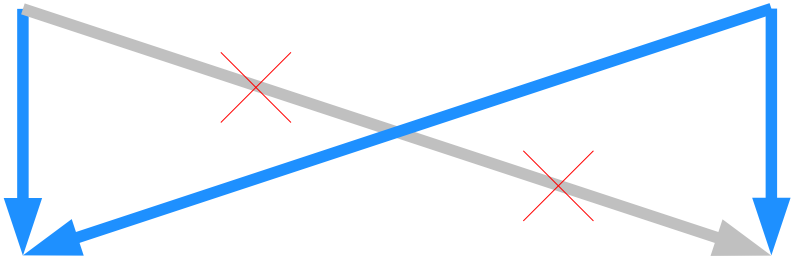
2-d array access using array pointers

A **2-d** array pointers

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

An array of **1-d** array pointers

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



(*p)[m][n];

access method for
a **2-d** array pointers

(*p[m**])**[n];

Access methods for
an array of **1-d** array pointers

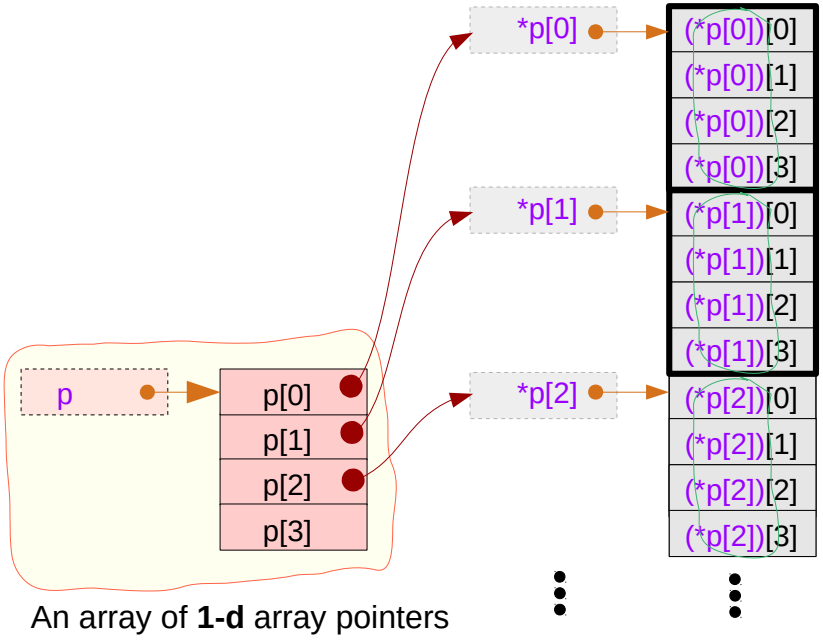
int (*p[4])[4] and (*p)[i][j] : OK

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

assignment	equivalence
p[0]=&x[0]	*p[0]=x[0]
p[1]=&x[1]	*p[1]=x[1]
p[2]=&x[2]	*p[2]=x[2]
p[3]=&x[3]	*p[3]=x[3]

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

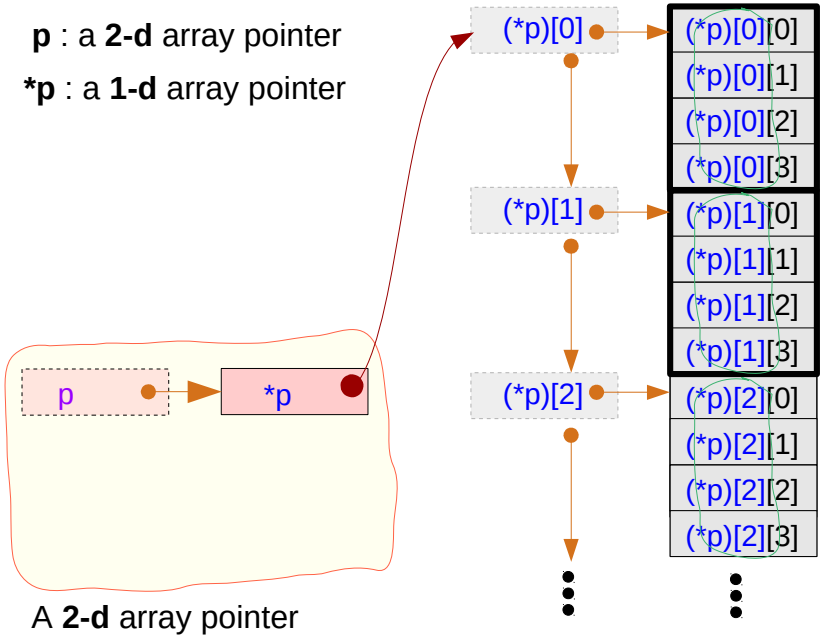
assignment	equivalence
p=&x	(*p)[0]=x[0]
	(*p)[1]=x[1]
	(*p)[2]=x[2]
	(*p)[3]=x[3]



