Array Pointers (1A)

Young Won Lim
9/23/23

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

Young Won Lim
9/23/23

Assumption

assume that

value(c) returns the hexadecimal number
that is obtained by printf("%p", c),
when the variable c contains
an address as its value

```
#include <stdio.h>
int main(void) {
  int c[3] ;
  printf ("c= %p \n", &c);
}
```

```
c= 0x7fffd923487c
```

type(c) can be determined
by the warning messsage of printf("%d", c),
when the variable c contains
an address as its value

```
#include <stdio.h>
int main(void) {
  int c[3] ;
  printf ("c= %d \n", &c);
}
```

```
t.c: In function 'main':
t.c:5:16: warning: format '%d' expects argument of type 'int',
but argument 2 has type 'int (*)[3]' [-Wformat=]
   printf ("c= %d \n", &c);
```

Array Pointers v.s. Pointer Arrays

Array pointers and pointer arrays

1. array pointer p – a pointer to an array of int [4] type

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

2. pointer array x – an array of pointers of int * type

```
int   *x  [4] ;
```

Types of array pointer **p** and pointer arrays **y**

array pointer:
a pointer to an array

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

(*p) is an array with 4 elements
each element is an integer
p is a pointer to such an array

the type of **(*p)** : **int [4]**
the type of **p** : **int (*) [4]**

pointer array:
an array of pointers

```
int * y [4];
```

y is an array with 4 elements
each element is an integer pointer

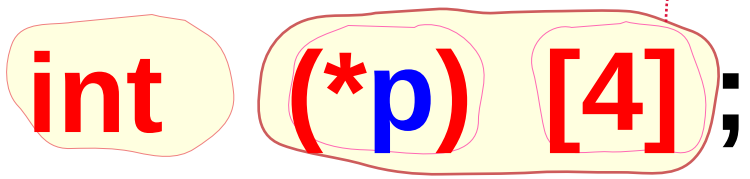
the type of **y** : **int * [4]**

[ ] has a higher priority than ***

(*p) must be grouped with **[4]**
y must be grouped with **[4]**

Types of array elements **(*p)[i]** and **y[i]**

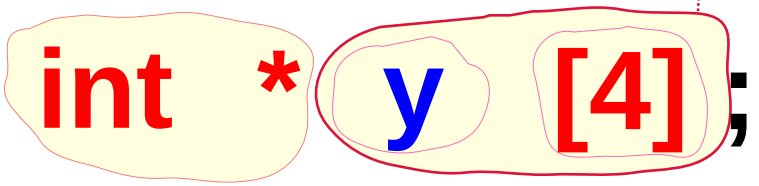
array pointer:
a pointer to an array



(*p)[i] in a statement

(*p) is an array with 4 elements
(*p)[0], (*p)[1], (*p)[2], (*p)[3]
the type of elements : **int**

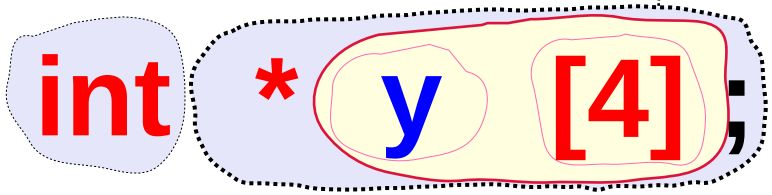
pointer array:
an array of pointers



y[i] in a statement

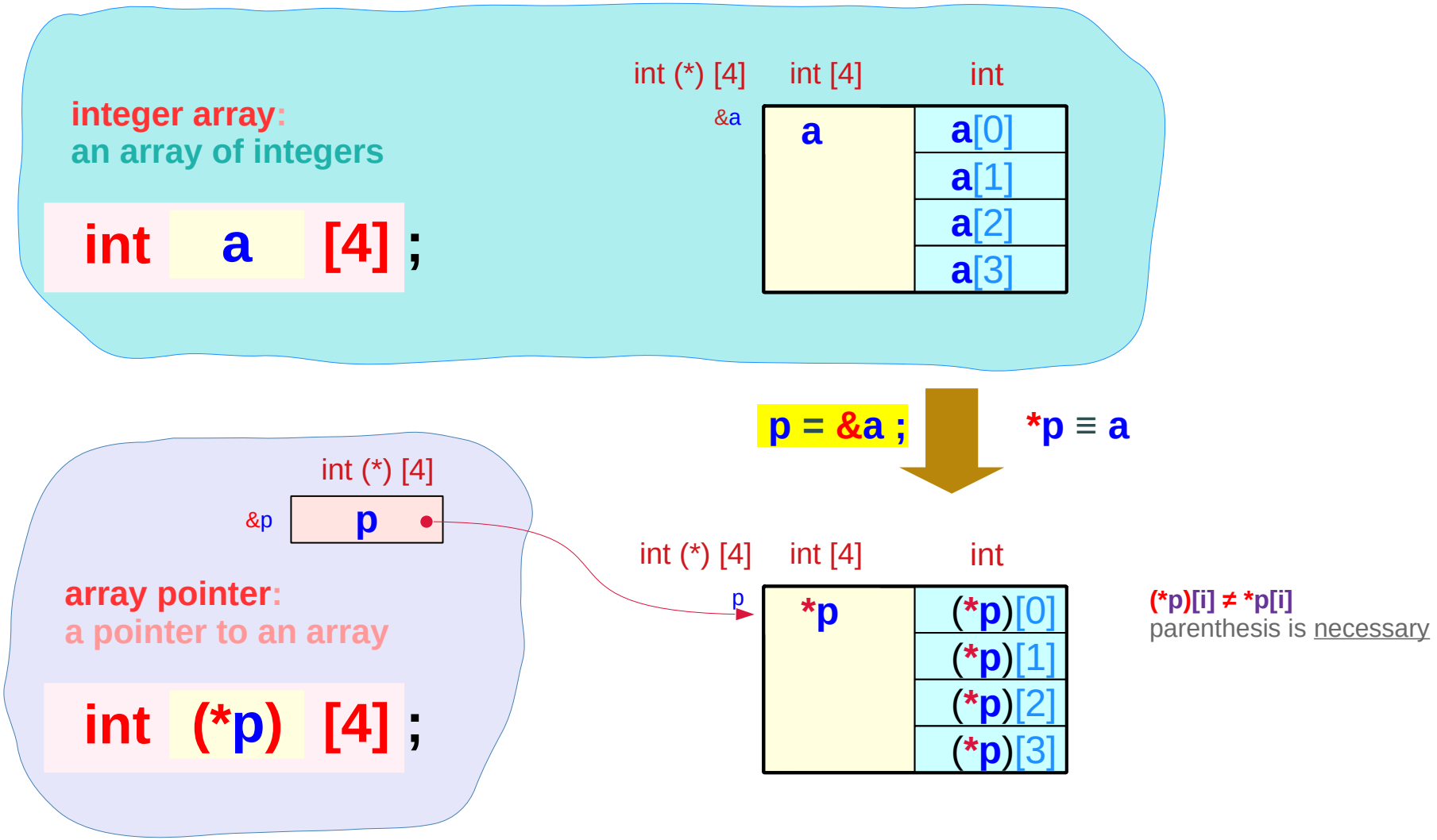
y is an array with 4 elements
y[0], y[1], y[2], y[3]
the type of elements : **int ***

***y[i]** in a statement



y is an array with 4 elements
***y[0], *y[1], *y[2], *y[3]**
the type of dereferenced elements : **int**

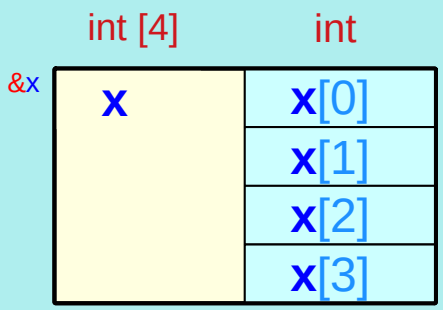
Array pointer **p**, array ***p**, integer **(*p)[i]**



Pointer array **y**, integer pointer **y[i]**, integer ***y[i]**

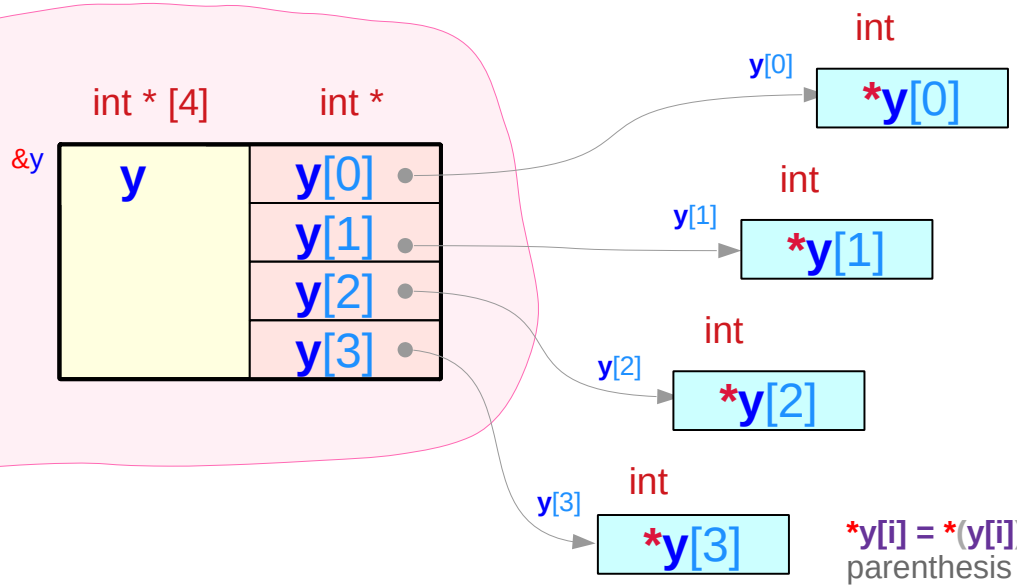
integer array:
an array of integers

```
int x [4] ;
```



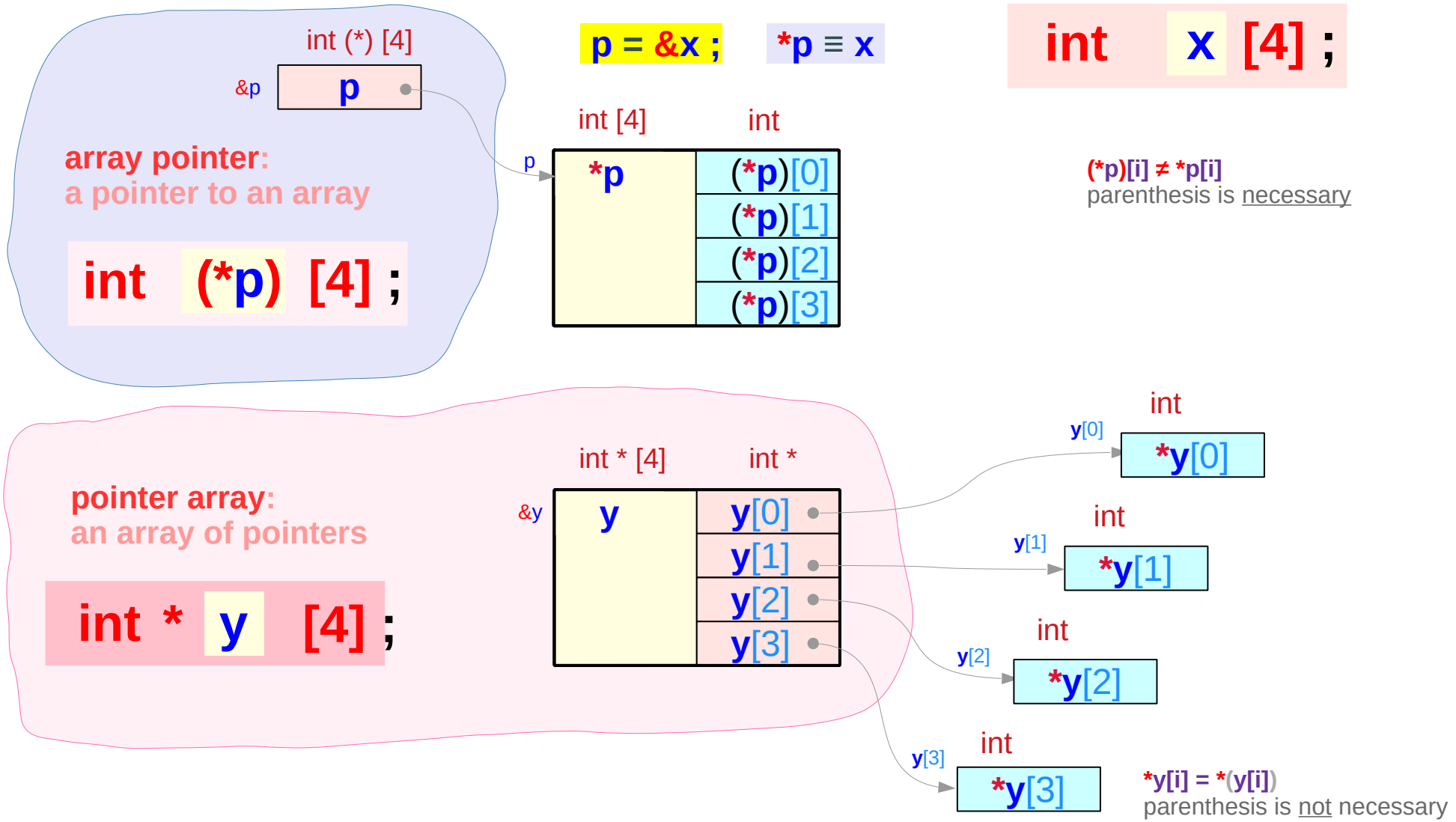
pointer array:
an array of pointers

```
int * y [4] ;
```

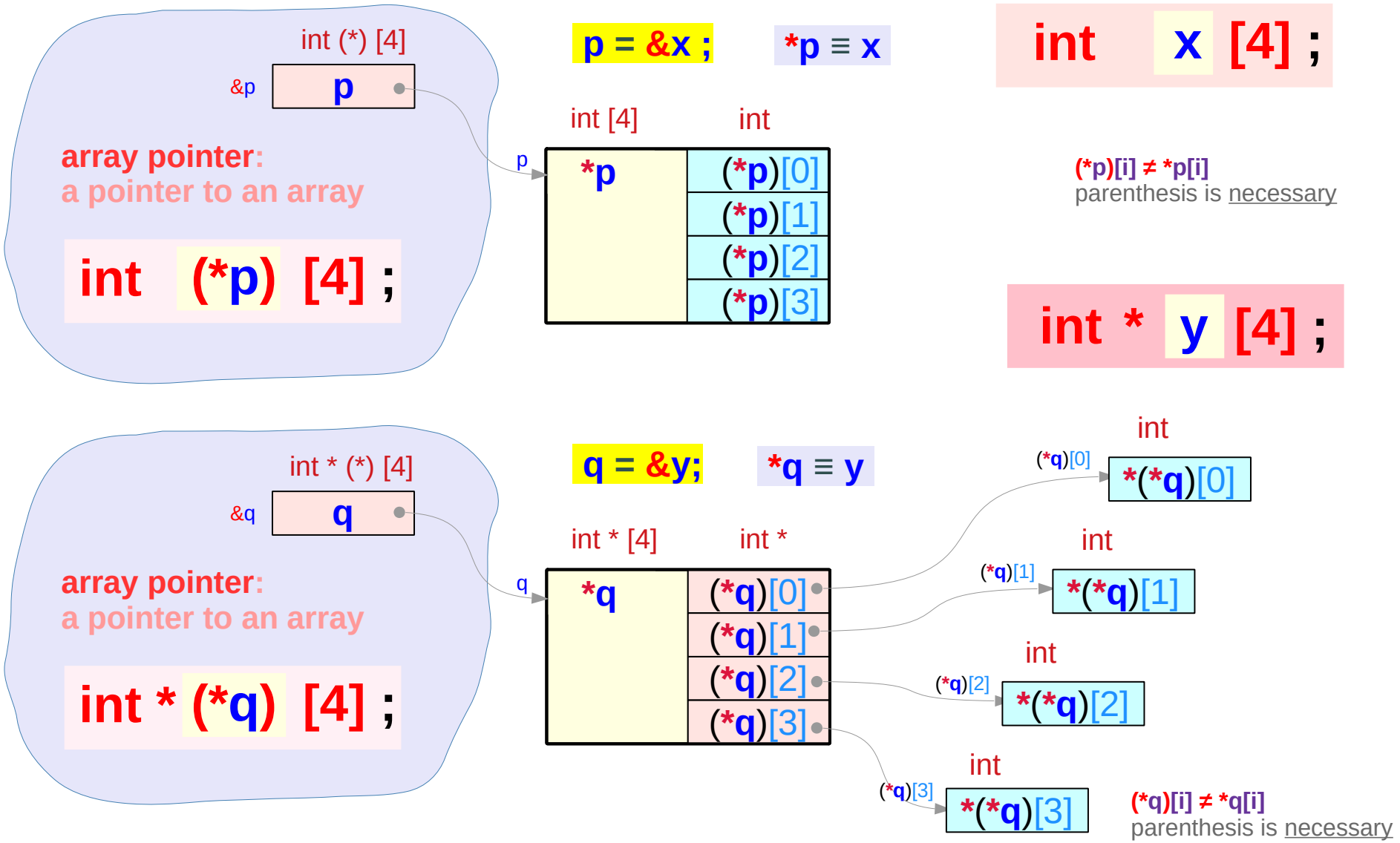


***y[i] = *(y[i])**
parenthesis is not necessary

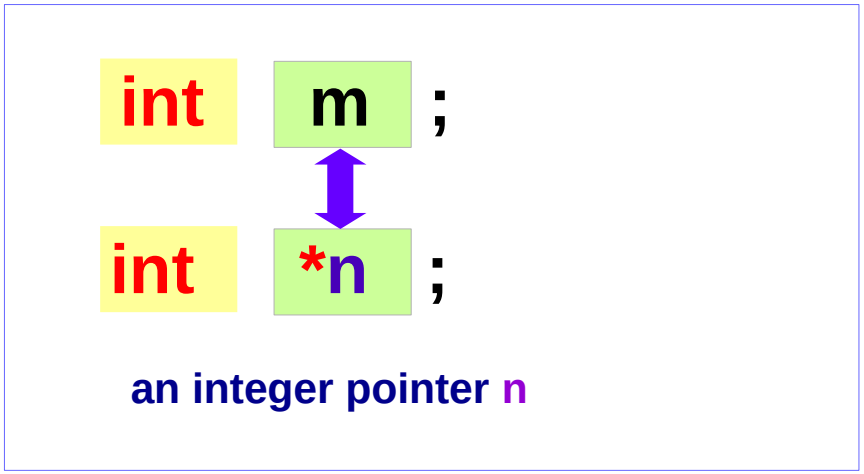
Array pointers **p** v.s. pointer arrays **y**



Array pointers **p** v.s. array pointer **q**



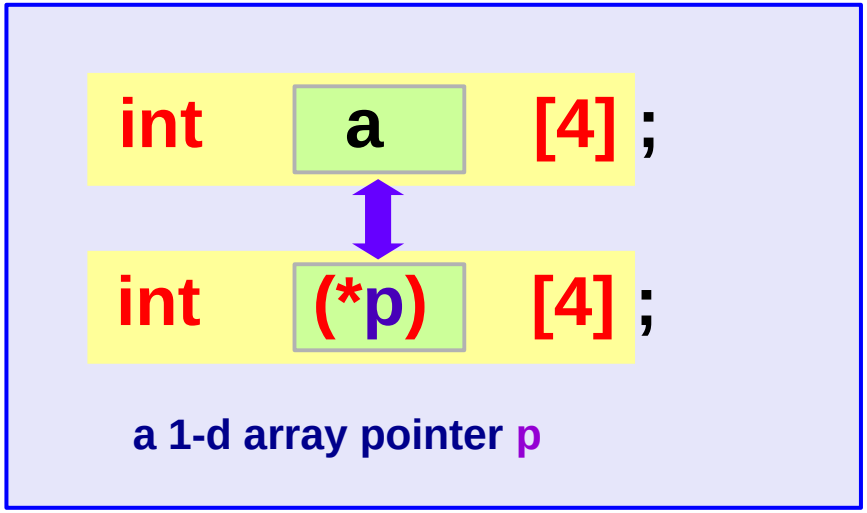
Correspondence of pointer dereference *n, *p, *fp

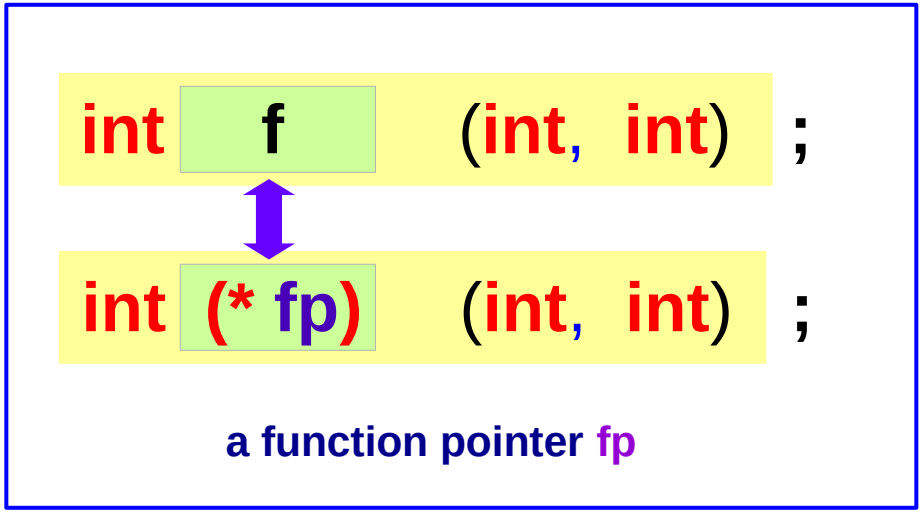


m and ***n** : an **integer** (**int**)

a and ***p** : a **1-d array** (**int [4]**)
with 4 integer elements

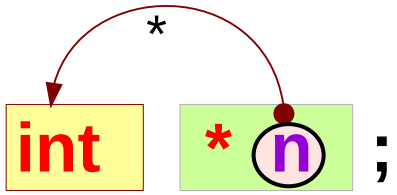
f and ***fp** : a **function** (**int (int, int)**)
taking two integers
returning an integer





Variables and pointed types

n points to a integer
***n** dereference of **n** (int)



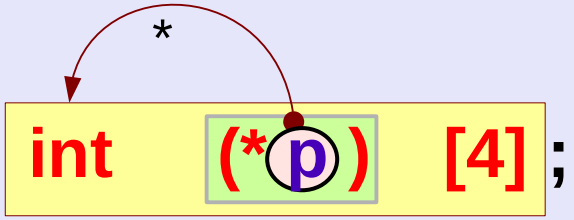
an integer pointer n int *

n points to an **integer**

p points to a **1-d array**
that has 4 integer elements

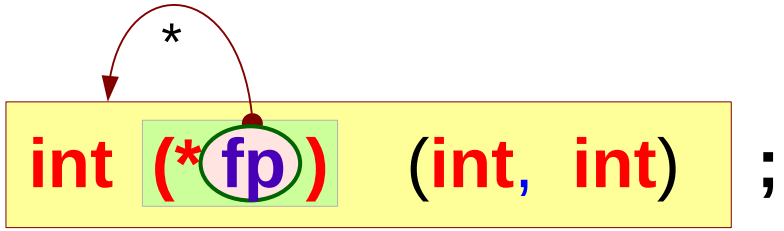
fp points to a **function**
that takes two integers
returns an integer

p points to a 1-d array
***p** dereference of **p** (int [4])



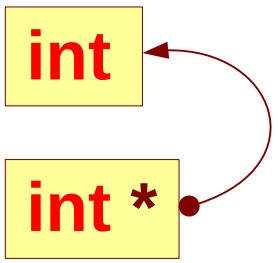
a 1-d array pointer p int (*) [4]

fp points to a function
***fp** dereference of **fp** (int (int, int))



a function pointer fp int (*) (int, int)

Referring and referenced types



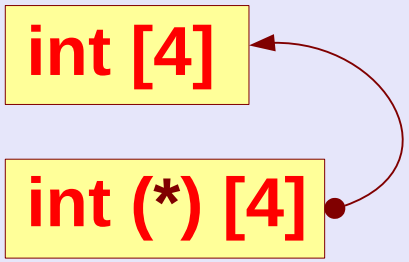
4 byte data

an integer pointer

int * type variable
points to an **int** type data

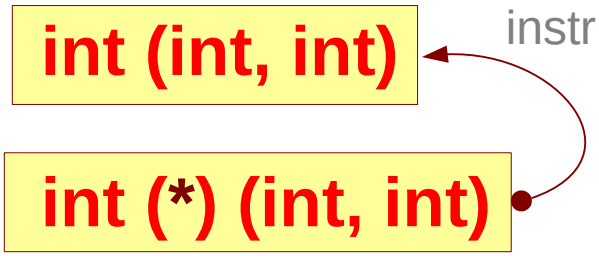
int (*) [4] type variable
points to an **int [4]** type data

int (*) (int, int) type variable
points to an **int (int, int)** type function



4*4 byte data

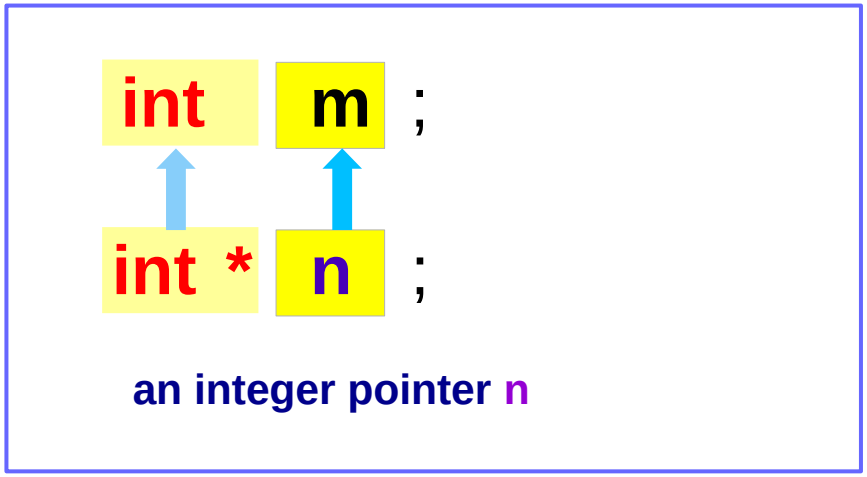
a 1-d array pointer

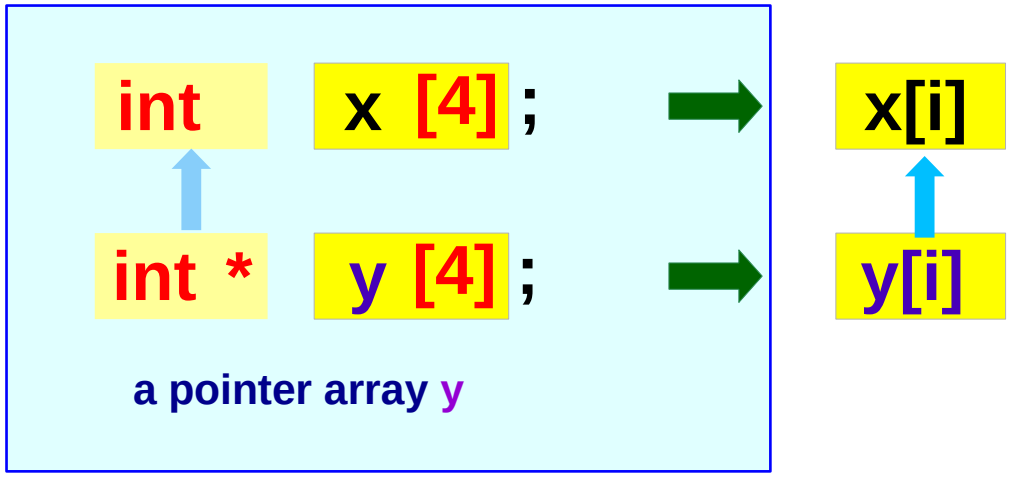


instructions

a function pointer

Correspondence of pointer reference **n**, **y[i]**





m : an **integer**

n : an **integer pointer**

n can point to **m**

n = &m ;

***n ≡ m**

x[i] : an **integer**

y[i] : an **integer pointer**

y[i] can point to **x[i]**

y[i] = &x[i] ;

***y[i] ≡ x[i]**

[ ] has a higher priority than ***

***y[i] = *(y[i])** unnecessary parenthesis

Element types of a pointer array **y**

int * y [4] ;

[ ] has a higher priority than ***

y must be grouped with **[4]**

int x [4] ; ➡ **x[i]**

x is a **1-d array** with 4 elements

each element **x[i]**, **i=0**, **1**, **2**, **3**
has the type of an **integer** (**int**)

int * y [4] ; ➡ **y[i]**

integer pointer

y is a **1-d array** with 4 elements

each element **y[i]**, **i=0**, **1**, **2**, **3**
has the type of an **integer pointer**
(**int ***)

int *y [4] ; ➡ ***y[i]**

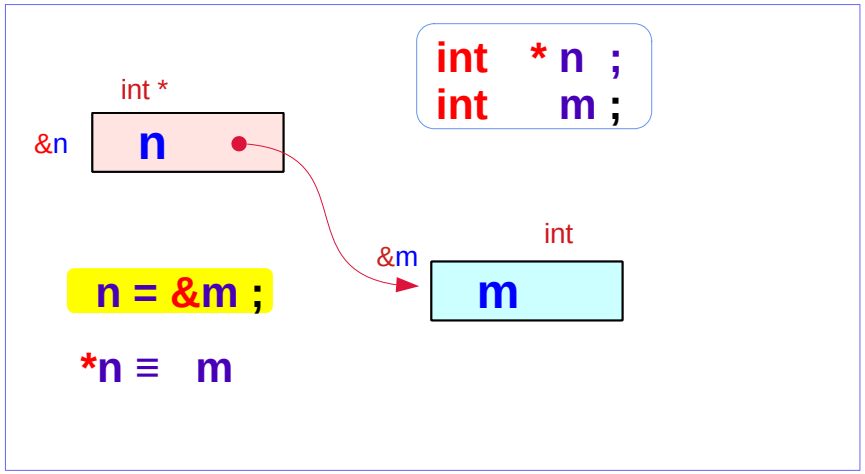
integer

y is a **1-d array** with 4 elements

the dereference of each element
***y[i]**, **i=0**, **1**, **2**, **3** is an **integer**

here, ***y[i]** = ***(y[i])**

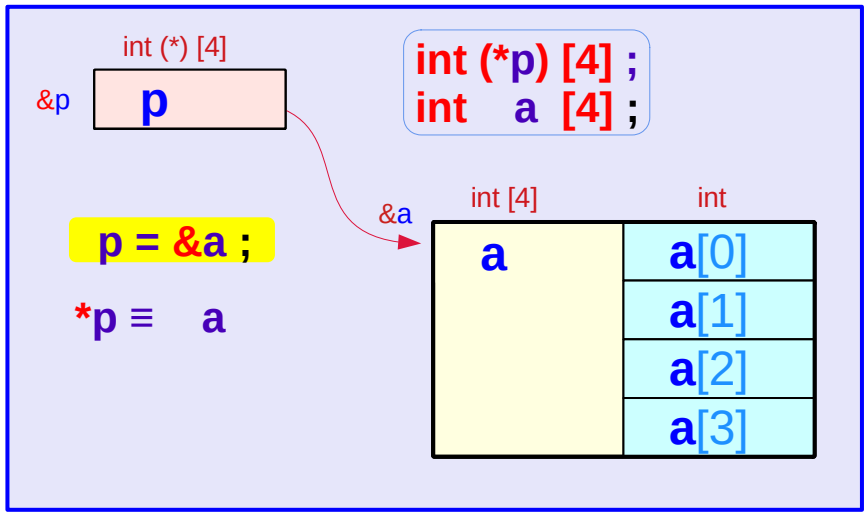
Assigning pointer variables **n**, **p**, **fp**

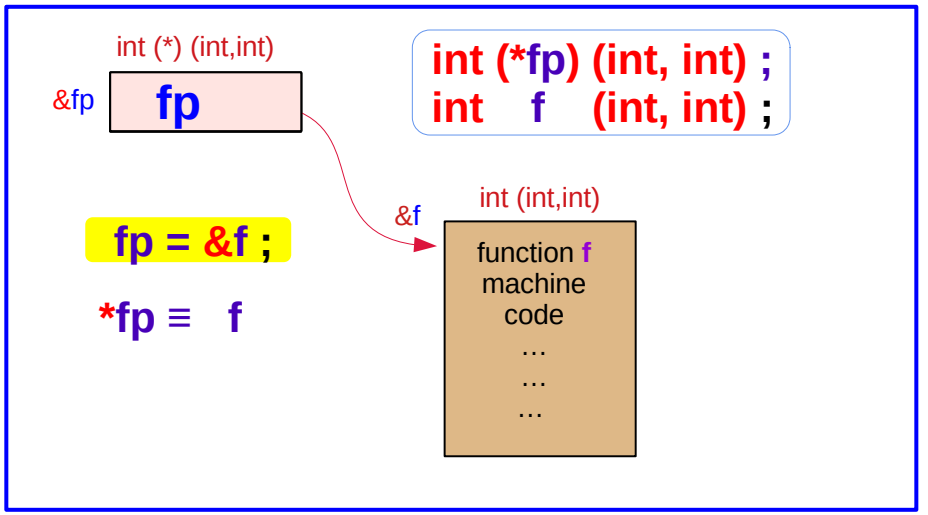


n : a **pointer to** an **integer** **int ***

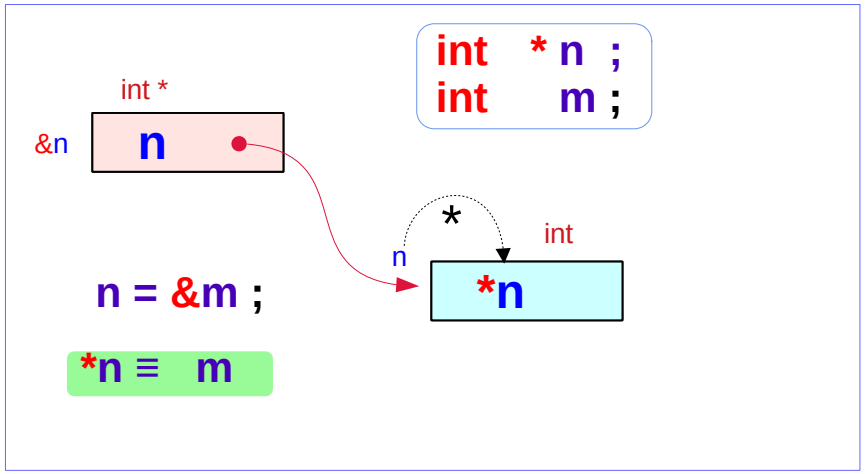
p : a **pointer to** a **1-d array** **int (*) [4]**
with 4 integer elements

fp : a **pointer to** a **function** **int (*) (int, int)**
that takes two integers
and returns an integer





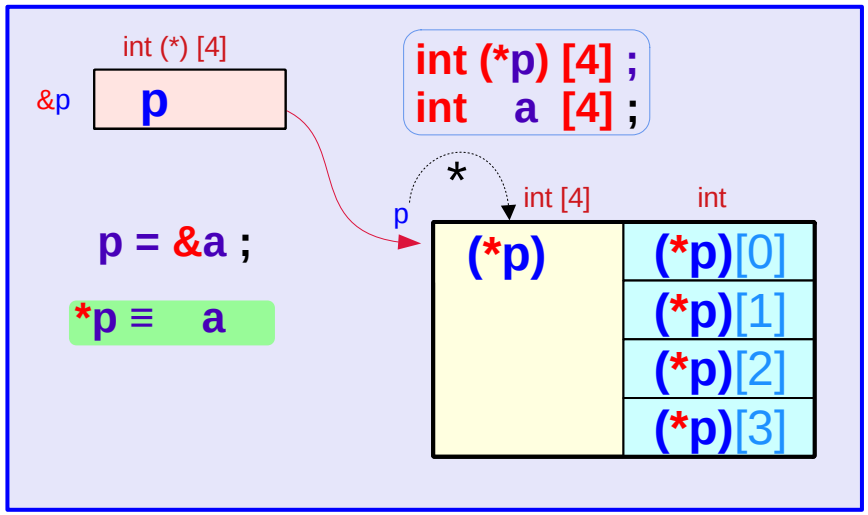
Dereferencing pointer variables **n**, **p**, **fp**

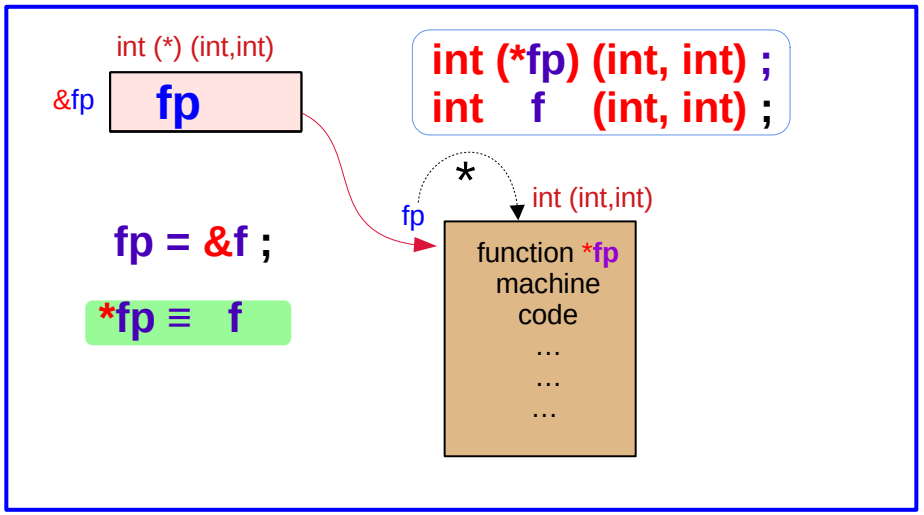


***n** : an **integer** **int**

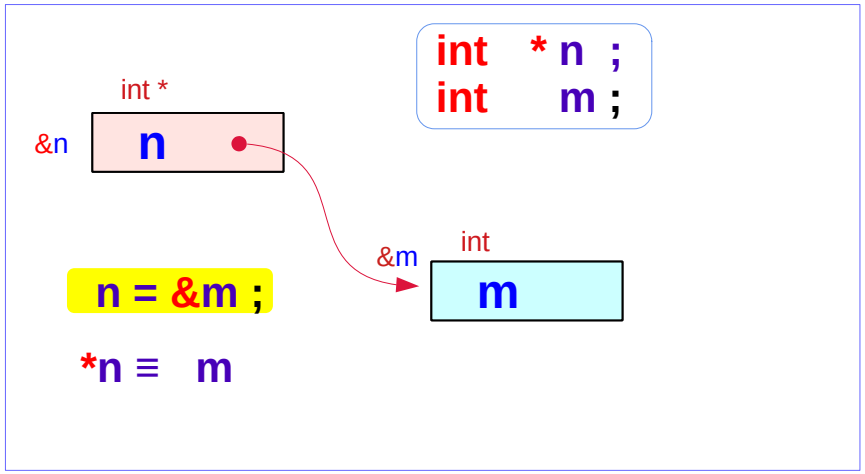
***p** : a **1-d array** **int [4]**
with 4 integer elements

***fp** : a **function** **int (int, int)**
that takes two integers
and returns an integer





Assigning pointer variables **n** and **y[i]**

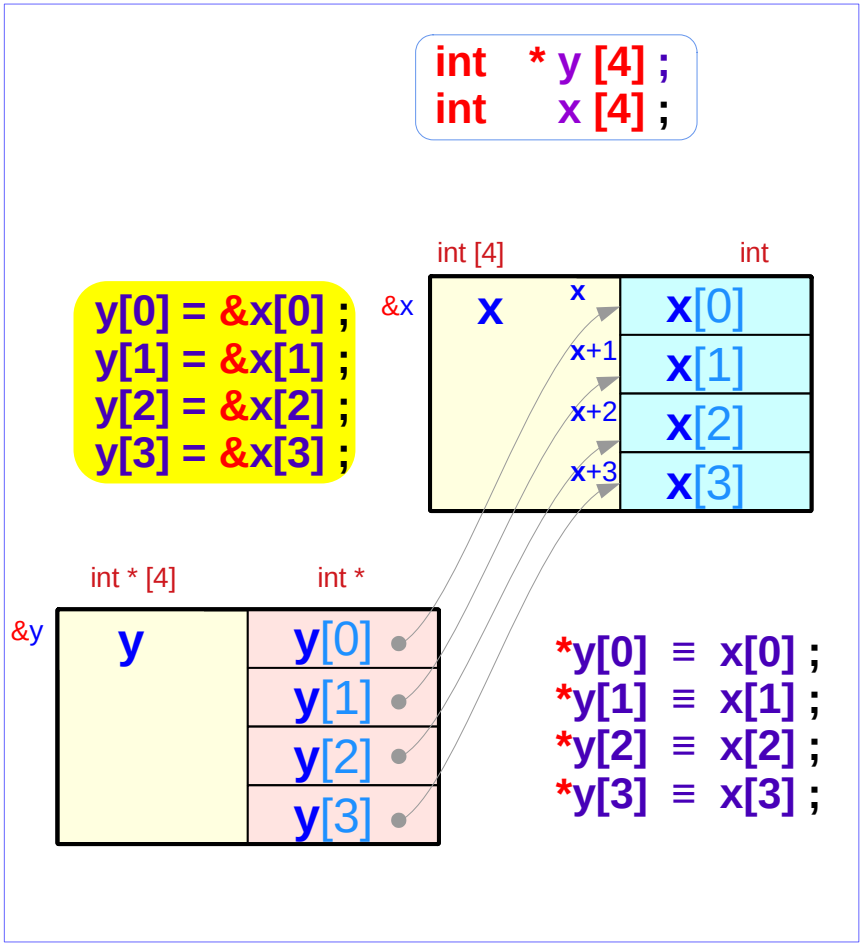


m : an **integer** — **int**

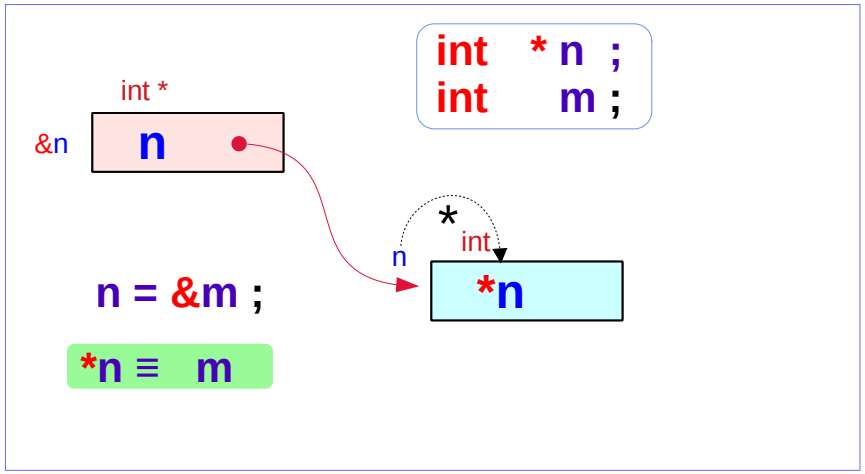
n : an **integer pointer** — **int ***

x : a **1-d array** with 4 **integer** elements — **int [4]**

y : a **1-d array** with 4 **integer pointer** elements — **int * [4]**



Dereferencing pointer variables **n** and **y[i]**

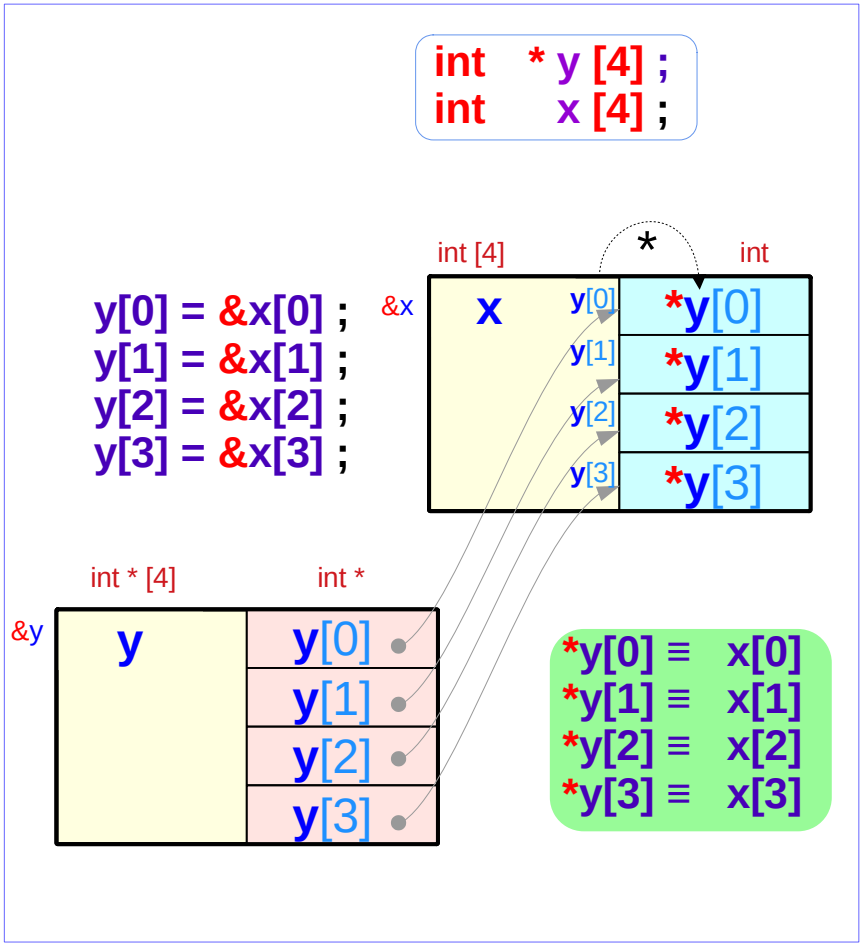


n : an **integer pointer** **int ***

***n** : an **integer** **int**

y[i] : an **integer pointer** **int ***

***y[i]** : an integer **int**



Explicit and implicit array pointers

- **Explicit array pointers**

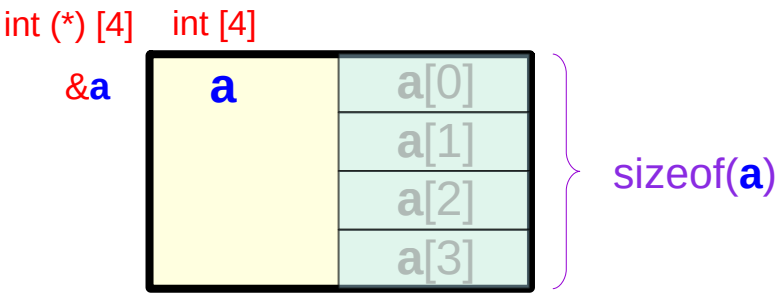
- **Implicit array pointers in an array**

- **Sizes**
- **Types**
- **Values**

Type, size, and value of **a** and &**a** for a **1-d** array **a**

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

outside of an array a –
a** as an **abstract data



abstract data a

value(**a**) = value(&**a**[0])
sizeof(**a**) = 4 * sizeof(int)
type(**a**) = int [4] (outside)
type(**a**) = int (*) (inside)

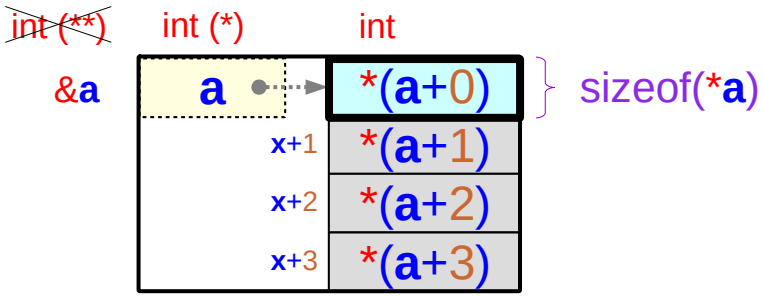
&a : the address of a

value(&**a**) = value(**a**)
sizeof(&**a**) = 4 or 8 bytes
type(&**a**) = int (*) [4]