An array of **1-d** array pointers

p : a **2-d** array pointer

***p** : a **1-d** array pointer



A **2-d** array pointer

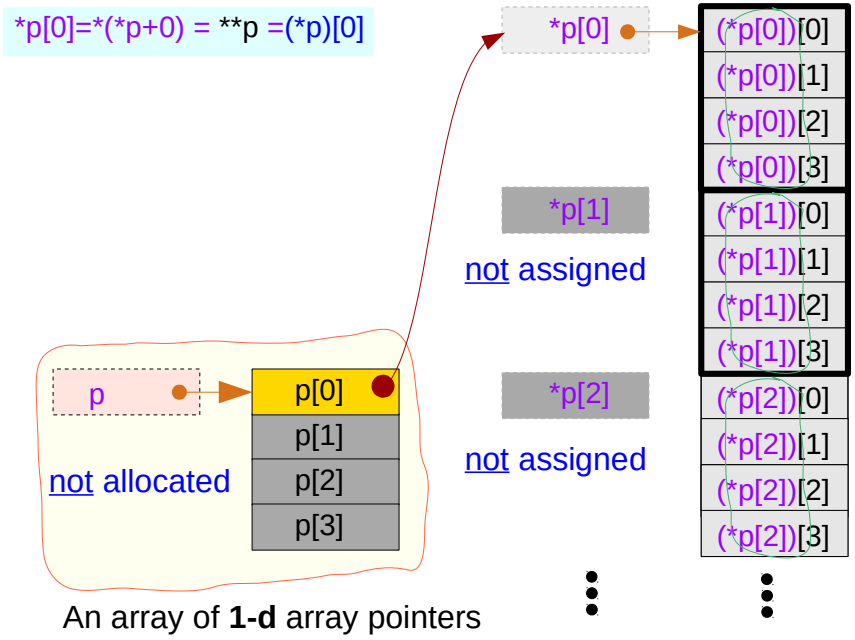
int (*p)[4][4] and (*p[i])[j] : not OK

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

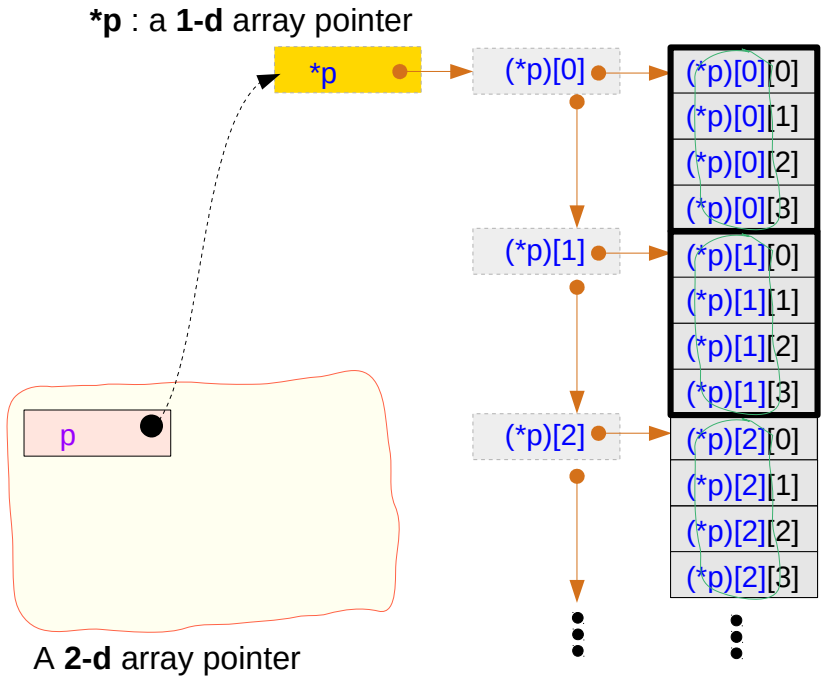
assignment	equivalence
p[0]=&x[0]	*p[0]=x[0]
~~p[1]=&x[1]~~	~~*p[1]=x[1]~~
~~p[2]=&x[2]~~	~~*p[2]=x[2]~~
~~p[3]=&x[3]~~	~~*p[3]=x[3]~~

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

assignment	equivalence
p=&x	(*p)[0]=x[0]
	(*p)[1]=x[1]
	(*p)[2]=x[2]
	(*p)[3]=x[3]