Type, size, and value of *a and a for a 1-d array a

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

inside of an array a –
a** as a **virtual pointer



primitive data *a

value(*a) = value(a[0])
sizeof(*a) = sizeof(int)
type(*a) = int

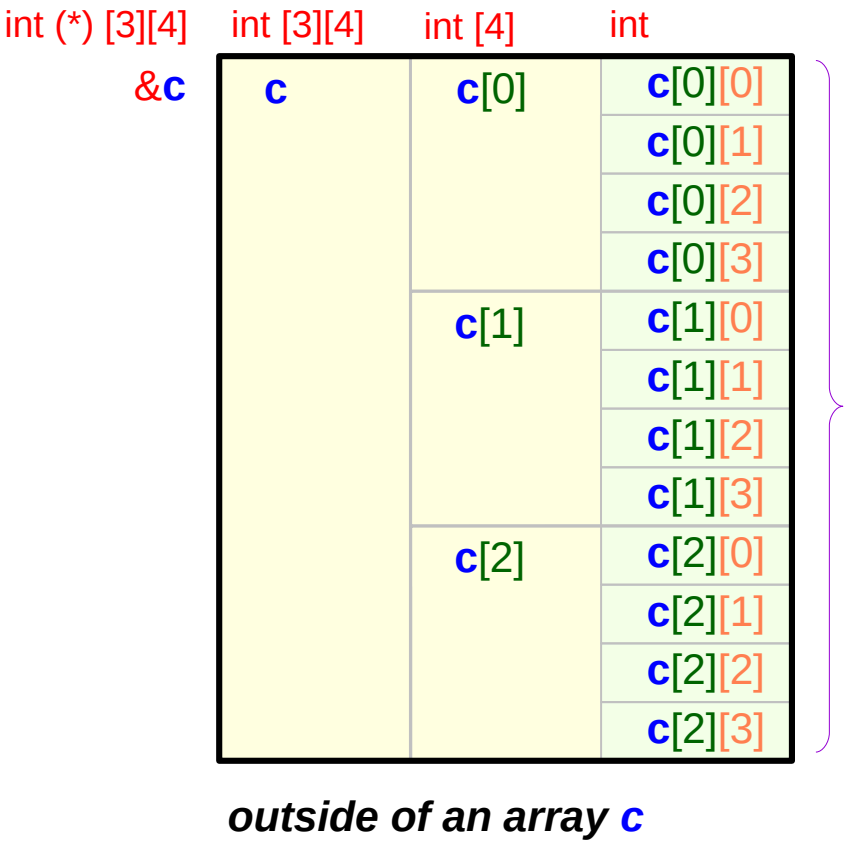
a : the address of *a

value(a) = value(&a[0])
sizeof(a) = 4 * sizeof(int)
type(a) = int [4] (outside)
type(a) = int (*) (inside)

Type, size, and value of **c** and &**c** for a **2-d** array **c**

```
int c [3][4] ;
```

outside of an array c –
c** as an **abstract data



outside of an array c

c : abstract data

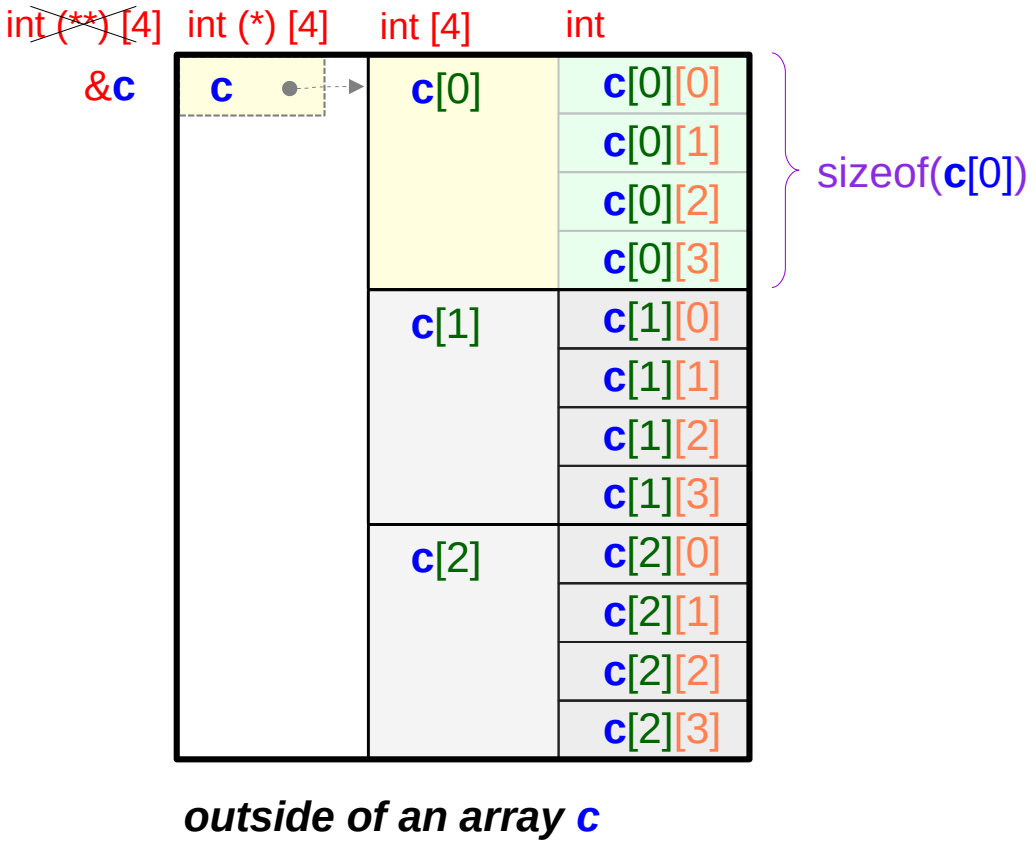
value(**c**) = value(&**c**[0])
sizeof(**c**) = 3 * 4 * sizeof(int)
type(**c**) = int [3][4] (outside)
type(**c**) = int (*)[4] (inside)

&c : the address of c

value(&**c**) = value(**c**)
sizeof(&**c**) = 4 or 8 bytes
type(&**c**) = int (*) [3][4]

Type, size, and value of **c[0]** and **c** for a **2-d** array **c**

```
int c [3][4] ;
```



outside of an array c

inside of an array c –
c as a virtual pointer

c[0] : abstract data *c

value(**c**[0]) = value(&**c**[0][0])
sizeof(**c**[0]) = 4 * sizeof(int)
type(**c**[0]) = int [4] (outside)
type(**c**[0]) = int (*) (inside)

c : the address of c[0]

value(**c**) = value(&**c**[0])
sizeof(**c**) = 3 * 4 * sizeof(int)
type(**c**) = int [3][4] (outside)
type(**c**) = int (*)[4] (inside)

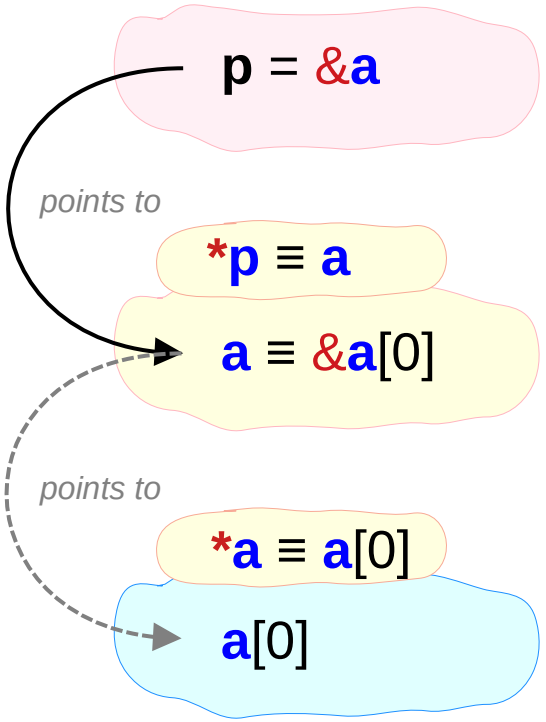
1-d array pointer p – (1) pointer chain

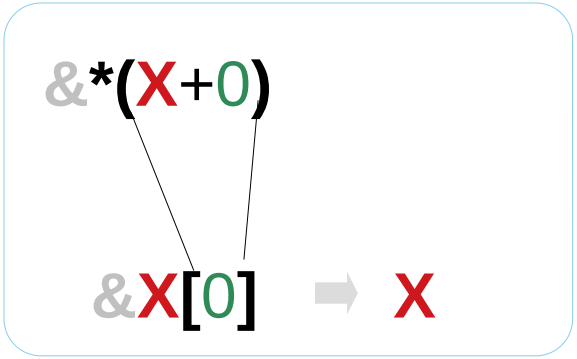
1-d array pointer

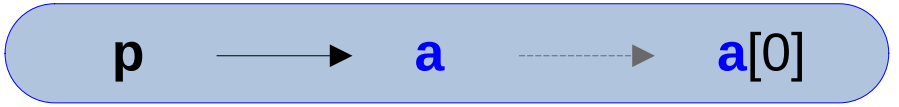
```
int (*p) [4];
```

1-d array

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

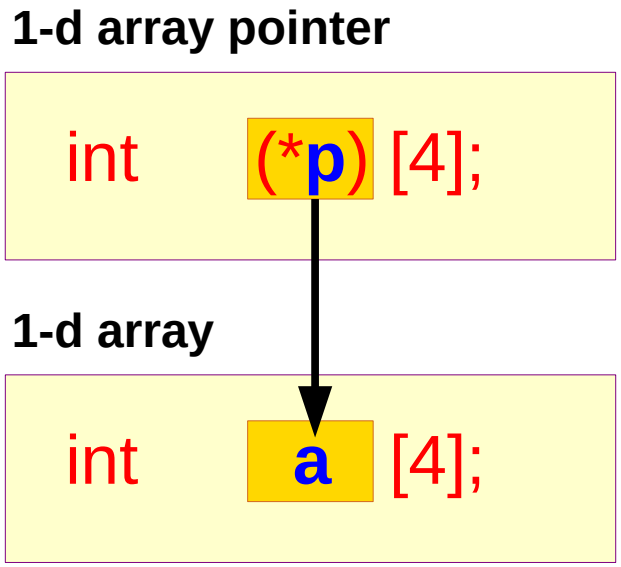


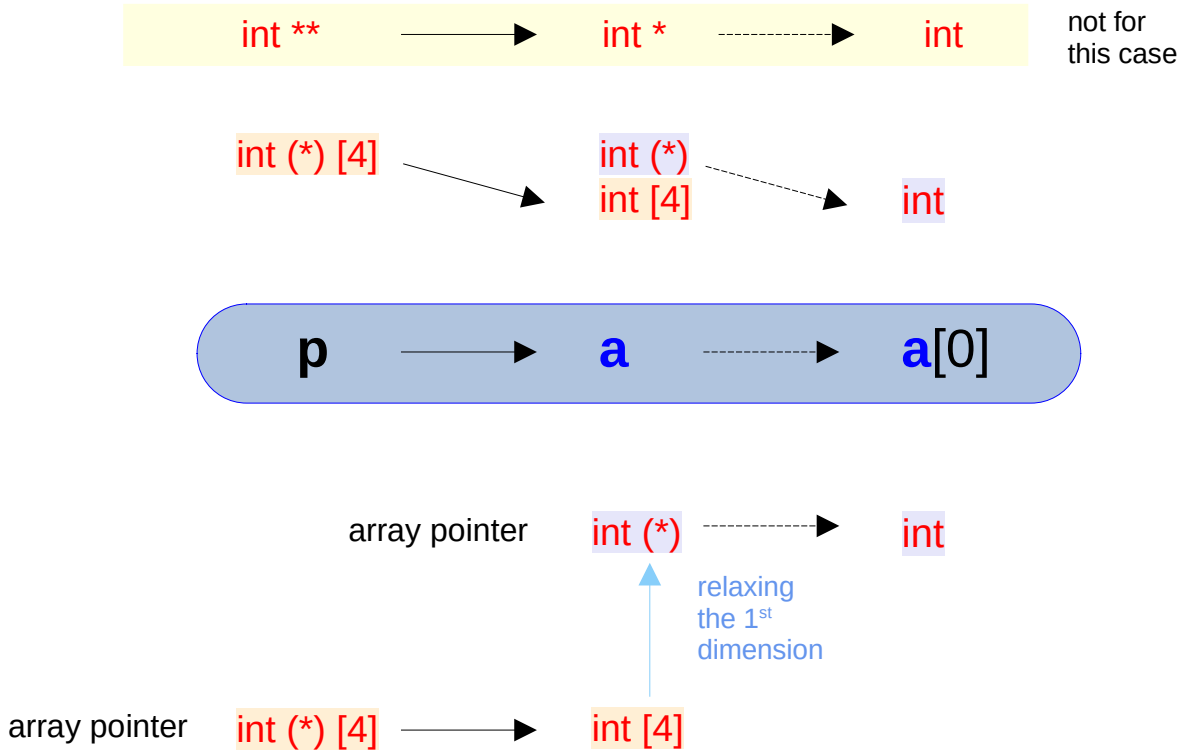




a pointer chain

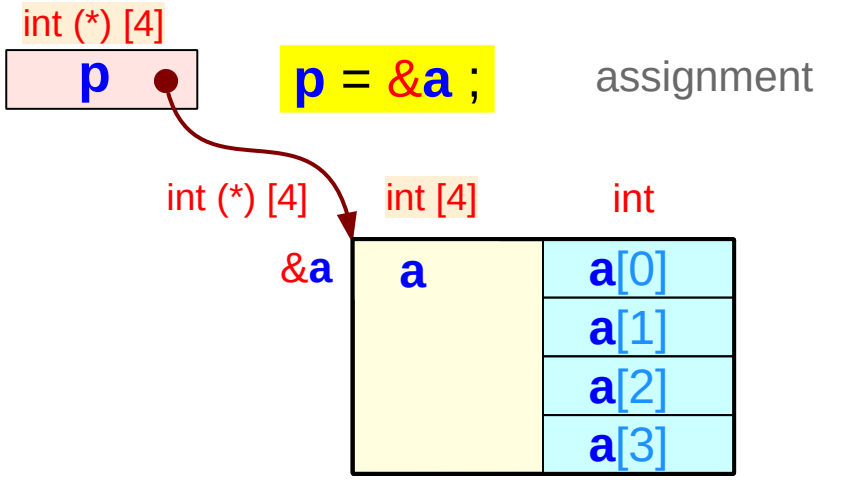
1-d array pointer p – (2) types in a pointer chain

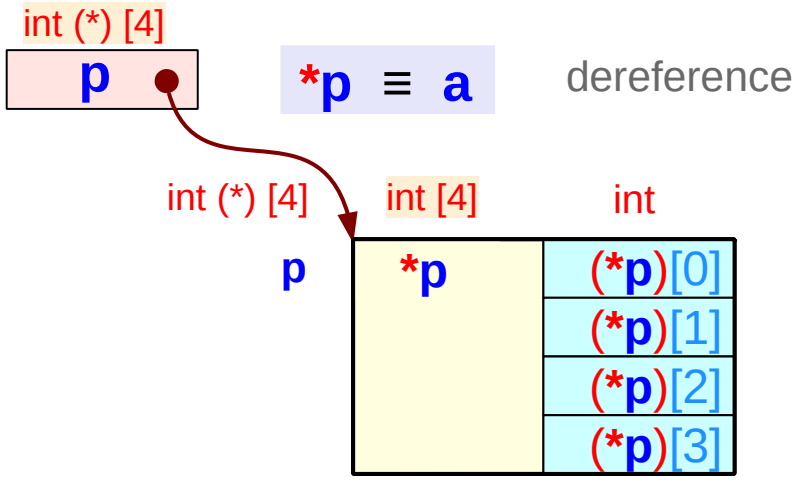




1-d array pointer p – (3) assignment and dereference

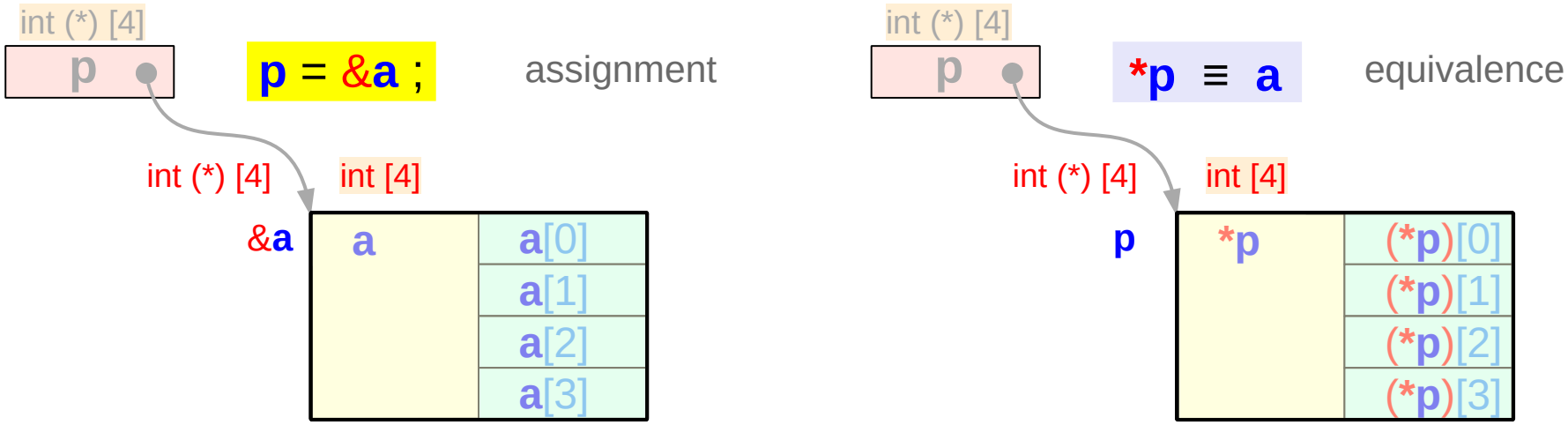
	assignment	dereference	equivalence	dereference
int a [4];				
int (*p) [4];	p = &a	*p ≡ a	a ≡ &a[0]	*a ≡ a[0]





1-d array pointer p – (4) abstract data a

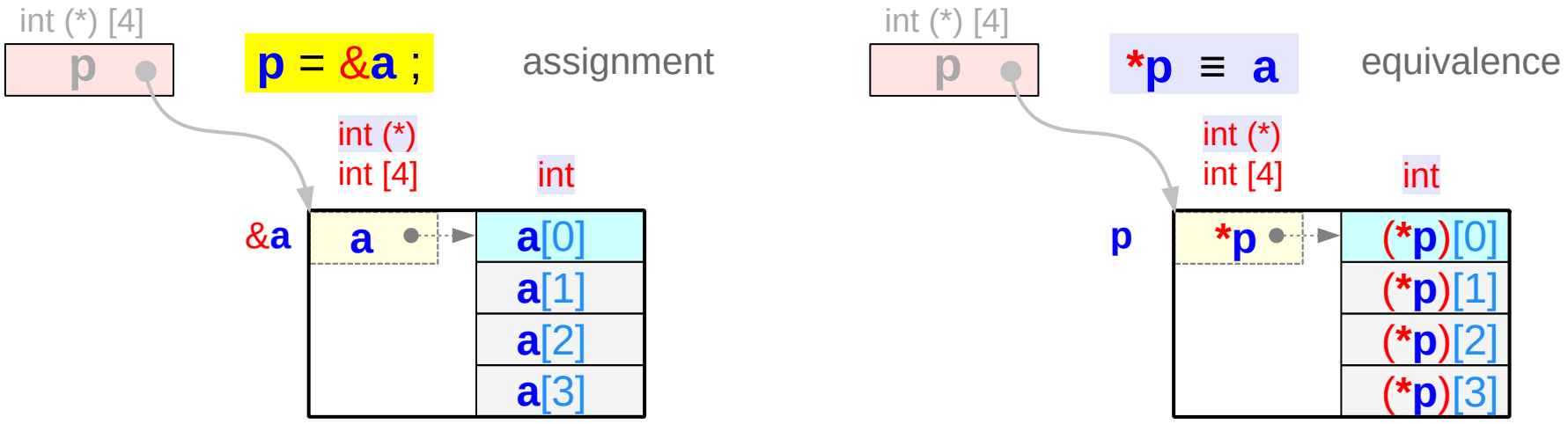
outside of an array a (***a*** *as an* ***abstract data***)



sizeof(&a)	= 4 or 8 bytes	size of a pointer	value(&a)	address value of a 1-d array a
sizeof(a)	= 4 * sizeof(int)	size of a **1-d** array	= value(a)	data value of a 1-d array a

0-d array pointer **a** – (5) primitive data **a[0]**

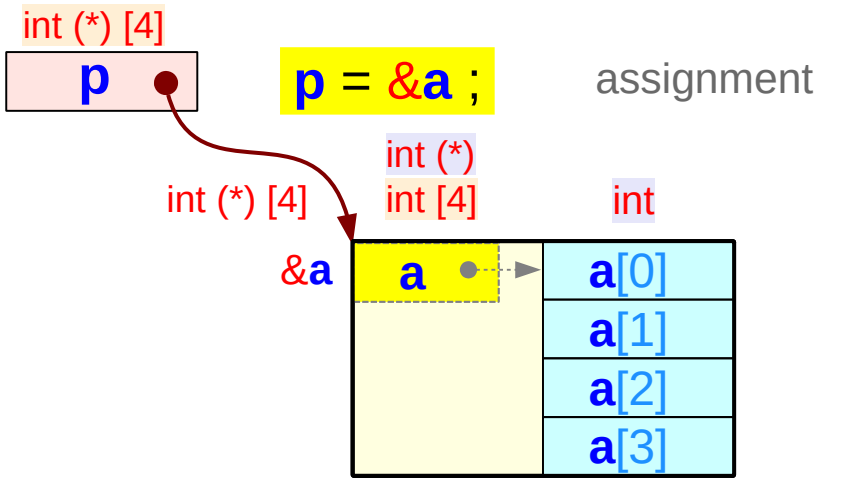
inside of an array a (***a*** *as a* ***virtual pointer***)

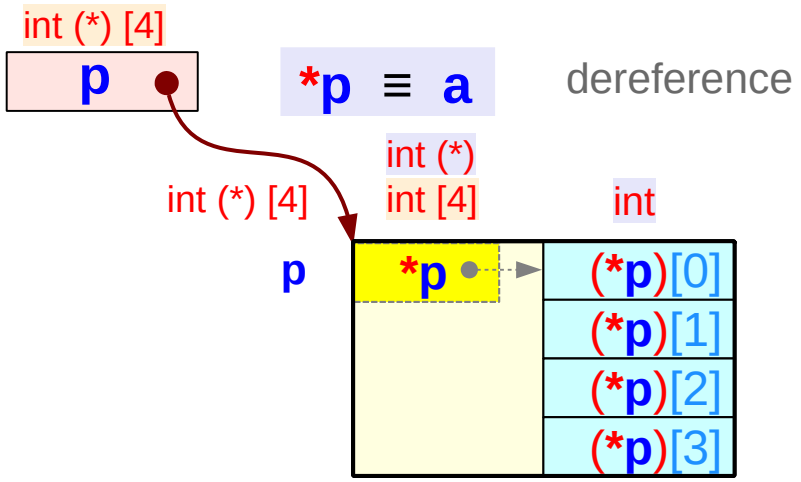


sizeof(**a**) = 4 * sizeof(int)	size of a **1-d** array	value(**a**) = value(&**a**[0])	address value of an integer **a**[0]
sizeof(**a**[0]) = 4 bytes	size of an integer	value(**a**[0])	data value of an integer **a**[0]

Overlayed representation

outside of an array a (***a*** *as an* ***abstract data***)
inside of an array a (***a*** *as a* ***virtual pointer***)





not a real pointer a value(&a) = value(a) = value(&a[0])

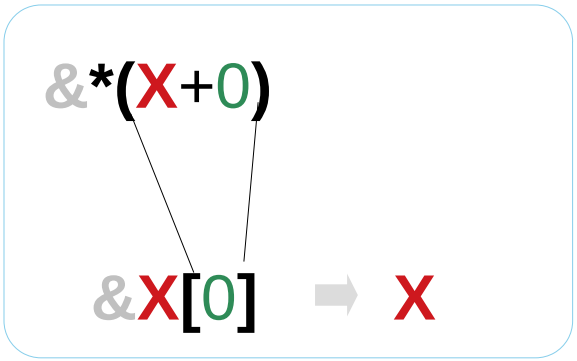
2-d array pointer q – (1) pointer chains

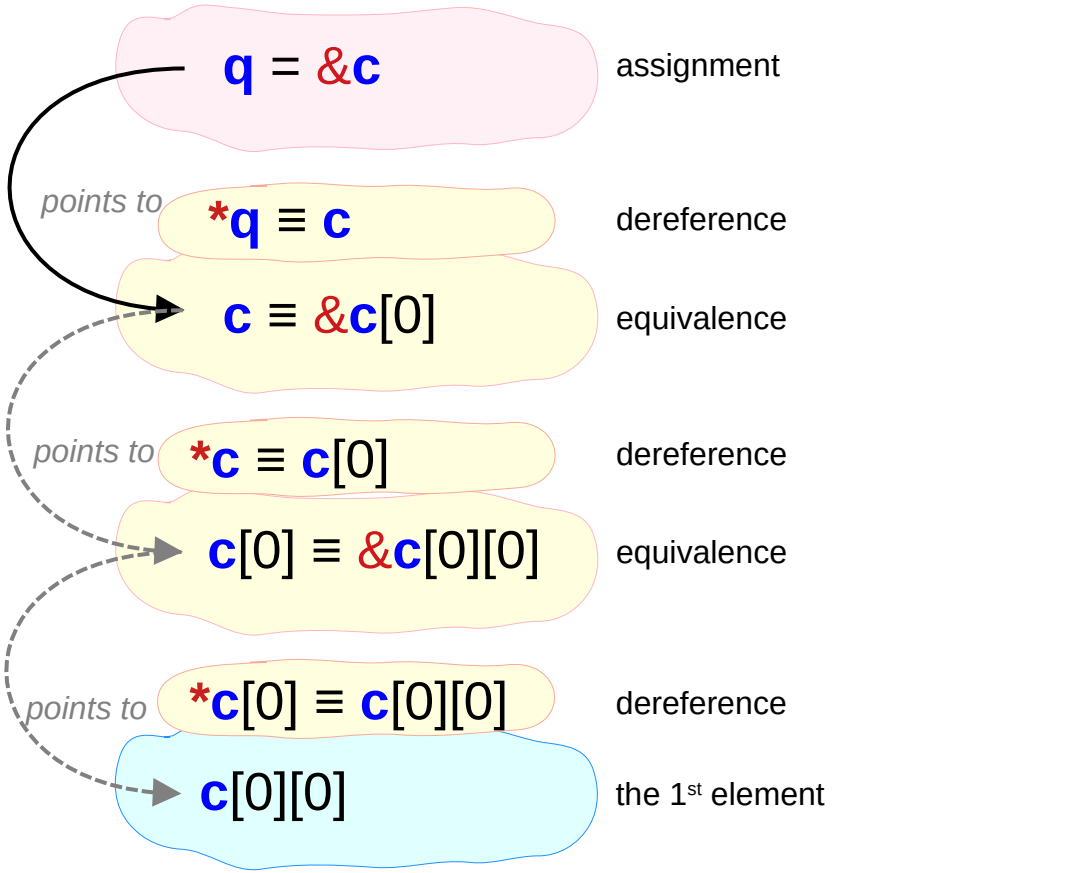
2-d array pointer

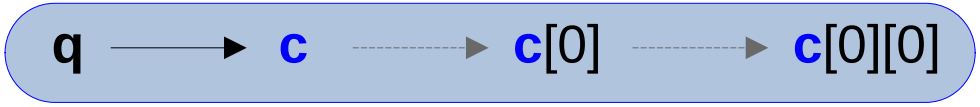
```
int (*q) [3][4];
```

2-d array

```
int  c  [3][4];
```

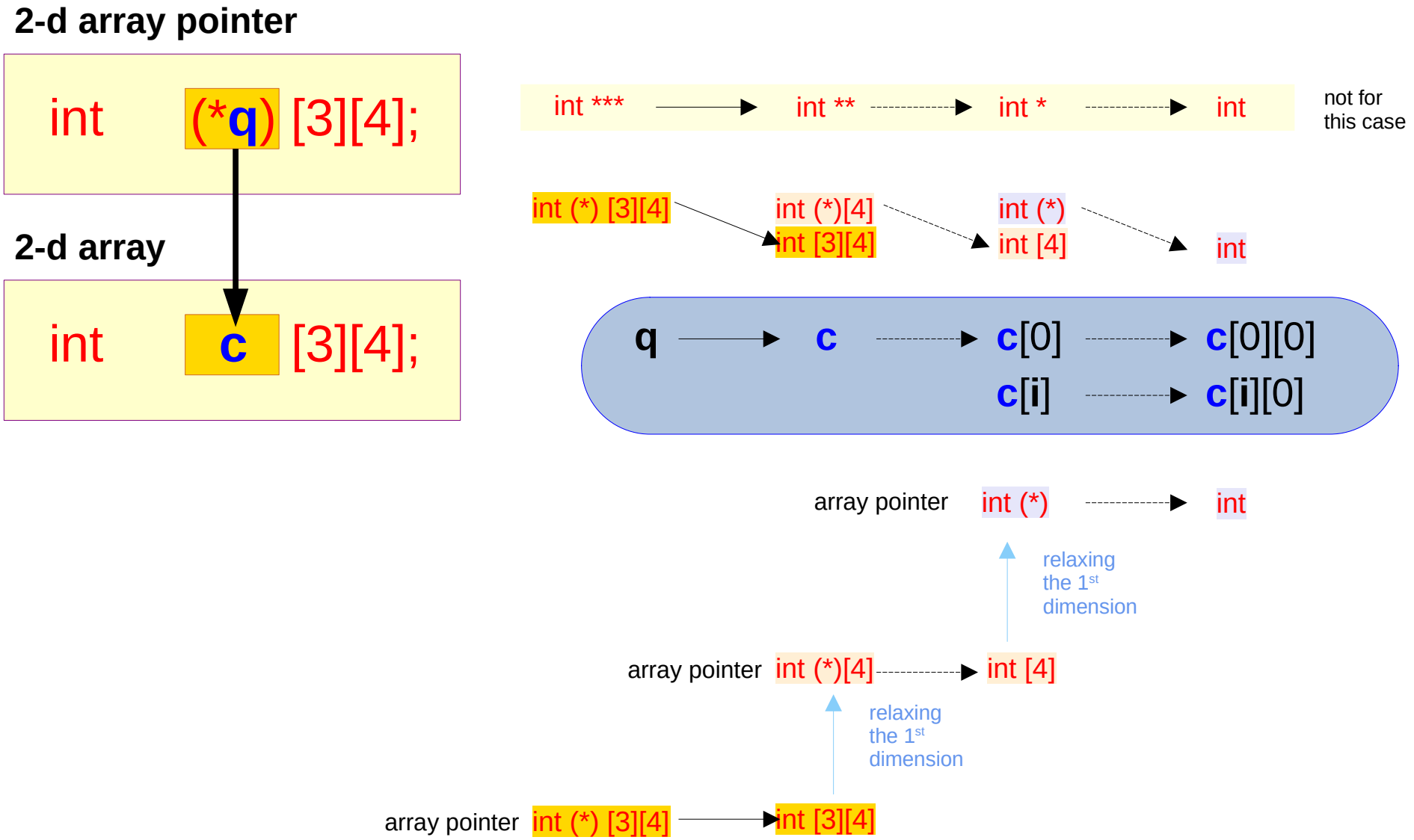






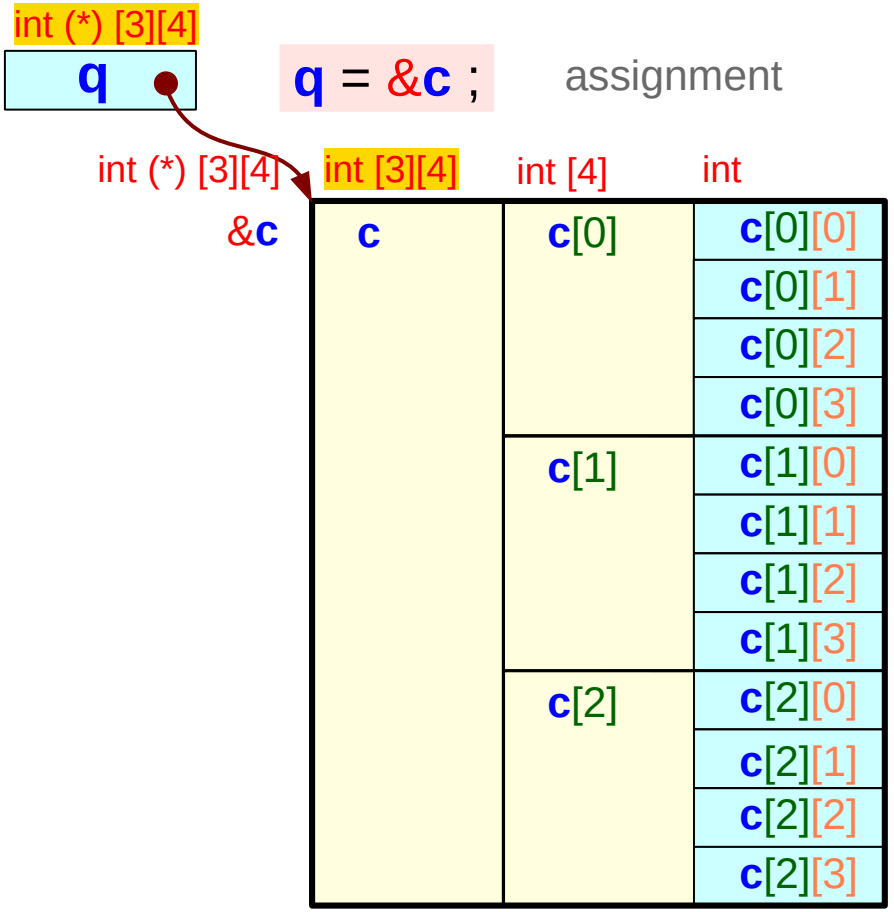
a pointer chain

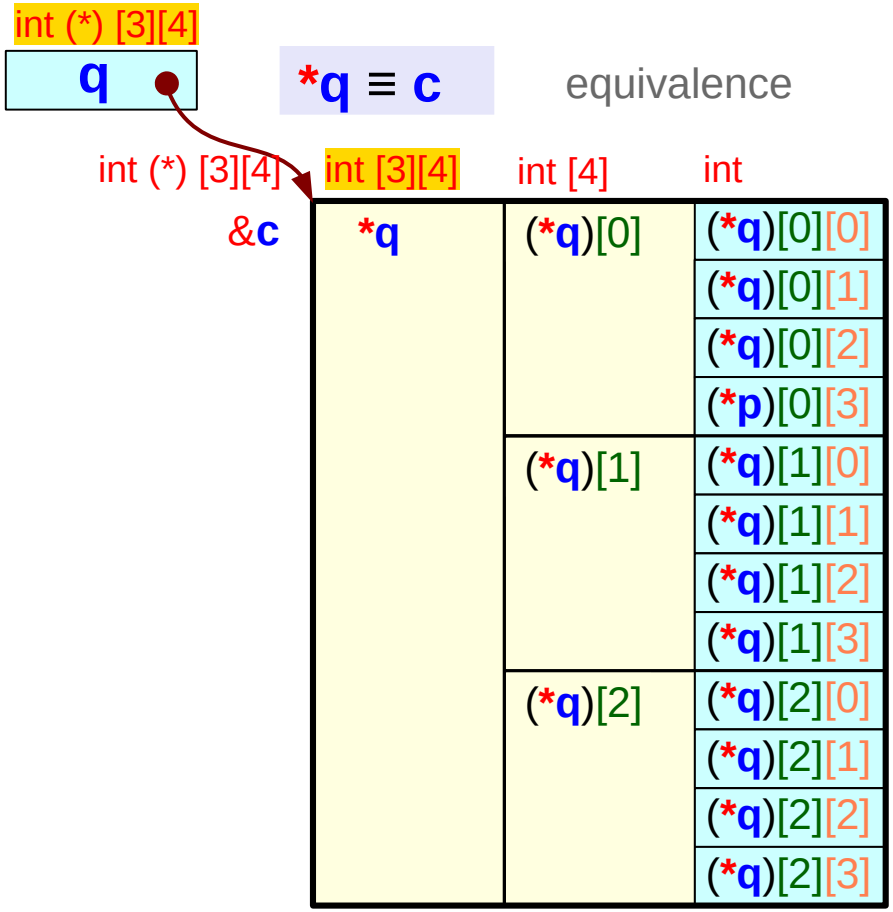
2-d array pointer q – (2) types in a pointer chain



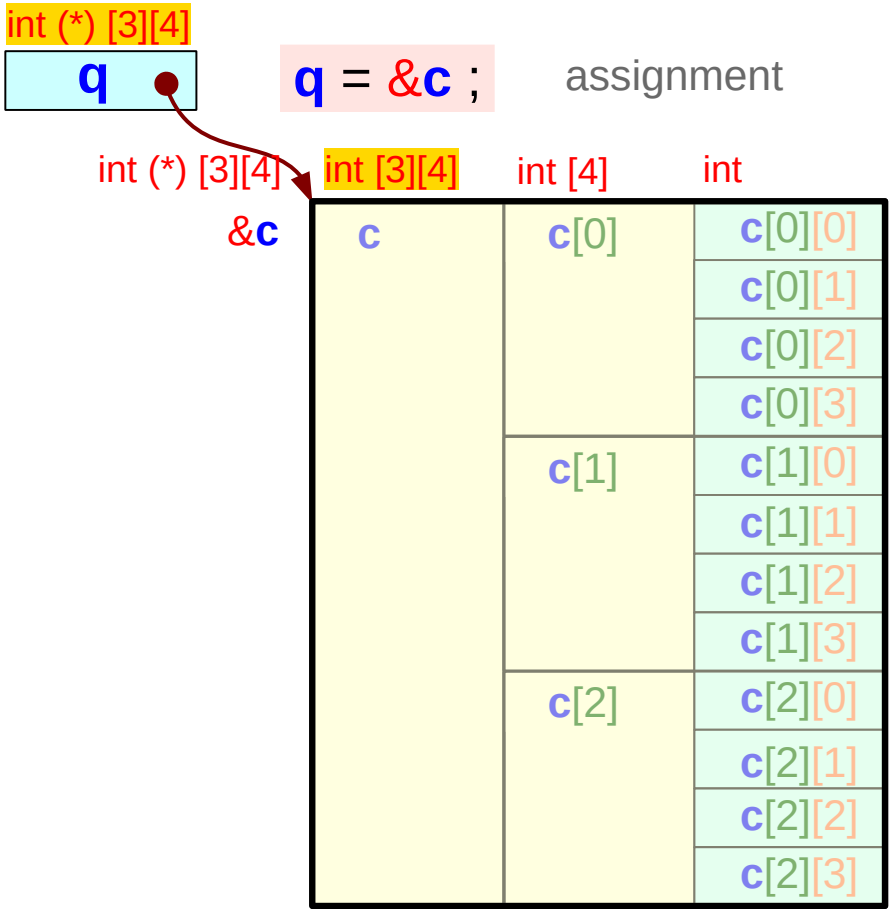
2-d array pointer q – (3) assignment and dereference

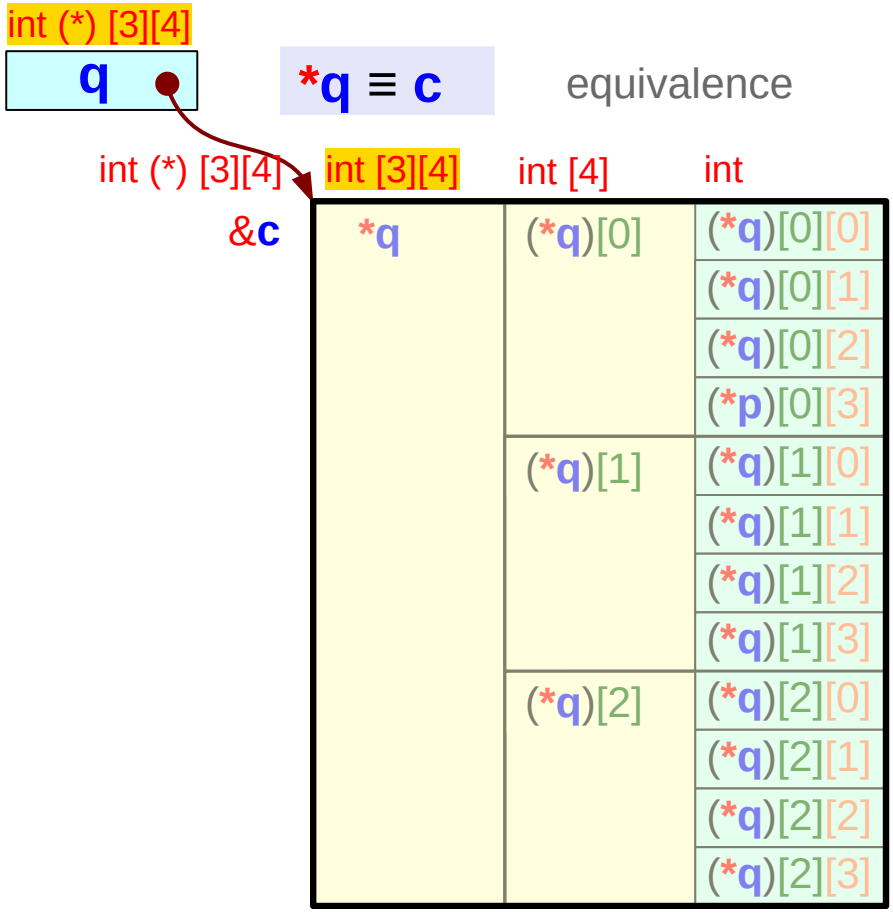
	assignment	dereference	equivalence	equivalence
int c [3][4];				
int (*q) [3][4];	q = &c	*q ≡ c	c ≡ &c[0]	c[0] ≡ &c[0][0]





2-d array pointer q – (4) abstract data c

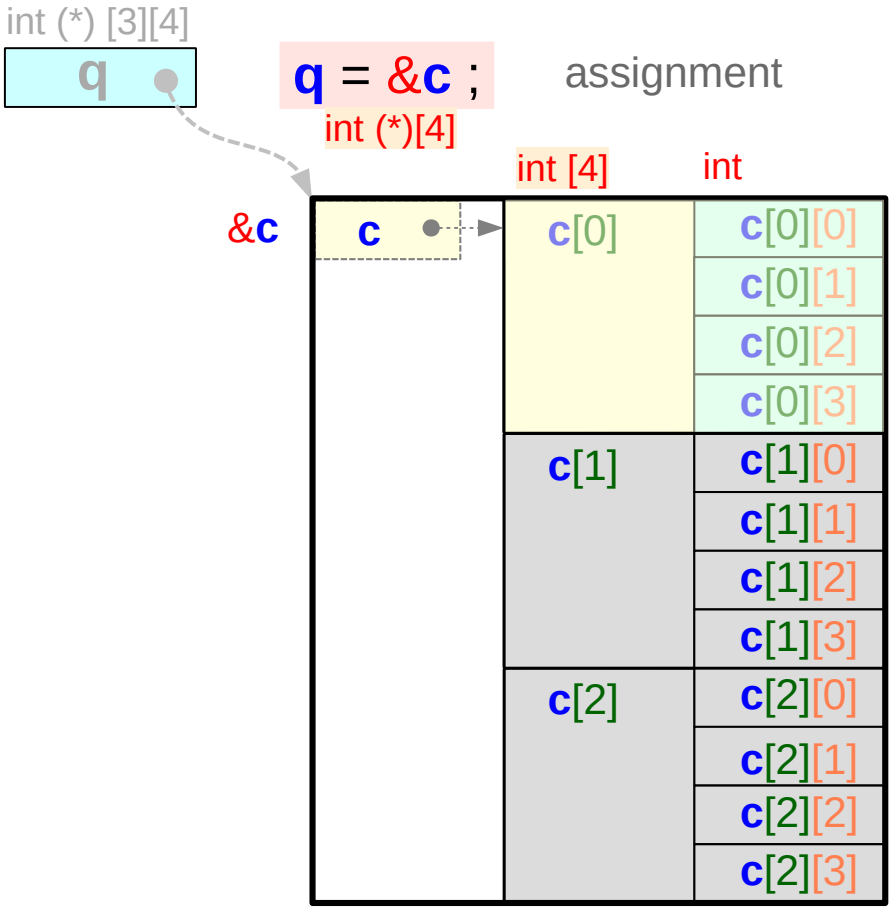


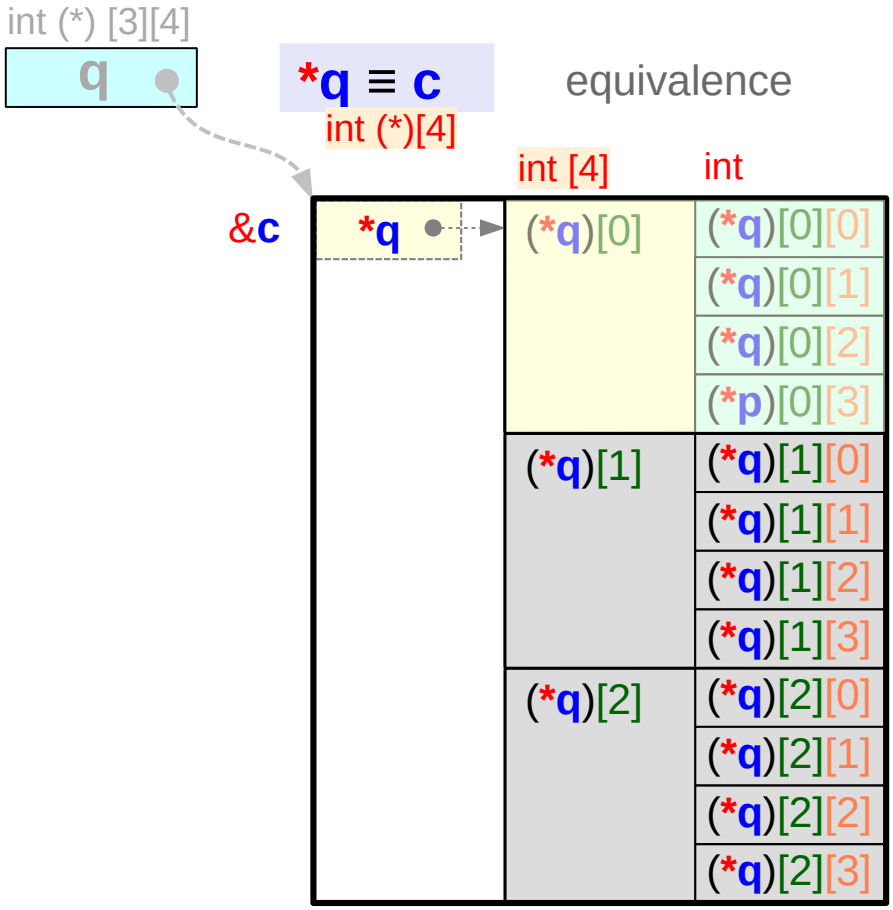


outside of an array c (**c** *as an* ***abstract data***)

sizeof(&**c**) = 4 or 8 bytes	size of a pointer	value(&**c**) =	address value of a **2-d** array **c**
sizeof(**c**) = 3 * 4 * sizeof(int)	size of a **2-d** array	value(**c**)	data value of a **2-d** array **c**

1-d array pointer c – abstract data c[0]

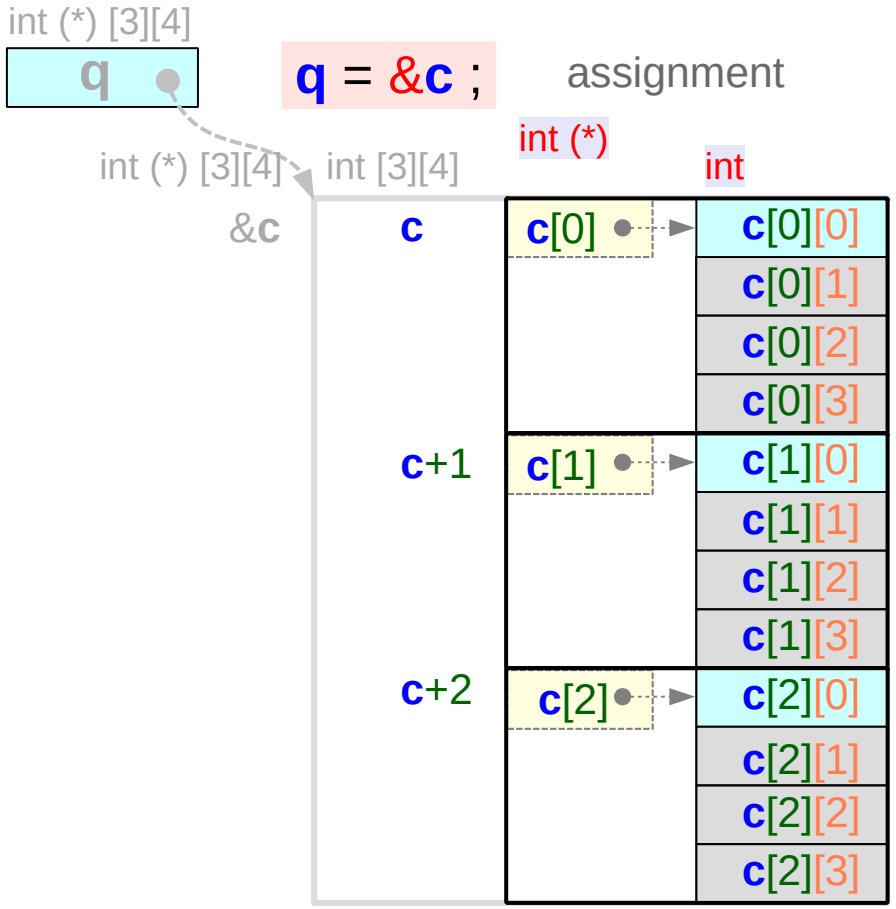


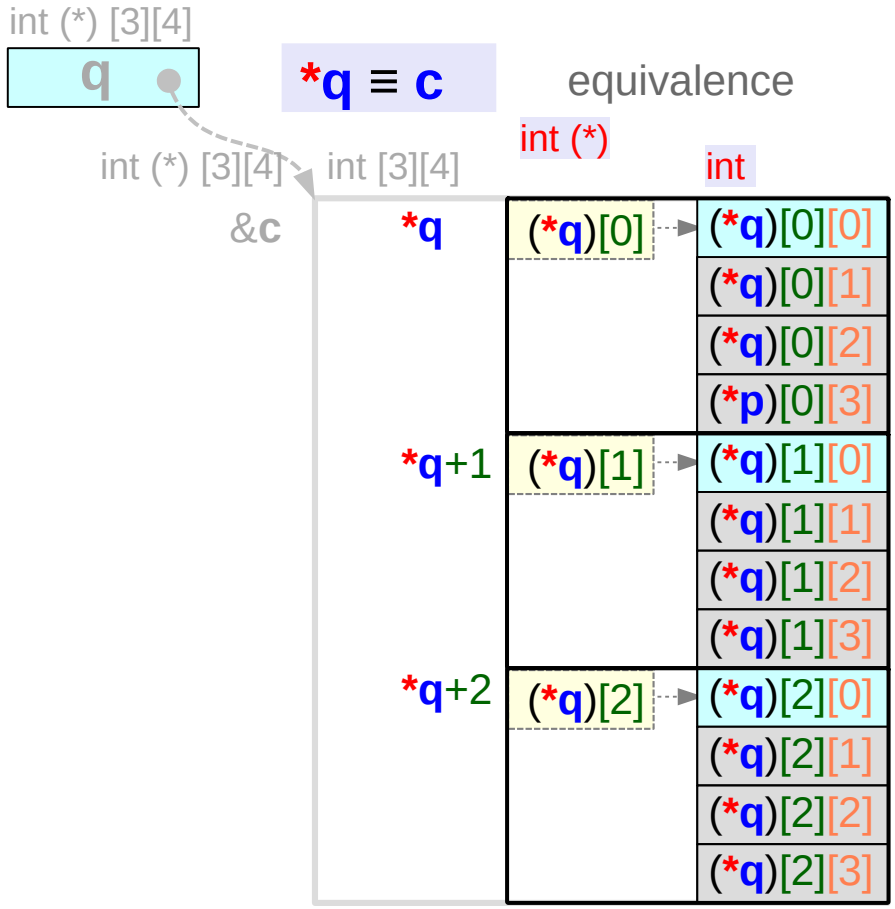


inside of an array c (***c as a virtual pointer, c[i]*** *as an* ***abstract data***)

sizeof(**c**) = 3 * 4 * sizeof(int)	size of a **2-d** array	value(**c**) =	address value of a **1-d** array **c**[0]
sizeof(**c**[0]) = 4 * sizeof(int)	size of a **1-d** array	value(**c**[0])	data value of a **1-d** array **c**[0]