An array of **1-d** array pointers

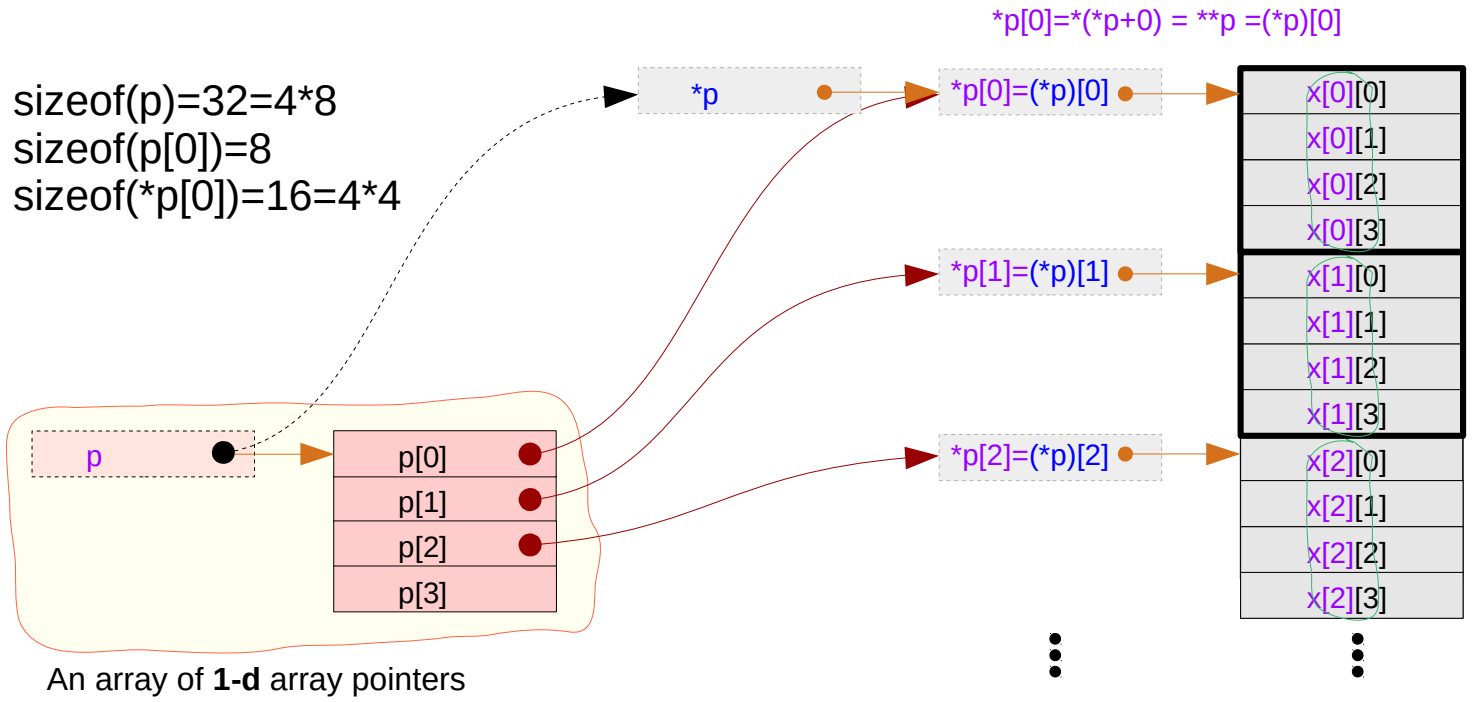


A **2-d** array pointer

int (*p[4])[4] and accessing a 2-d array

int (*p)[4][4]; → **(*p)[m][n];**

int (*p[4])[4]; → **(*p[m])[n];**



assignment

*(p[0])=x[0]
*(p[1])=x[1]
*(p[2])=x[2]
*(p[3])=x[3]