0-d array pointer c[i] – primitive data c[i][0]

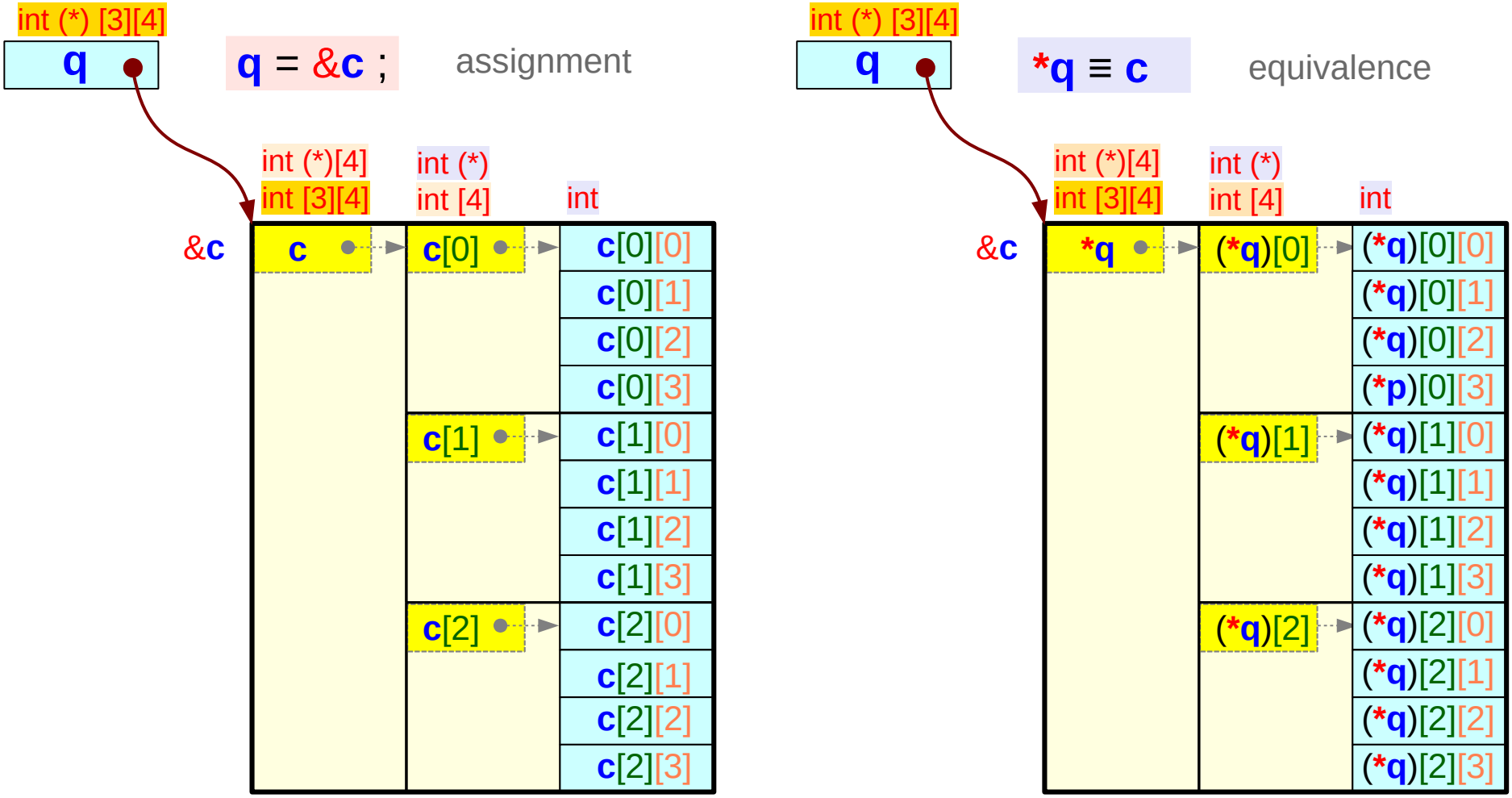




inside of an array c[i] (***c[i]*** ***as a virtual pointer,*** ***c[i][j]*** *as a* ***primitive data***)

sizeof(**c**[i]) = 4 * sizeof(int)	size of a **1-d** array	value(**c**[i]) = value(&**c**[i][0])	address value of an integer **c**[i][0]
Sizeof(**c**[i][0]) = sizeof(int)	size of an integer	value(**c**[i][0])	data value of an integer **c**[i][0]

Overlayed representation



not a real pointer c

not a real pointer c[i]

value(&c) = value(c) = value(&c[0]) = value(c[0]) = value(&c[0][0])

value(&c[i]) = value(c[i]) = value(&c[i][0])

Array pointers

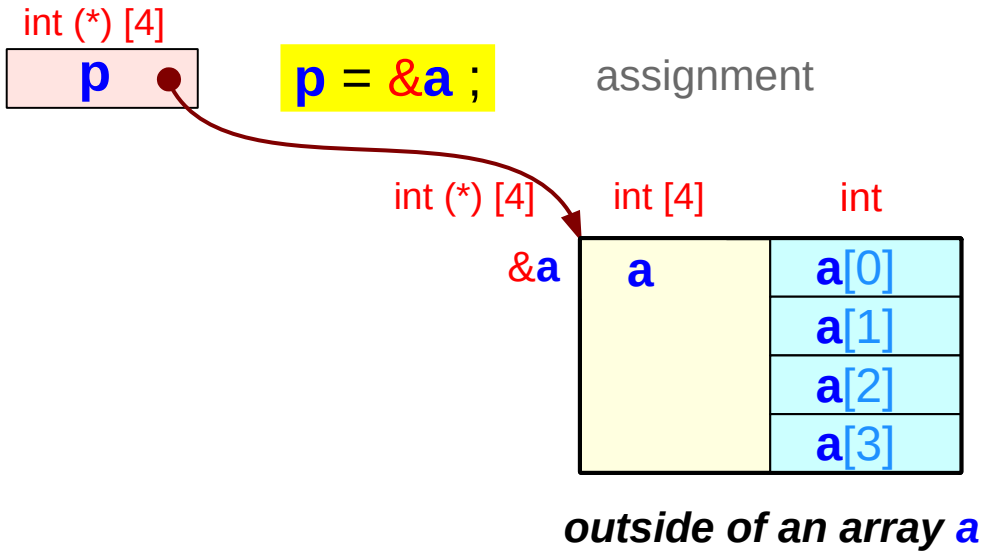
pointer to a 1-d array a and its 0-d sub-array a[0]

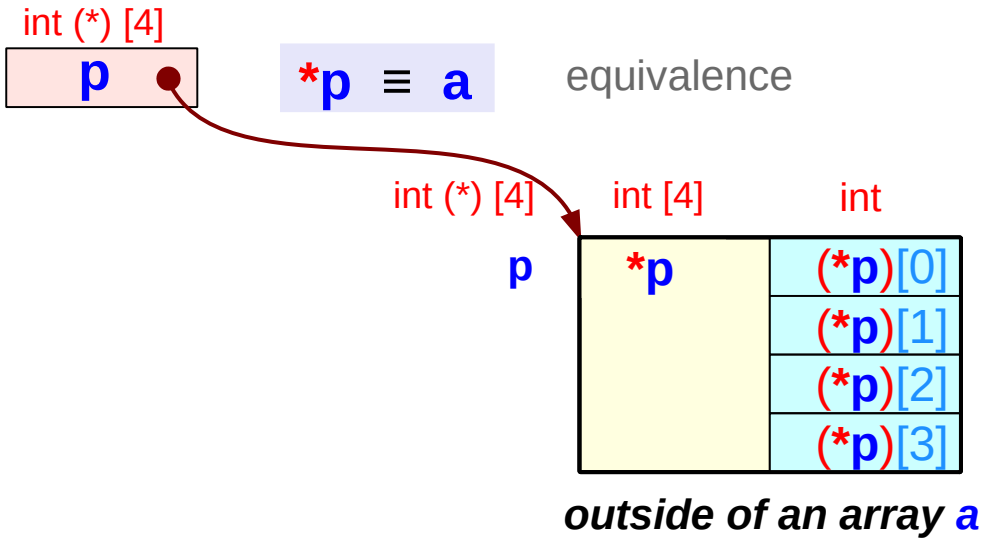
```
int  (*p) [4];     (1-d array pointer)      int a [4];     (1-d array)
int  (*q) ;        (0-d array pointer)
```

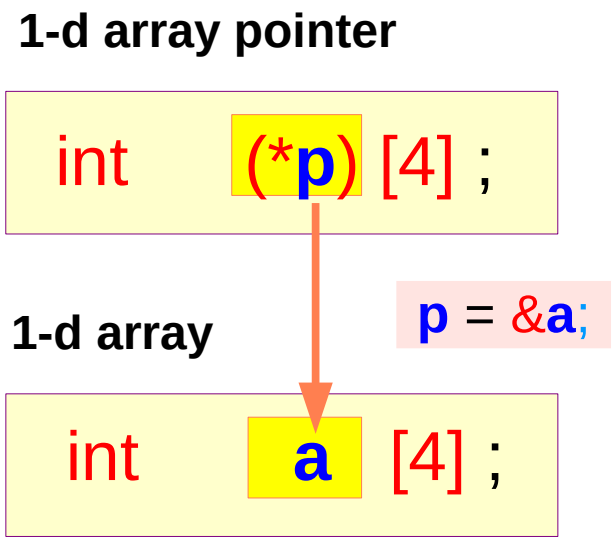
pointer to a 2-d array c and its 1-d sub-array c[0]

```
int  (*p) [3][4] ;  (2-d array pointer)     int c [3][4];  (2-d array)
int  (*q) [4] ;     (1-d array pointer)
```

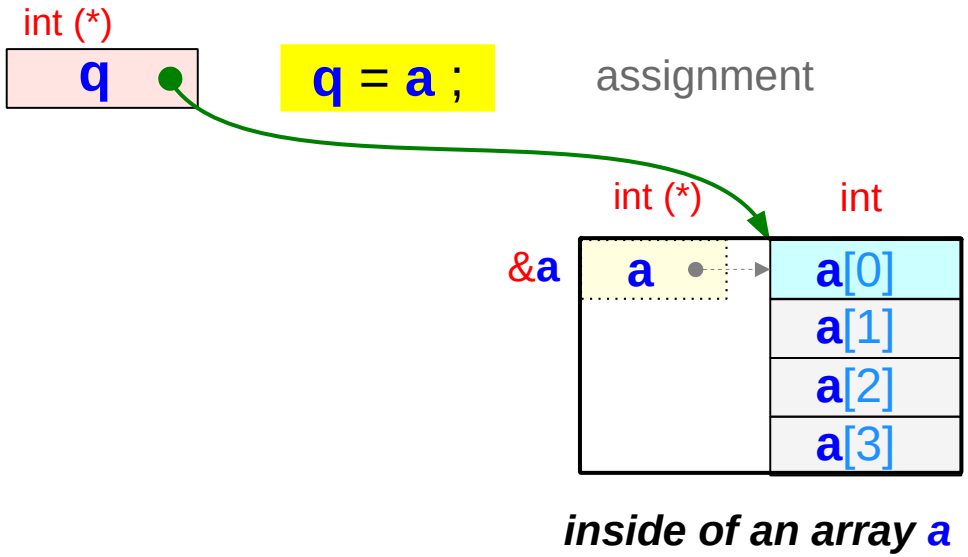
1-d array pointer p to a 1-d array a

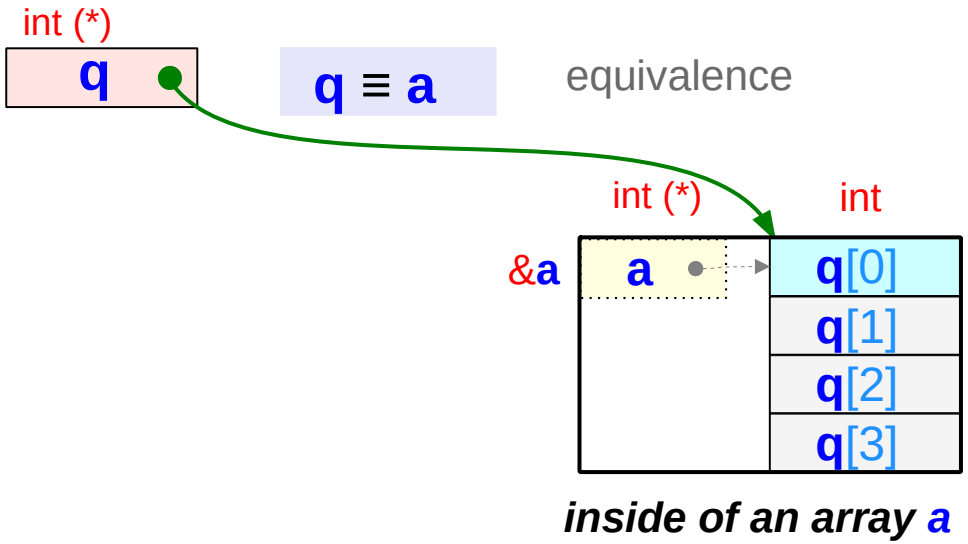


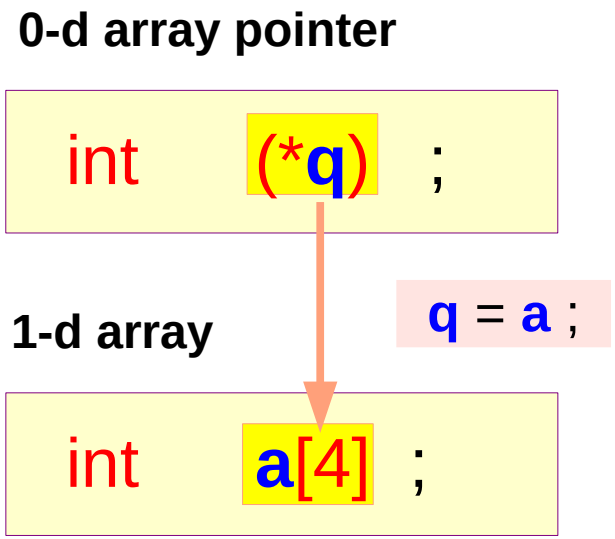




0-d array pointer q to a 0-d sub-array a[0]







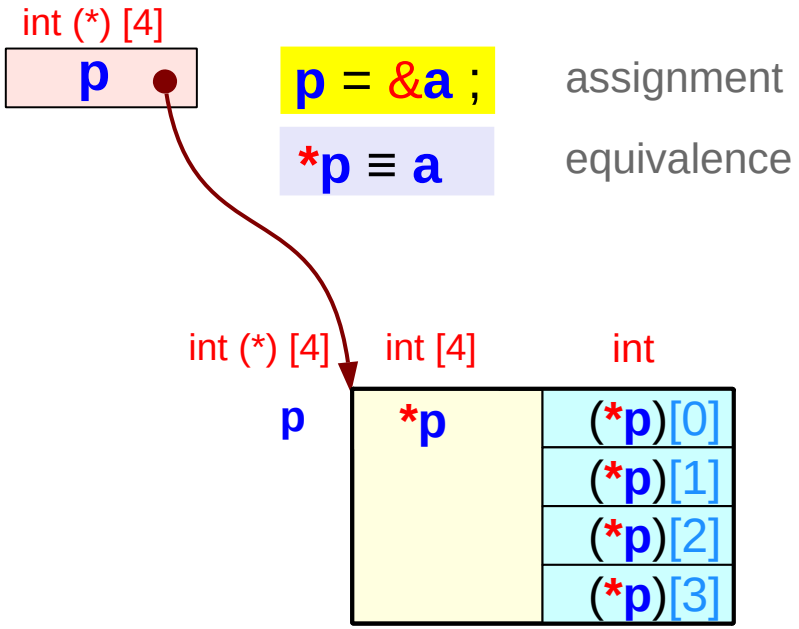
a[i]
a[0]

&a[0] = &*(a+0) = a

1-d array access using p and q

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

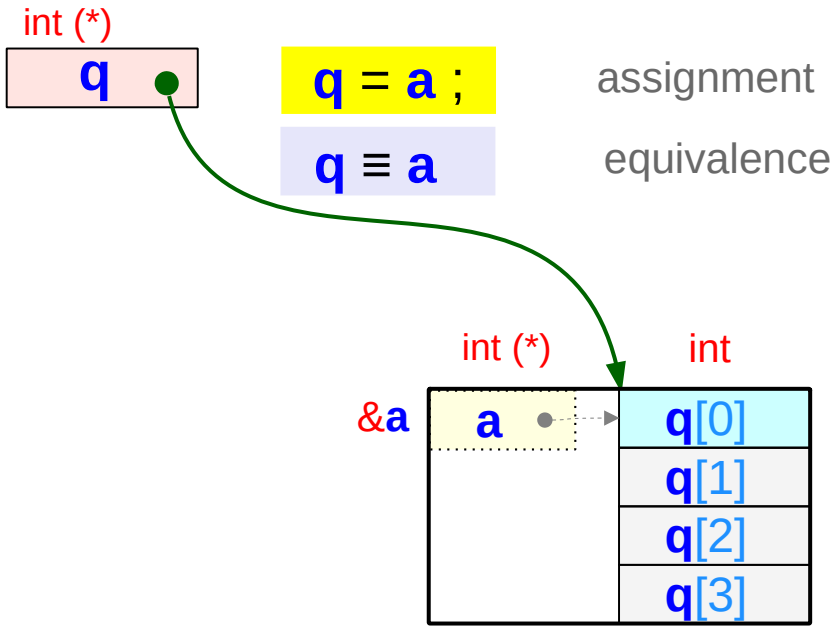
1-d array pointer



outside of an array a

```
int  (*q)   = a;
```

0-d array pointer



inside of an array a

2-d array pointer p to a 2-d array c – reference

2-d array pointer

int (*) [3][4]

p

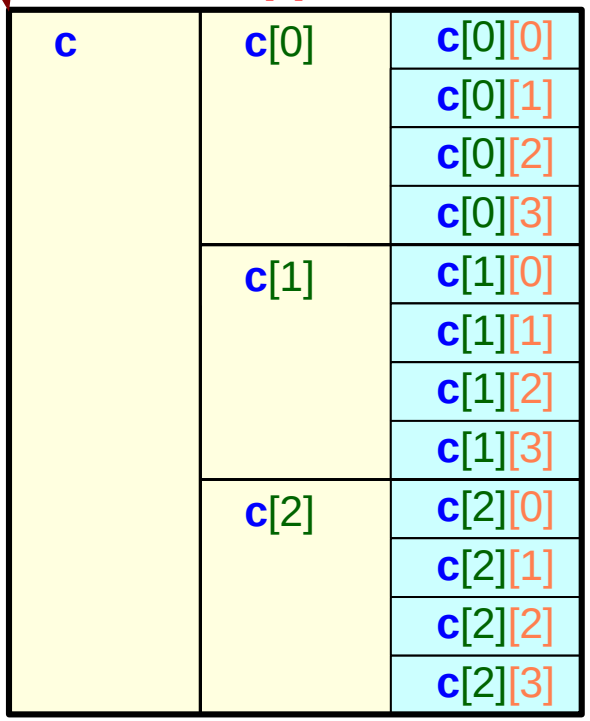
p = &**c** ; assignment

int (*) [3][4] &**c**

int [3][4]

int [4]

int



outside of an array c

2-d array pointer

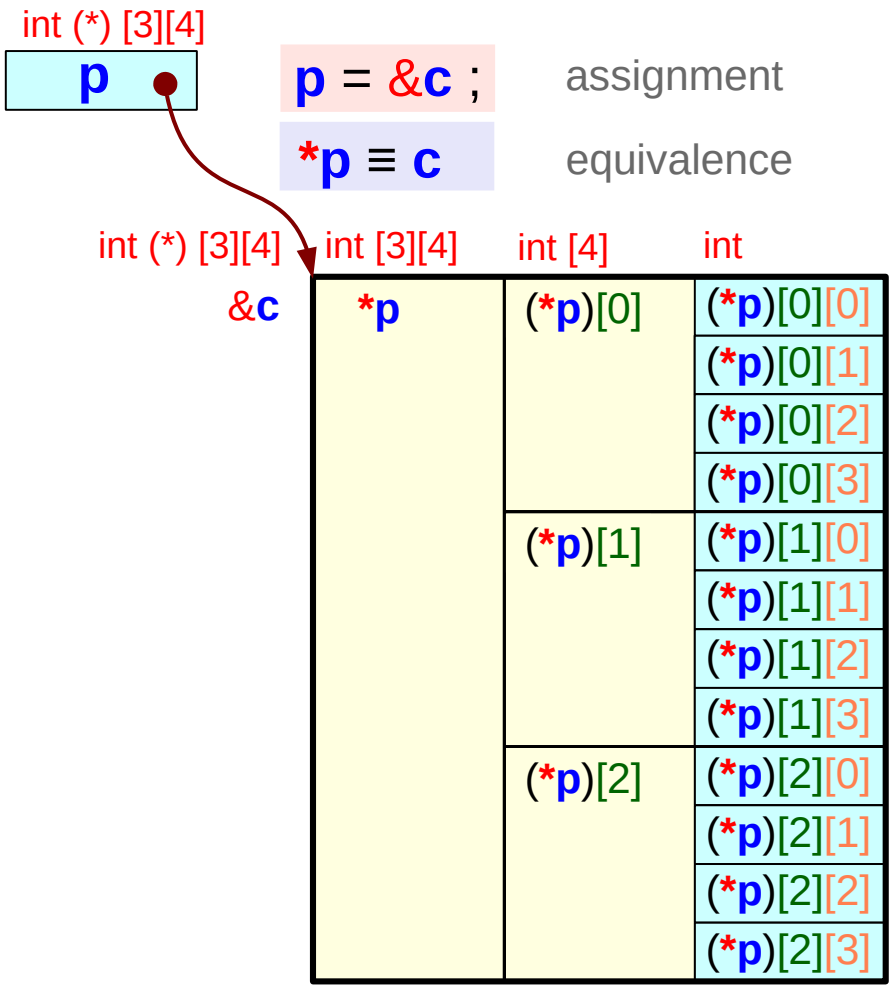
int (*p) [3][4] ;

2-d array

int c [3][4] ;

2-d array pointer p to a 2-d array c – dereference

2-d array pointer



outside of an array c

2-d array pointer

```
int (*p) [3][4] ;
```

2-d array

```
int  c  [3][4] ;
```

1-d array pointer q to a 1-d subarray c[0] – reference

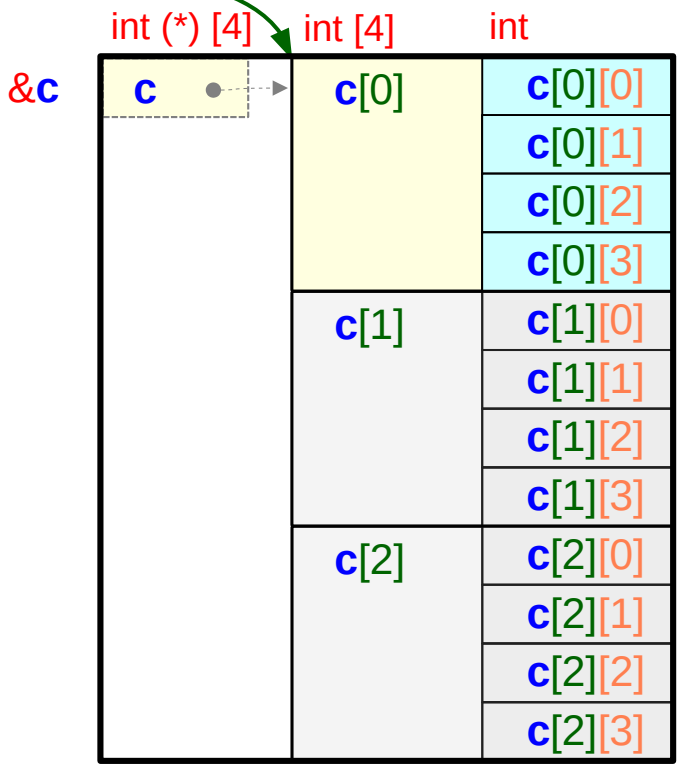
1-d array pointer

int (*) [4]

q

&**c**[0] = &***(c**+0) = **c**

q = **c** ; assignment



1-d array pointer

int (*q) [4] ;

1-d subarray

int c[3] [4] ;

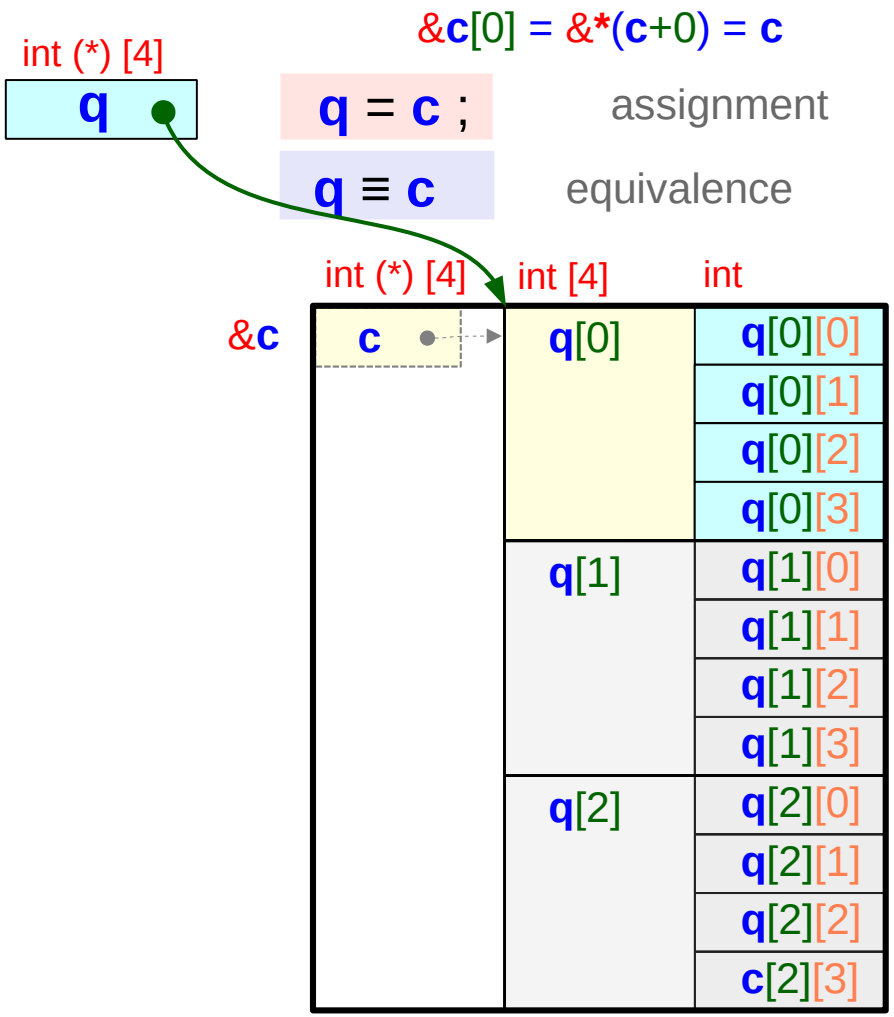
Declaration:
[3] means there are 3 elements
Expression:
[0], [1], [2] are used

among 3 elements c[0], c[1], c[2],
consider the first one c[0]

inside of an array c ***outside of an array c[0]***

1-d array pointer q to a 1-d subarray c[0] – dereference

1-d array pointer



1-d array pointer

int (*q) [4] ;

1-d subarray

int c[3] [4] ;

Declaration:
[3] means there are 3 elements
Expression:
[0], [1], [2] are used

among 3 elements c[0], c[1], c[2],
consider the first one c[0]

inside of an array c ***outside of an array c[0]***

2-d array access using p and q

2-d array pointer

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

p = &c ;

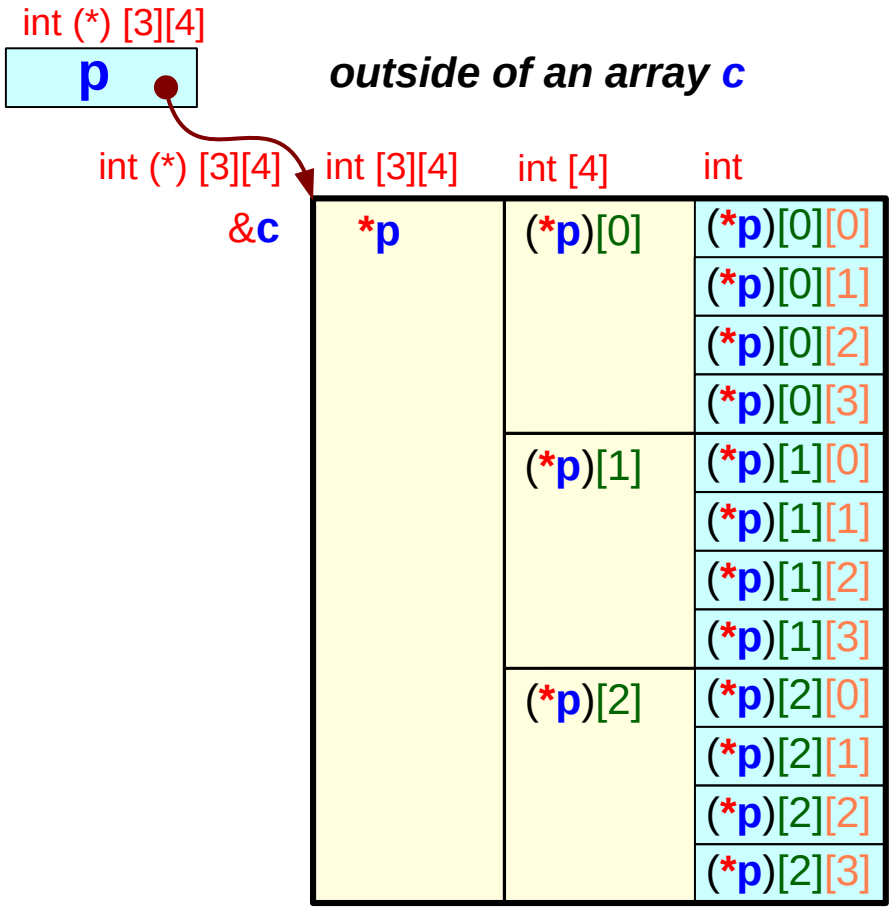
*p ≡ c

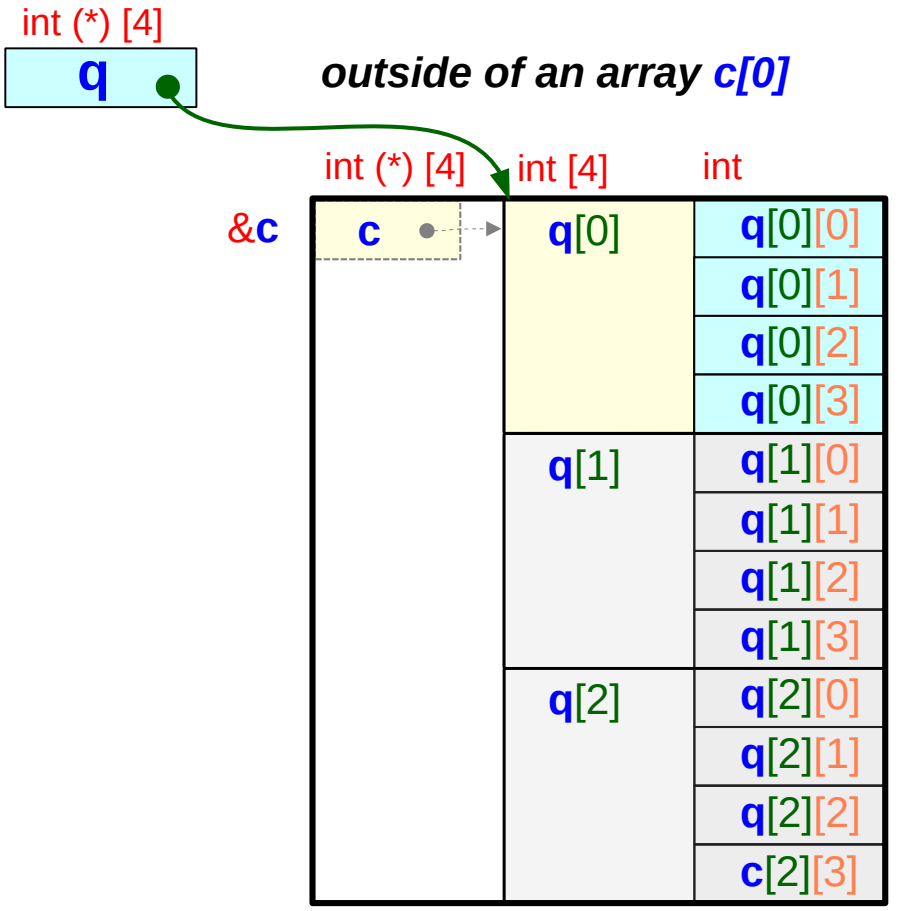
1-d array pointer

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

q = &c[0] ;

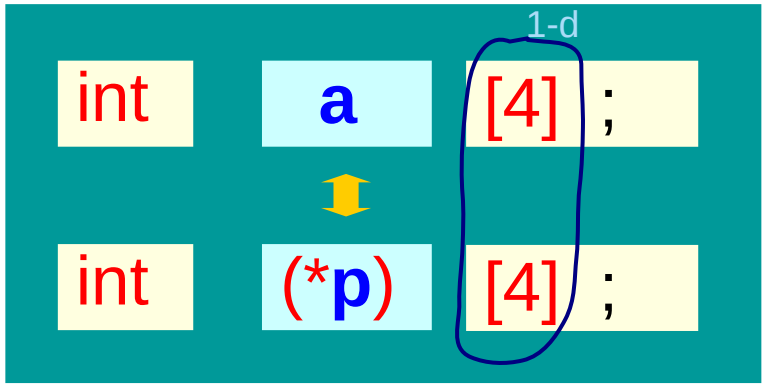
q ≡ c





1-d and 0-d array pointers to an 1-d array

1-d array pointer

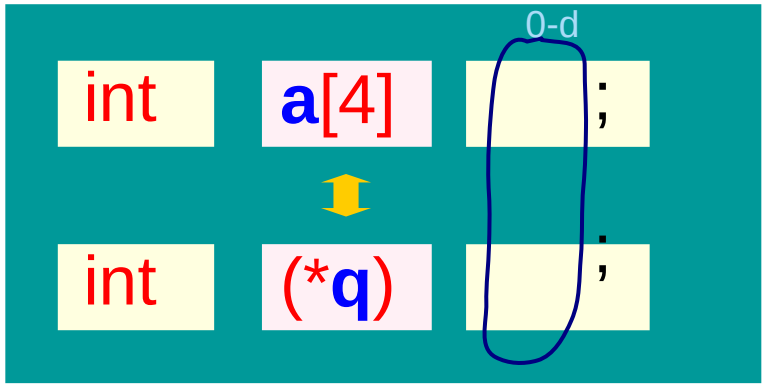


int (*) [4]

***p** ≡ **a**	equivalence
p = &**a**;	assignment

(***p**)[**i**] ≡ **a**[**i**] ≡ **p**[0][**i**]

0-d array pointer : int pointer



int (*)

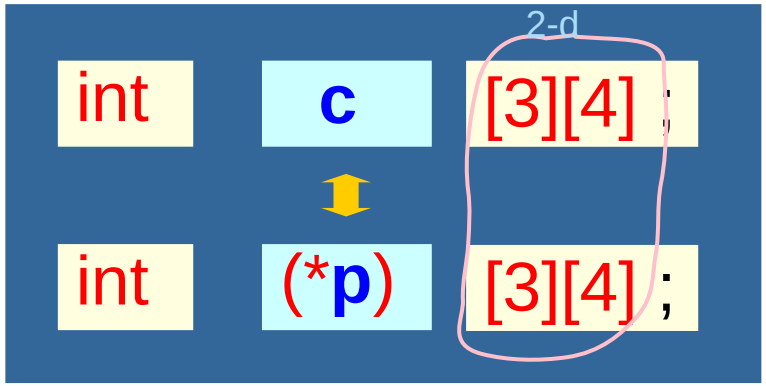
***q** ≡ **a**[0]	equivalence
q = **a**;	assignment

q[**i**] ≡ **a**[**i**] ≡ *(**q**+**i**)

among 4 elements a[0], a[1], a[2], a[3],
consider the first one a[0] = *a

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

2-d array pointer

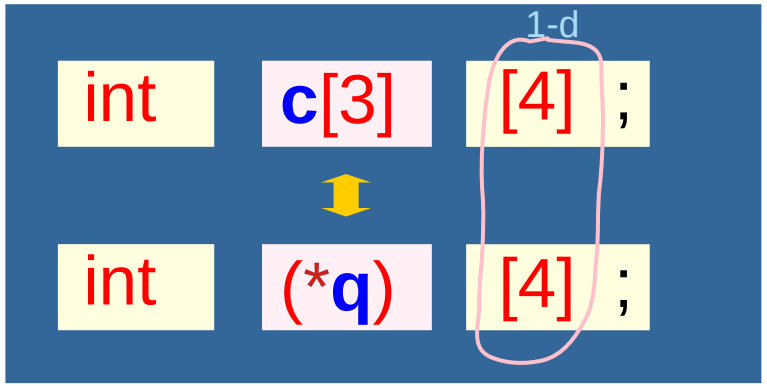


int (*) [3][4]

***p** ≡ **c**	equivalence
p = &**c**;	assignment

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

1-d array pointer



int (*) [4]

***q** ≡ **c**[0]	equivalence
q = **c**;	assignment

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

among 3 elements c[0], c[1], c[2],
consider the first one c[0] = *c

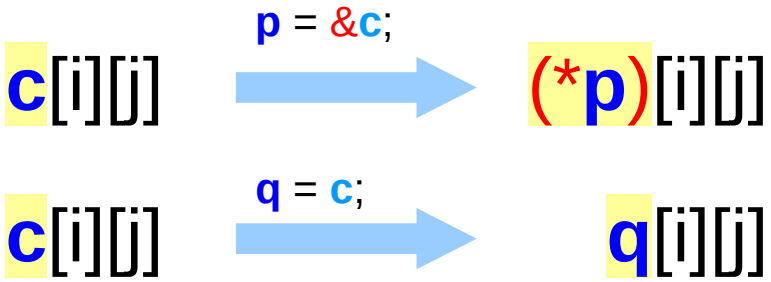
Array pointers to a **2-d** array and its sub-array

```
int   c   [3]  [4] ;
int (*p)  [3]  [4] = &c ;
int (  * q  )  [4] = &c[0] ;   (= c)
```

2-d array c

2-d array pointer p

1-d array pointer q



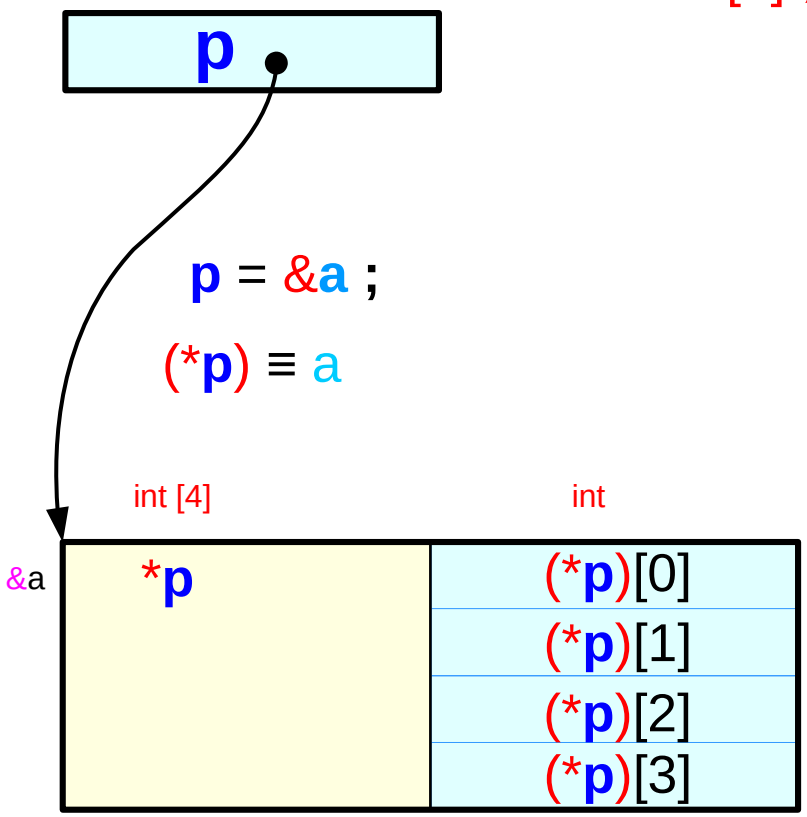
int [4]	array type
int (*) [4]	array pointer type
int *	integer pointer type
int **	integer double pointer type

Must point to an array type (array name)

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

int (*) [4]

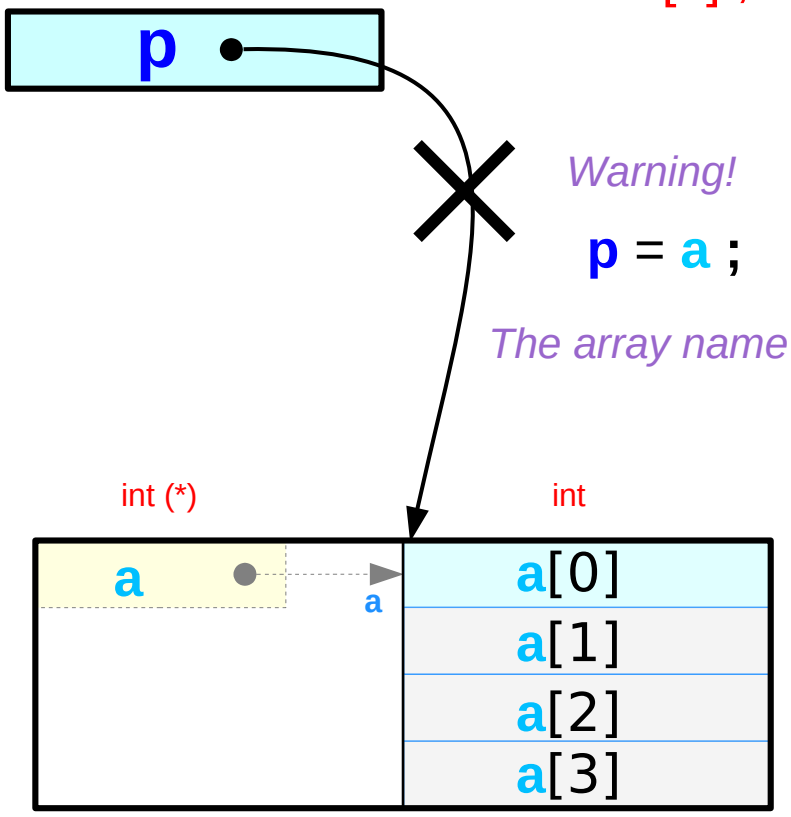
int a[4] ;



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

int (*) [4]

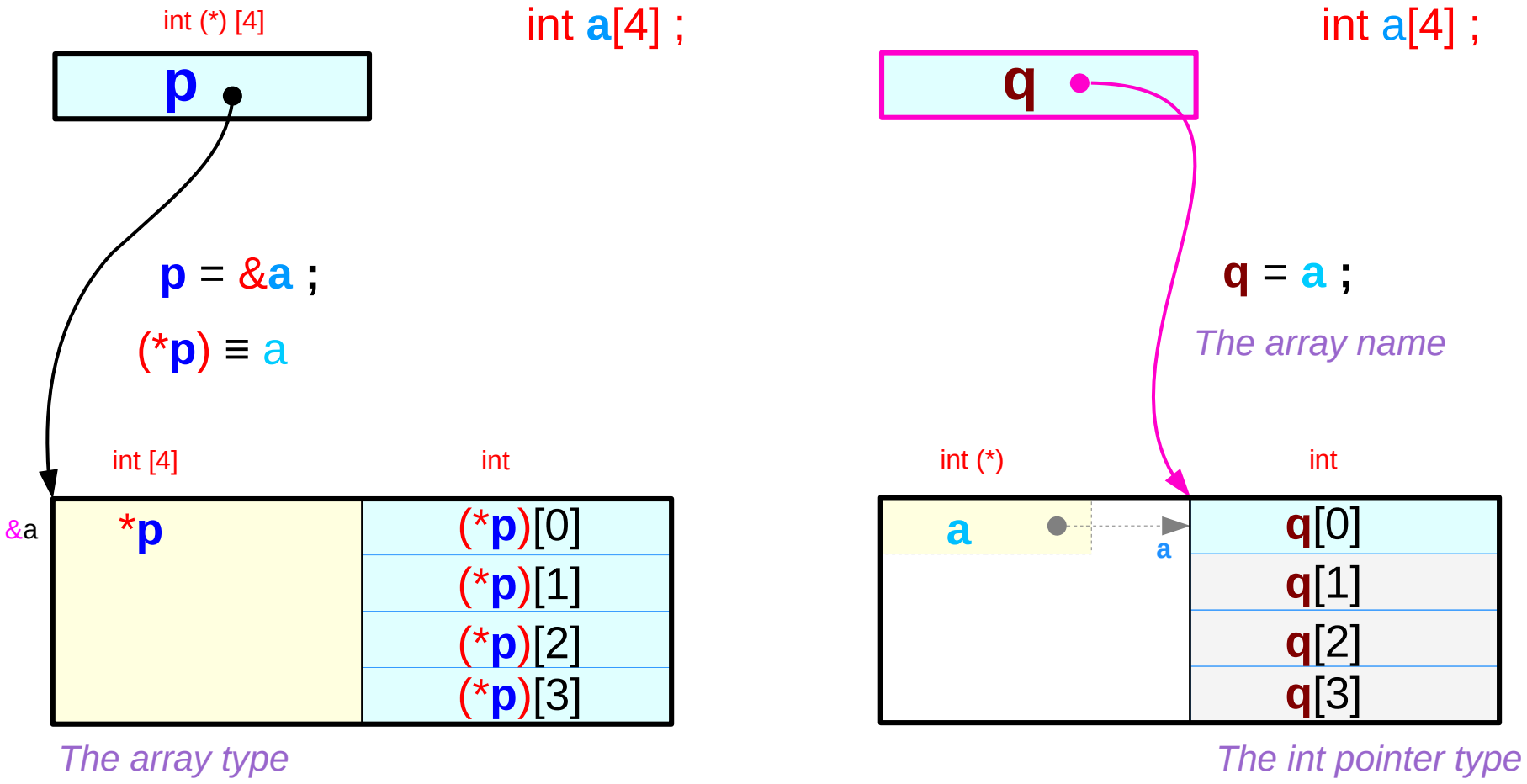
int a[4] ;