equivalence

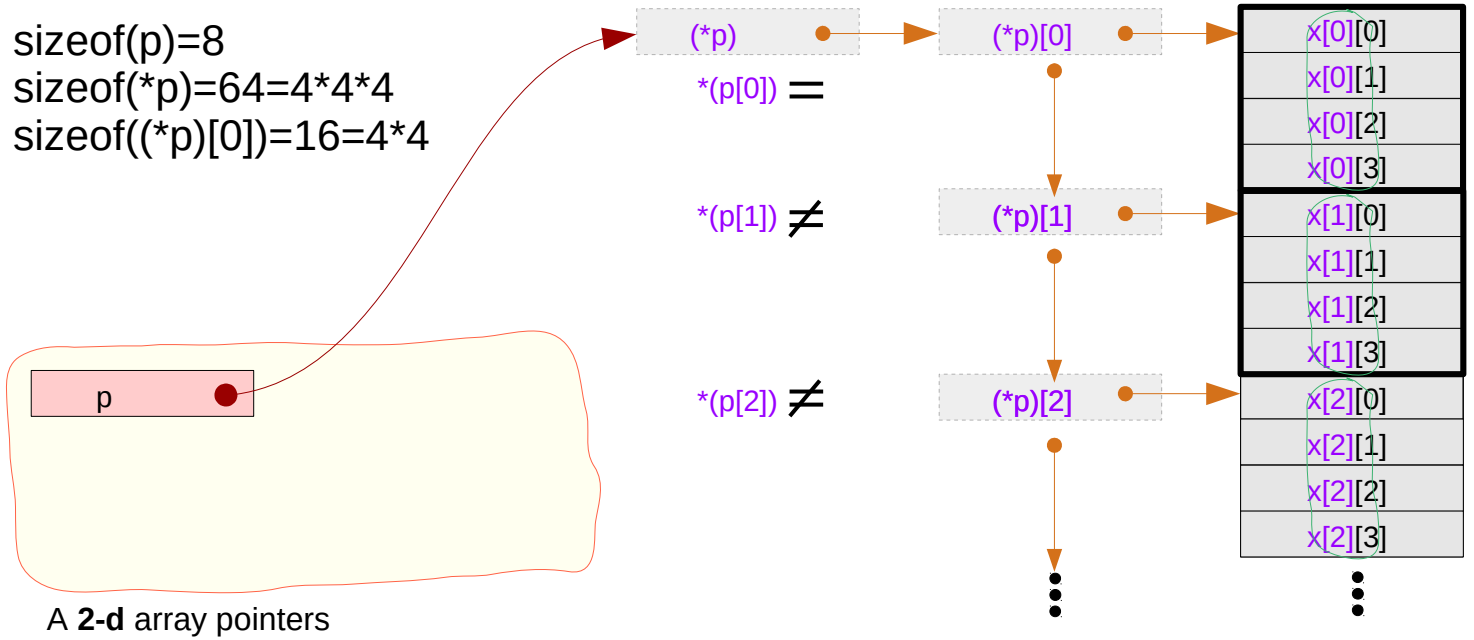
*(p[0])=(*p)[0]
*(p[1])=(*p)[1]
*(p[2])=(*p)[2]
*(p[3])=(*p)[3]

int (*p)[4][4] and accessing a 2-d array

int (*p)[4][4]; → **(*p)[m][n];**

int (*p[4])[4]; → **(*p[m])[n];**

sizeof(p)=8
sizeof(*p)=64=4*4*4
sizeof((*p)[0])=16=4*4



A **2-d** array pointers

int (*p[4])[4] and equivalence relations

int (*p[4])[4]; (*p[m])[n]

assignment	equivalence
p[0]=&x[0]	*p[0]=x[0]
p[1]=&x[1]	*p[1]=x[1]
p[2]=&x[2]	*p[2]=x[2]
p[3]=&x[3]	*p[3]=x[3]

int (*p)[4][4]; (*p)[m][n]

assignment	equivalence
p=&x	(*p)[0]=x[0]
	(*p)[1]=x[1]
	(*p)[2]=x[2]
	(*p)[3]=x[3]

***p** : a **1-d** array pointer

(p[0])=(*(p+0))	x[0]	*((*p)+0) =(*p)[0]
(p[1])=(*(p+1))	x[1]	*((*p)+1) =(*p)[1]
(p[2])=(*(p+2))	x[2]	*((*p)+2) =(*p)[2]
(p[3])=(*(p+3))	x[3]	*((*p)+3) =(*p)[3]

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

int (*p)[4][4] and equivalence relation

int (*p[4])[4]; (*p[m])[n]

assignment	equivalence
p[0]=&x[0]	*(p[0])=x[0]
p[1]=&x[1]	*(p[1])=x[1]
p[2]=&x[2]	*(p[2])=x[2]
p[3]=&x[3]	*(p[3])=x[3]

int (*p)[4][4]; (*p)[m][n]

assignment	equivalence
p=&x	(*p)[0]=x[0]
	(*p)[1]=x[1]
	(*p)[2]=x[2]
	(*p)[3]=x[3]

***p** : a **1-d** array pointer

(p[0])=(*(p+0))	x[0]	x[0]	*((*p)+0) =(*p)[0]
(p[1])=(*(p+1))	–	x[1]	*((*p)+1) =(*p)[1]
(p[2])=(*(p+2))	–	x[2]	*((*p)+2) =(*p)[2]
(p[3])=(*(p+3))	–	x[3]	*((*p)+3) =(*p)[3]

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