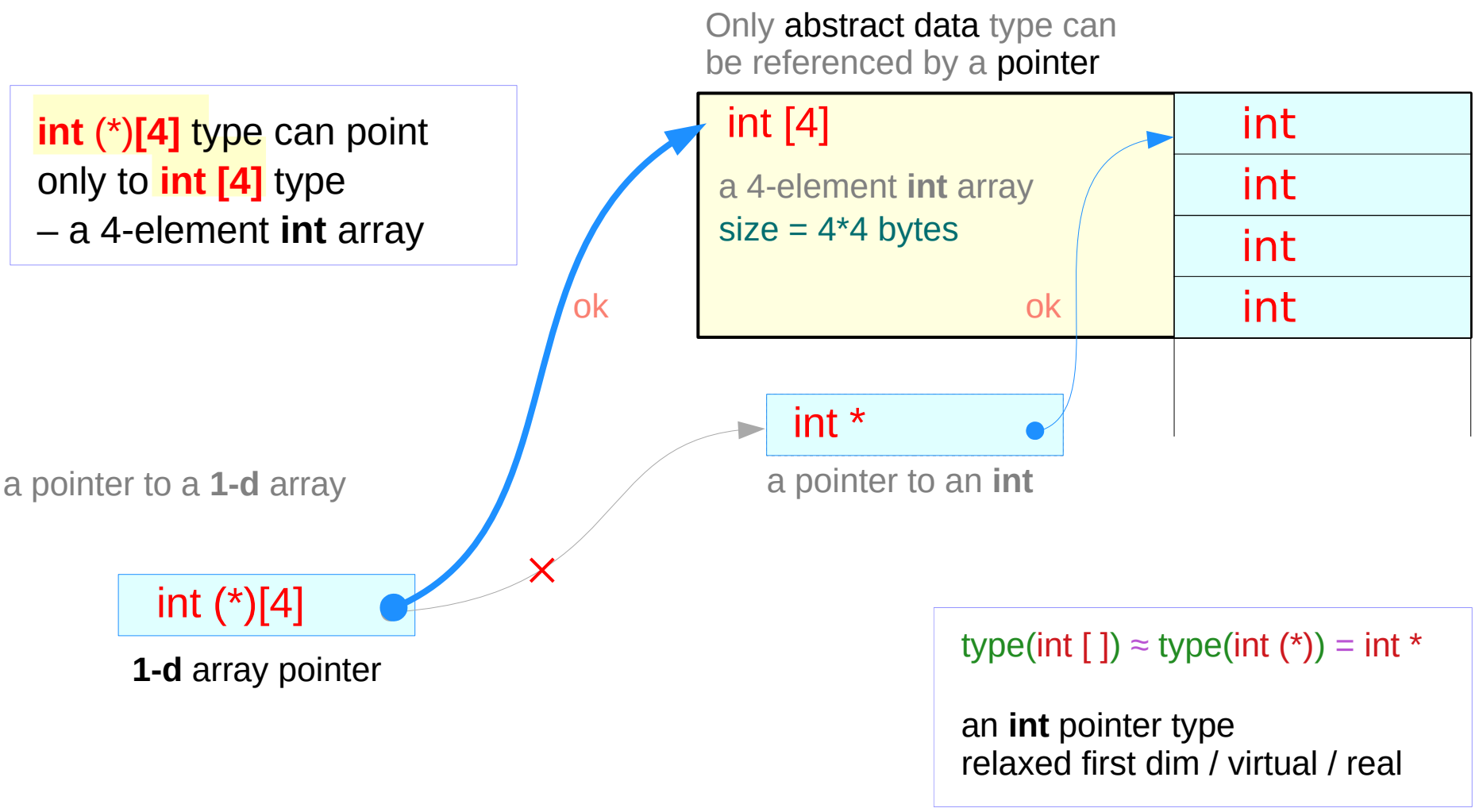
Integer pointer **p** vs. array pointer **q**

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

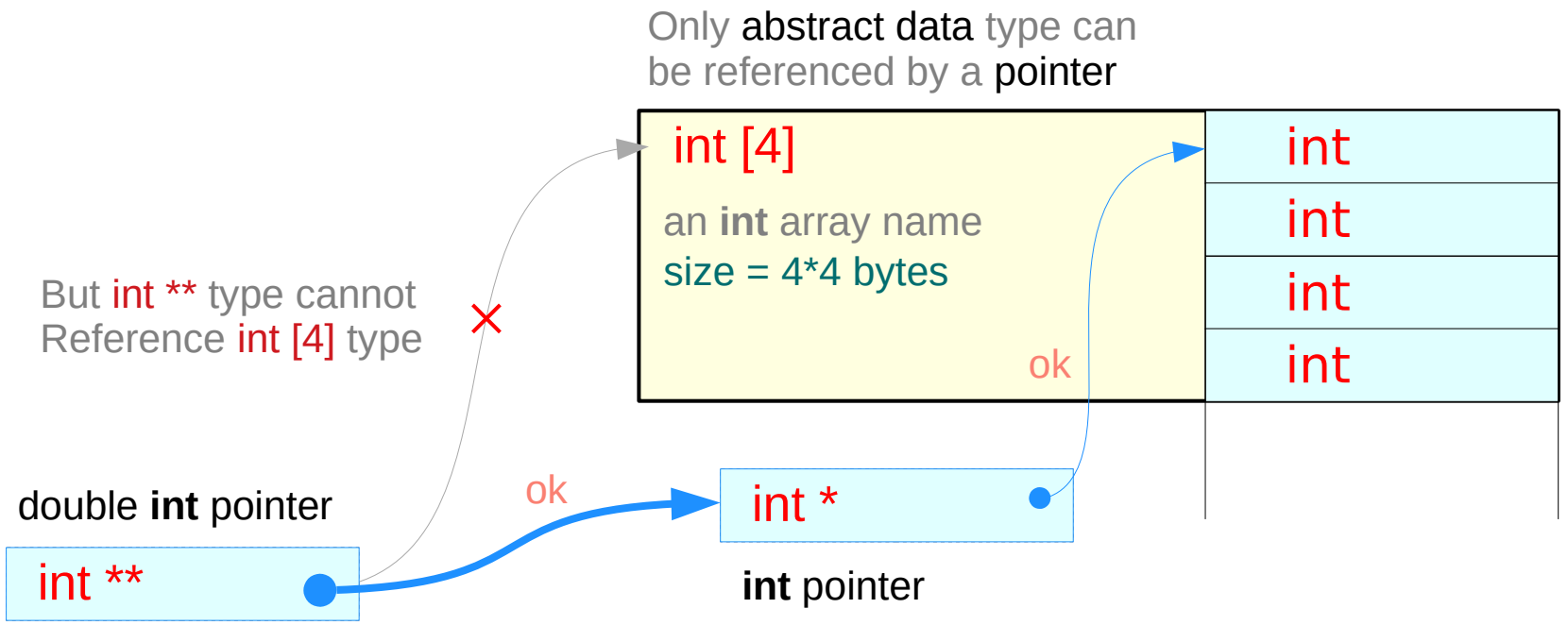
int *q ; **q = a ;**



Integer array pointer type – **int (*)[4]**



Double integer pointer type – **int ****

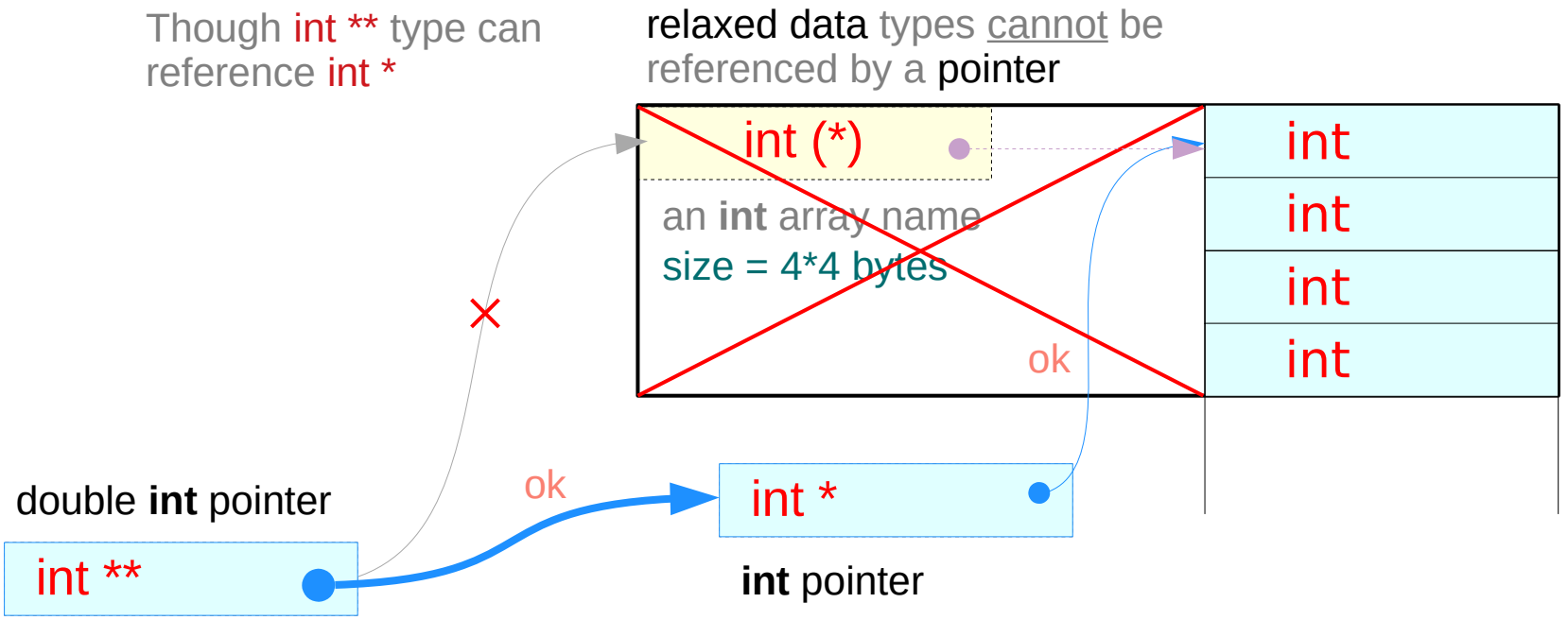


int ** type can point
only to **int *** type
– **int** pointer

type(int [ ]) ≈ type(int (*)) = int *

an **int** pointer type
relaxed first dim / virtual / real

Double integer pointer type – **int ****

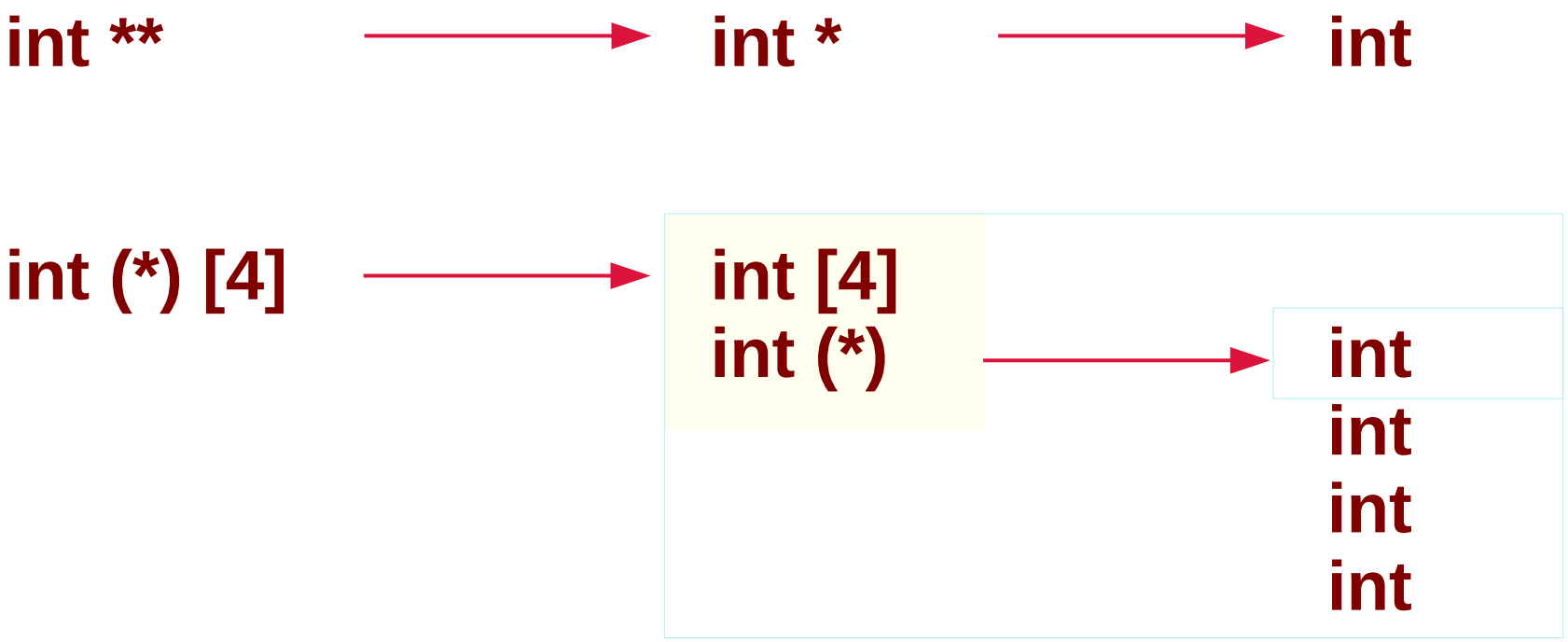


int ** type cannot point
either **int [4]** (abstract data)
nor **int (*)** (virtual pointer)

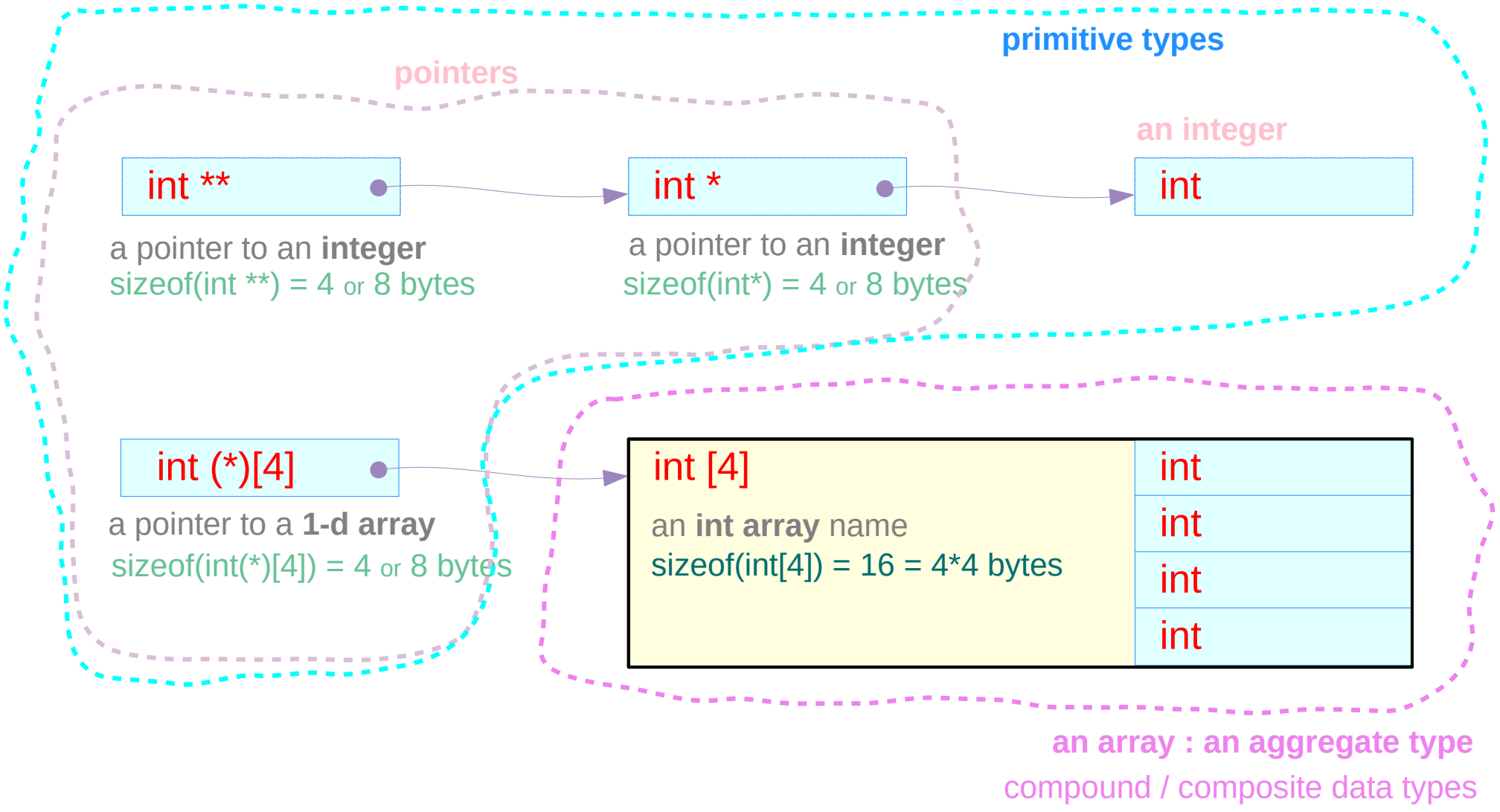
type(int [ ]) ≈ type(int (*)) = int *

an **int** pointer type
relaxed first dim / virtual / real

Pointer chains and nested pointers

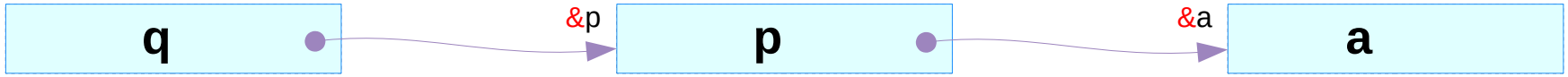


Types of integer pointers

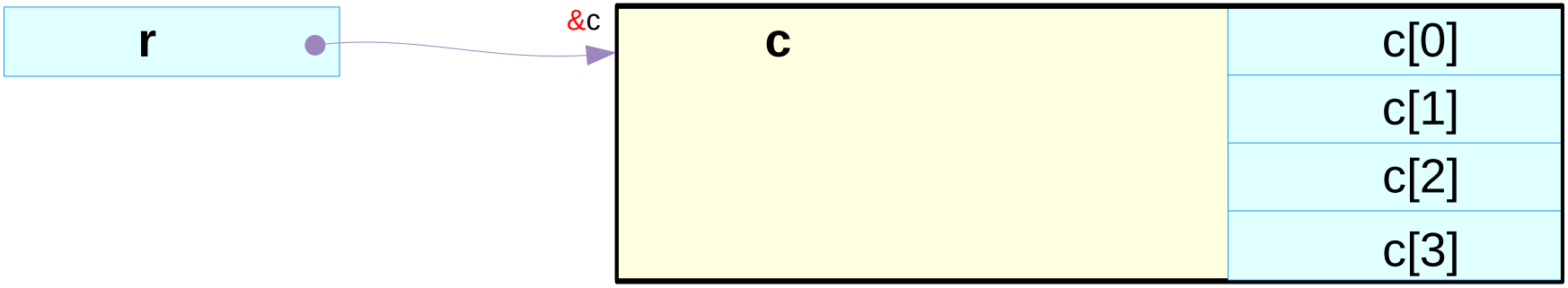


Variable declaration of integer pointers

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

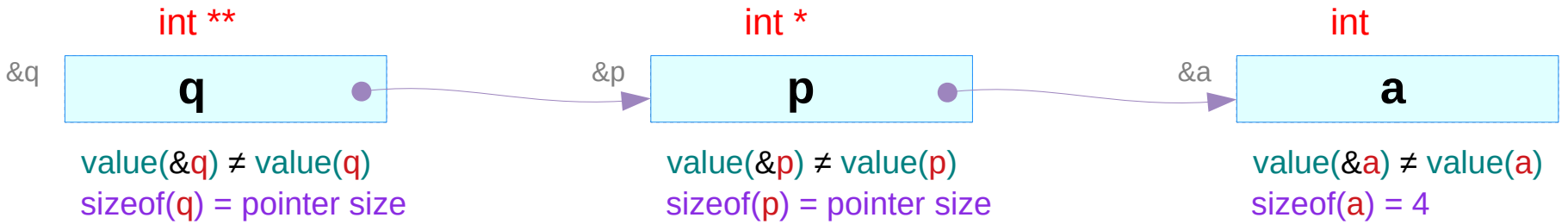


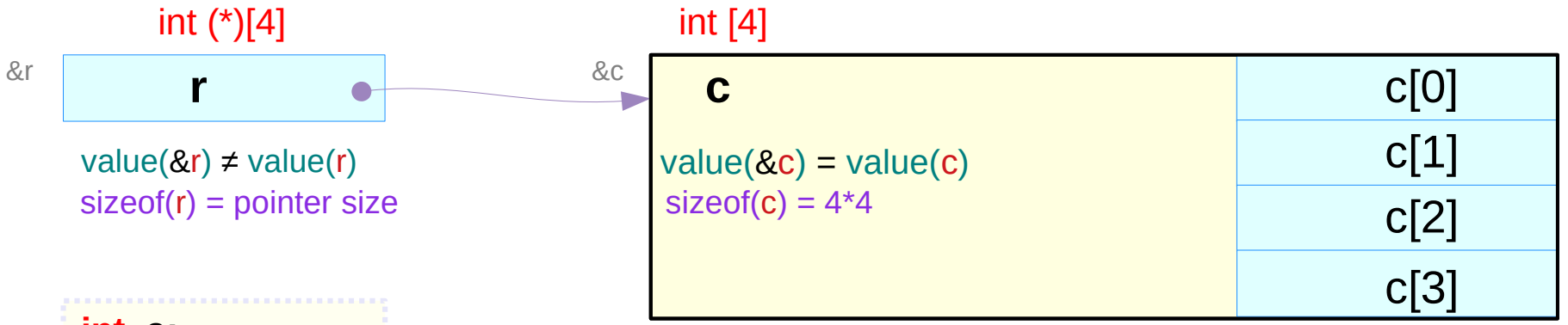
int (*r)[4] = &c; int c[4];



int (*)	**virtual pointers** cannot be referenced
int [4]	only **abstract data** can be referenced

Types, Sizes, and Values





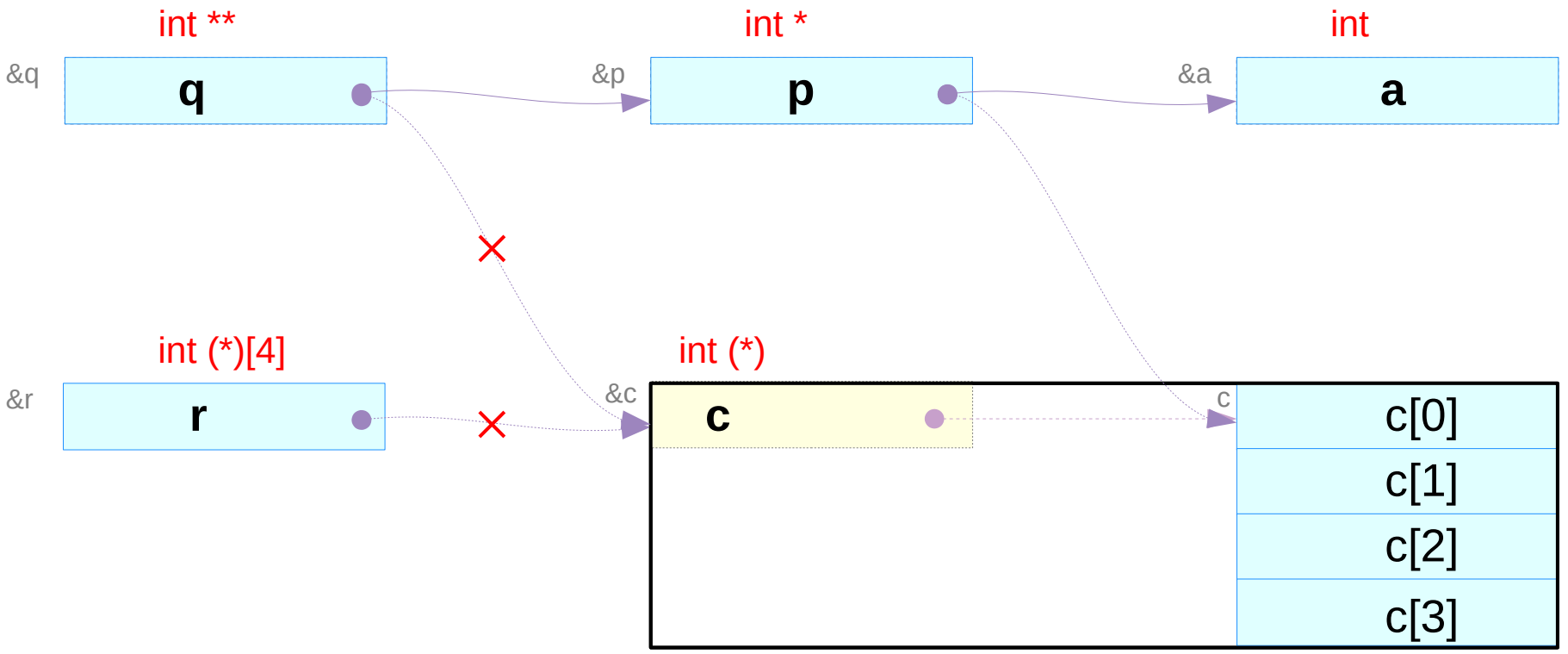

```
int  a;
int *p = &a;
int *q = &p;

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

type(**int [4]**) ⊂ type(**int [ ]**) ≈ type(**int (*)**) ≡ **int ***

abstract data — **relaxed data** — **0-d array pointer** — **int pointer**

Array as a pointer type



int (*)	**virtual pointers** cannot be referenced
int [4]	only **abstract data** can be referenced

int * an integer pointer

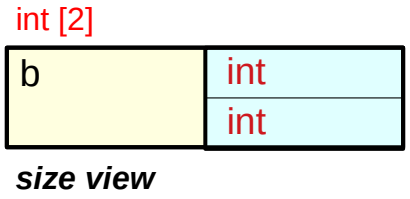
int [2] a **1-d** array with 2 integer elements

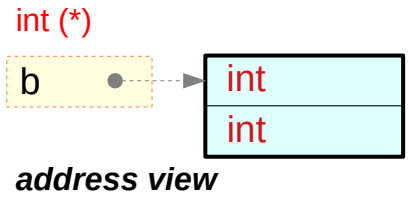
int [3] a **1-d** array with 3 integer elements

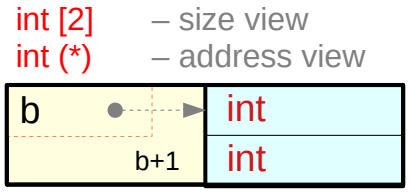
Integer pointer and array types – **int *, int [2], int [3]**

int *a;

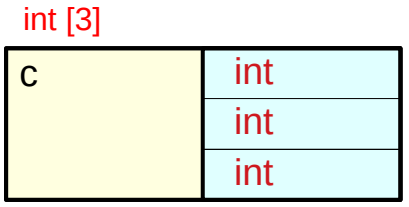
int b[2];

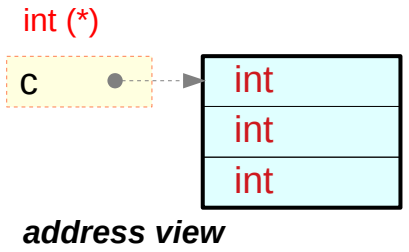


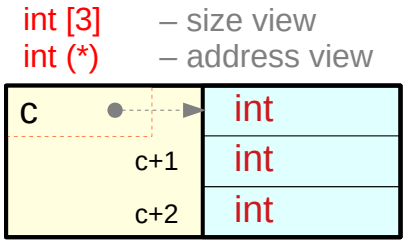




int c[3];

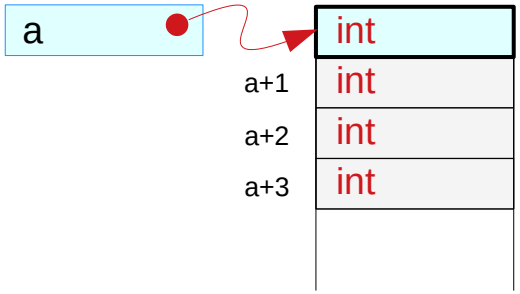






Incrementing pointers – **int *, int [2], int [3]**

int *a;

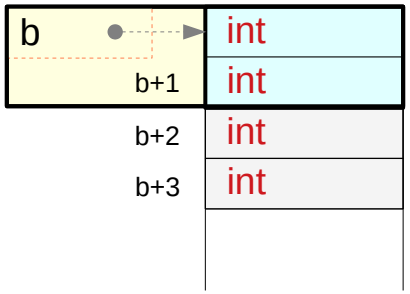


a[0] = *a
a[1] = *(a+1)
a[2] = *(a+2)
a[3] = *(a+3)

syntactically legitimate

programmers must ensure their validity

int b[2]

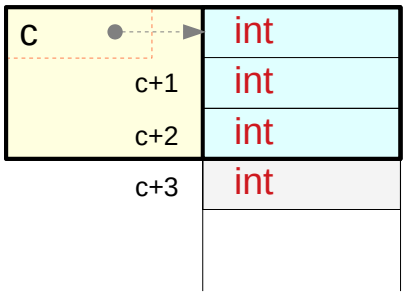


b[0] = *b
b[1] = *(b+1)
b[2] = *(b+2)
b[3] = *(b+3)

syntactically legitimate

programmers must ensure their validity

int c[3];



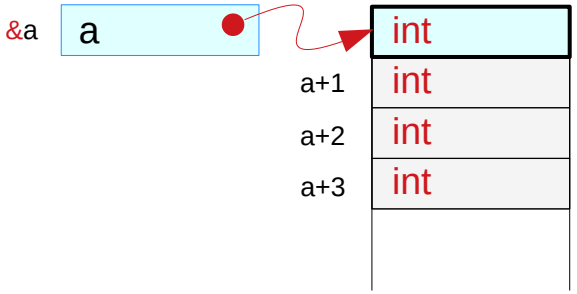
c[0] = *c
c[1] = *(c+1)
c[2] = *(c+2)
c[3] = *(c+3)

syntactically legitimate

programmers must ensure their validity

Types and sizes – int *, int [2], int [3]

int *a;



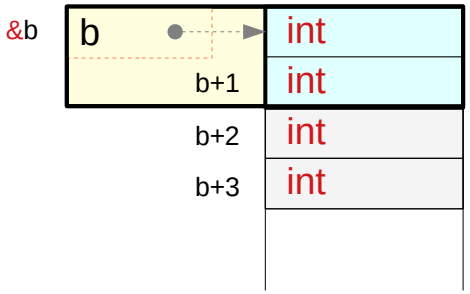
type(&a) = int **
type(a) = int *

type(*a) = int

value(&a) ≠ value(a)

sizeof(a)
= pointer size
= sizeof(int *)

int b[2]



type(&b) = int (*) [2]
type(b) = int (*) gcc printing type
int [2]

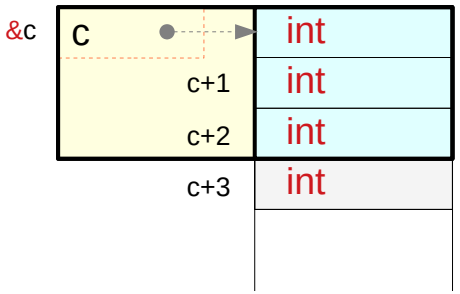
type(*b) = int

value(&b) = value(b)

sizeof(b)
= sizeof(*b) * 2
= sizeof(int) * 2

&b and b evaluate
the same address
but have different types
and also different sizes

int c[3];



type(&c) = int (*) [3]
type(c) = int (*) gcc printing type
int [3]

type(*c) = int

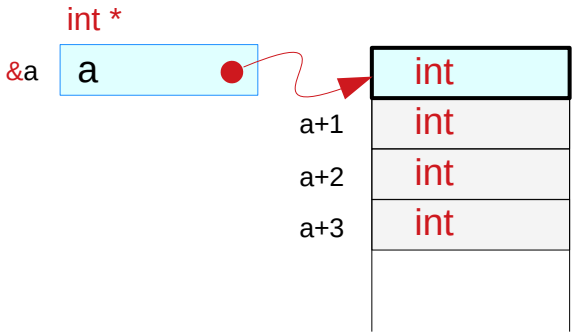
value(&c) = value(c)

sizeof(c)
= sizeof(*c) * 3
= sizeof(int) * 3

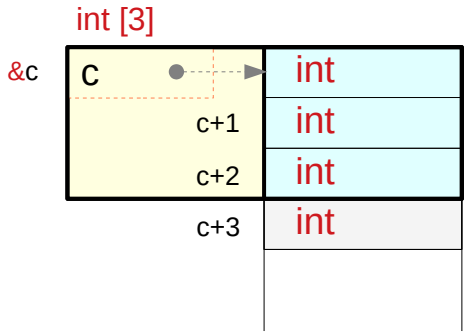
&c and c evaluate
the same address
but have different types
and also different sizes

Real pointer and virtual pointer types – **int *, int [3]**

int *a;



int c[3];



sizeof (a) = pointer size	int *	sizeof (c) = sizeof(*c) * 3	int [3]
value(&a) ≠ value(a)	int *	value(&c) = value(c)	int (*)

the address of pointer variable a is
not equal to the pointed address

real memory location for a

the starting address of array variable c
is equal to the address of the 1st element

no actual memory location for c

type(a) = int *

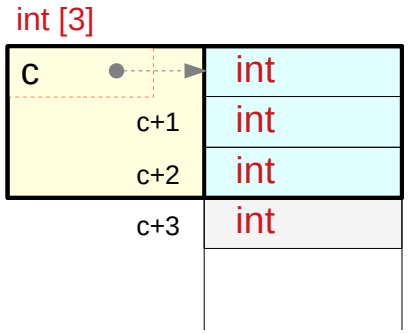
type(&a) = int **

type(c) = int (*)
int [3]
gcc printing type

type(&c) = int (*) [3]

Virtual pointer types of an array– **int [3]**

int c[3];



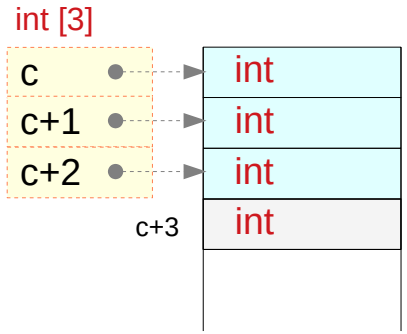
sizeof (c) = sizeof(int) * 3

value(&c) = value(c)

type(c) = int *

type(&c) = int (*) [3]

int c[3];



sizeof (c) = sizeof(*c) * 3 … *leading element*
sizeof (c+1) = pointer size
sizeof (c+2) = pointer size

value(&c) = value(c) … *leading element*
value(c+1) = value(c) + sizeof(*c) *1
value(c+2) = value(c) + sizeof(*c) *2

type(c) = int *
type(c+1) = int *
type(c+2) = int *

type(&c) = int (*) [3]

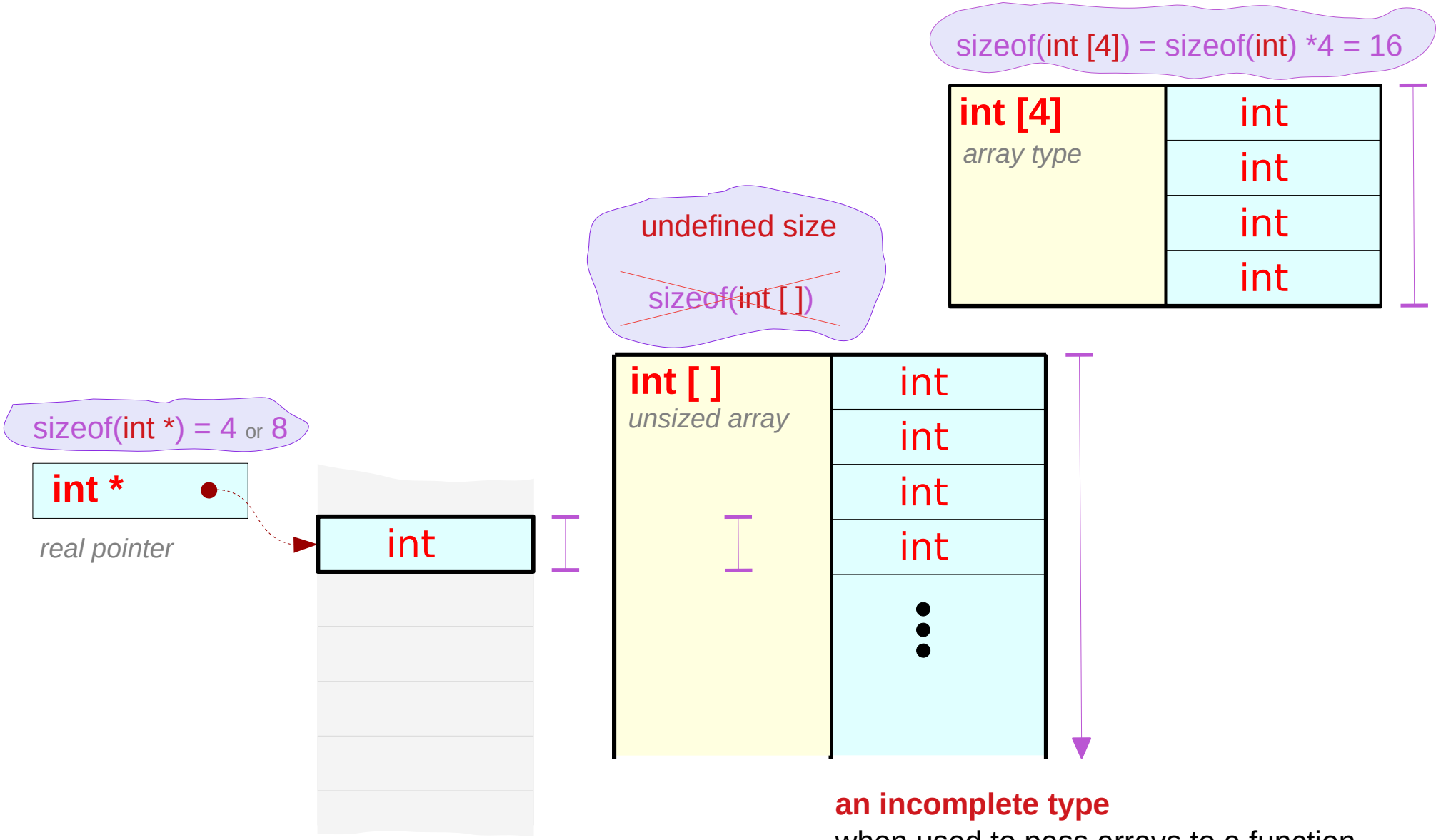
int [4] a **1-d** 4-element integer array

int [ ] a **1-d** unsized integer array

int (*) a **0-d** integer array pointer (**int ***)

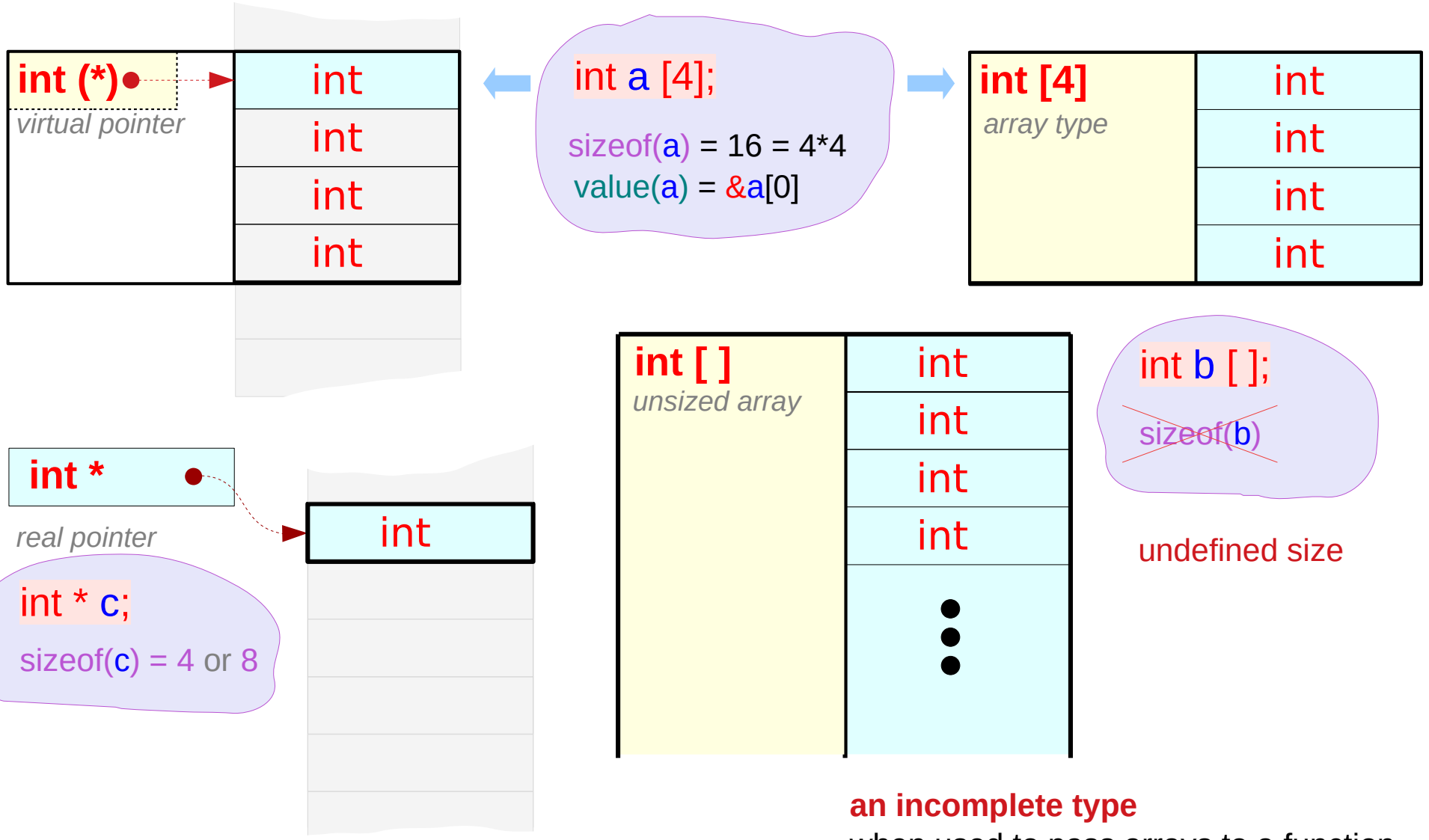
int (*) [4] a **1-d** integer array pointer

Sizes of **int [4]**, **int [ ]**, **int *** types



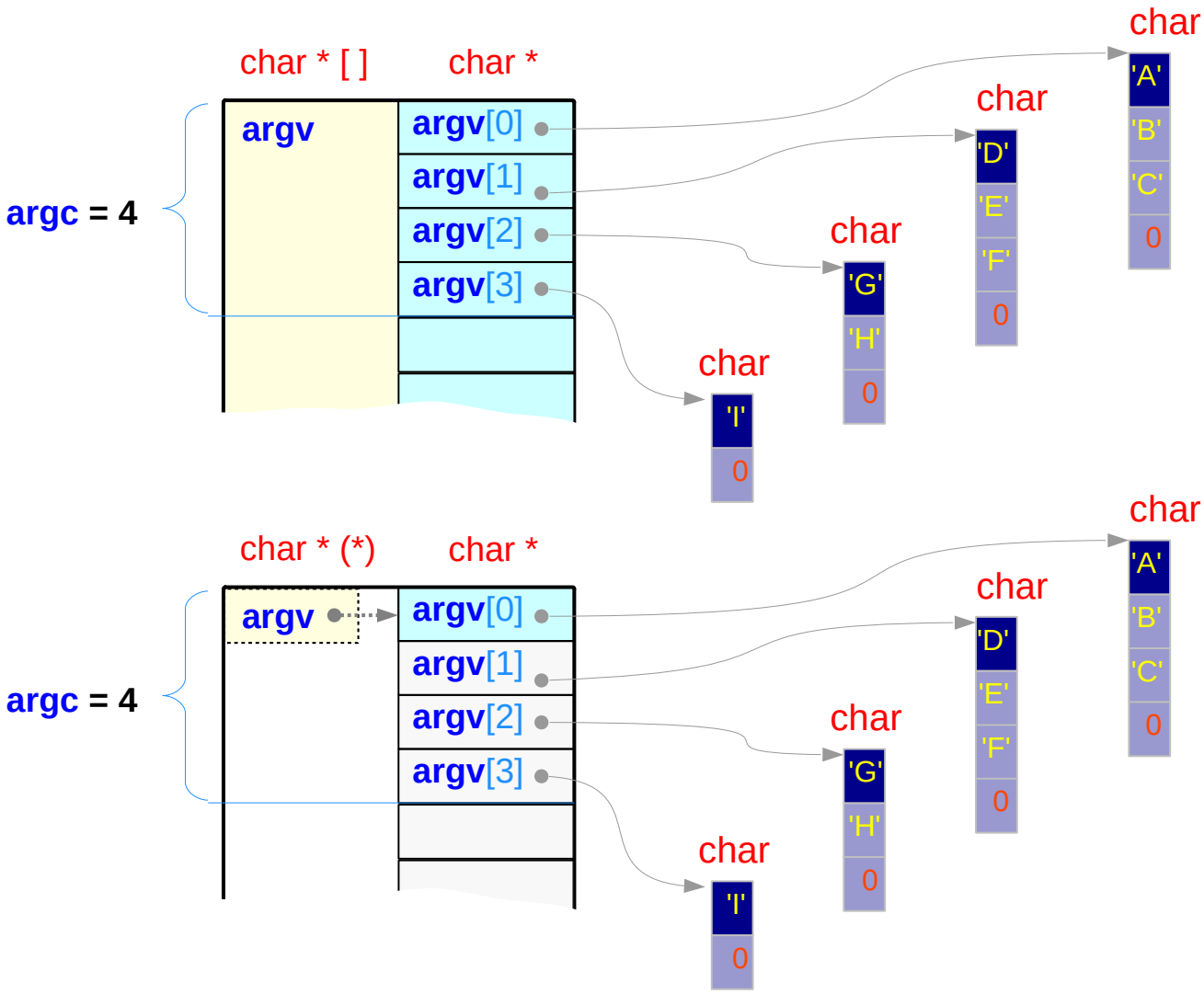
an incomplete type
when used to pass arrays to a function

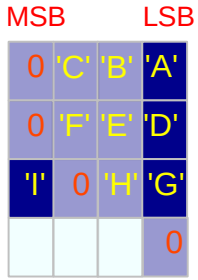
Sizes of variables of **int [4]**, **int [ ]**, **int *** types



an incomplete type
when used to pass arrays to a function

(int argc, char * argv[ ]) example





char * [ ] relaxed 1st dimension

char * (*) array pointer

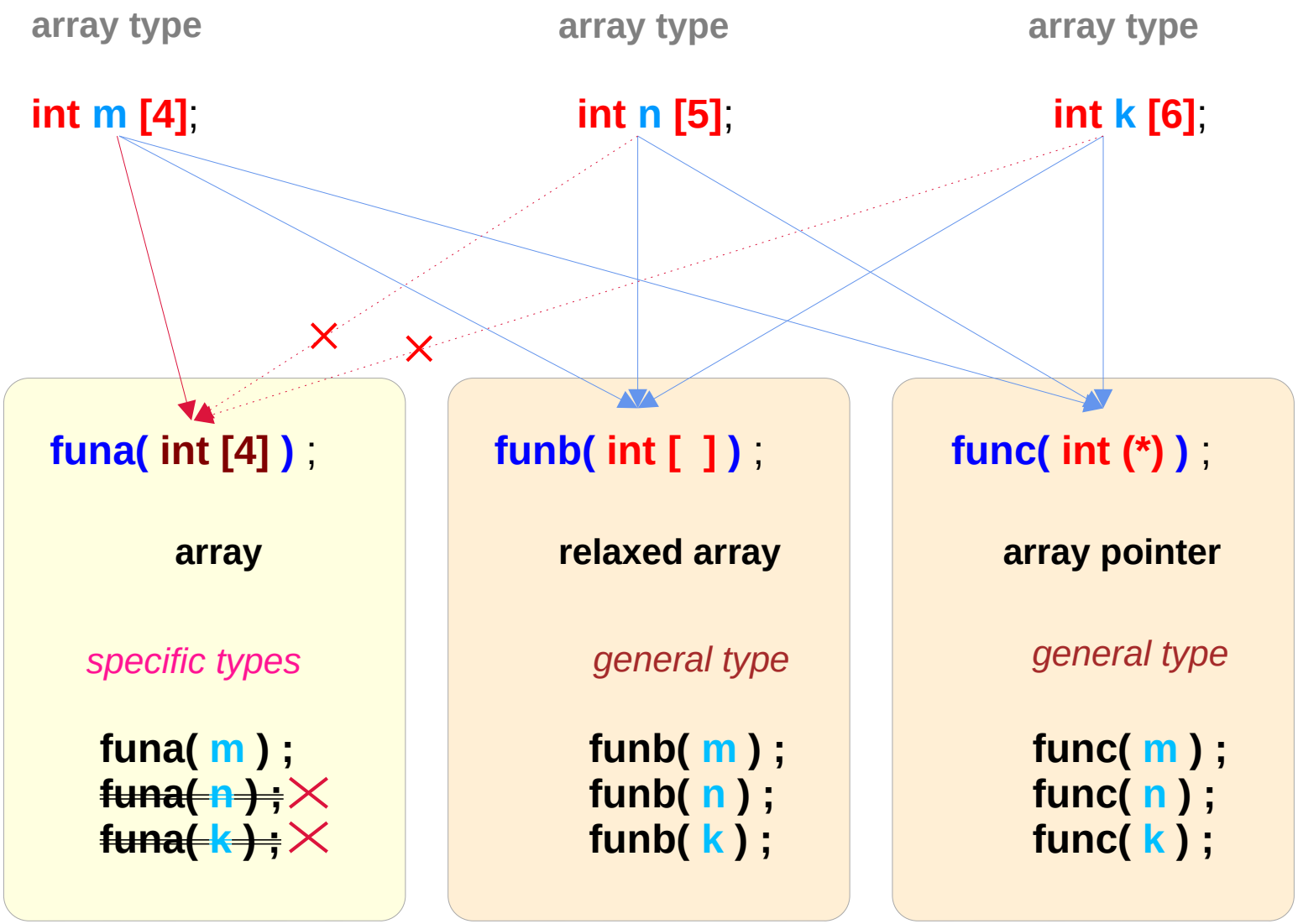
char * *

Relaxing the 1st dimension of an array

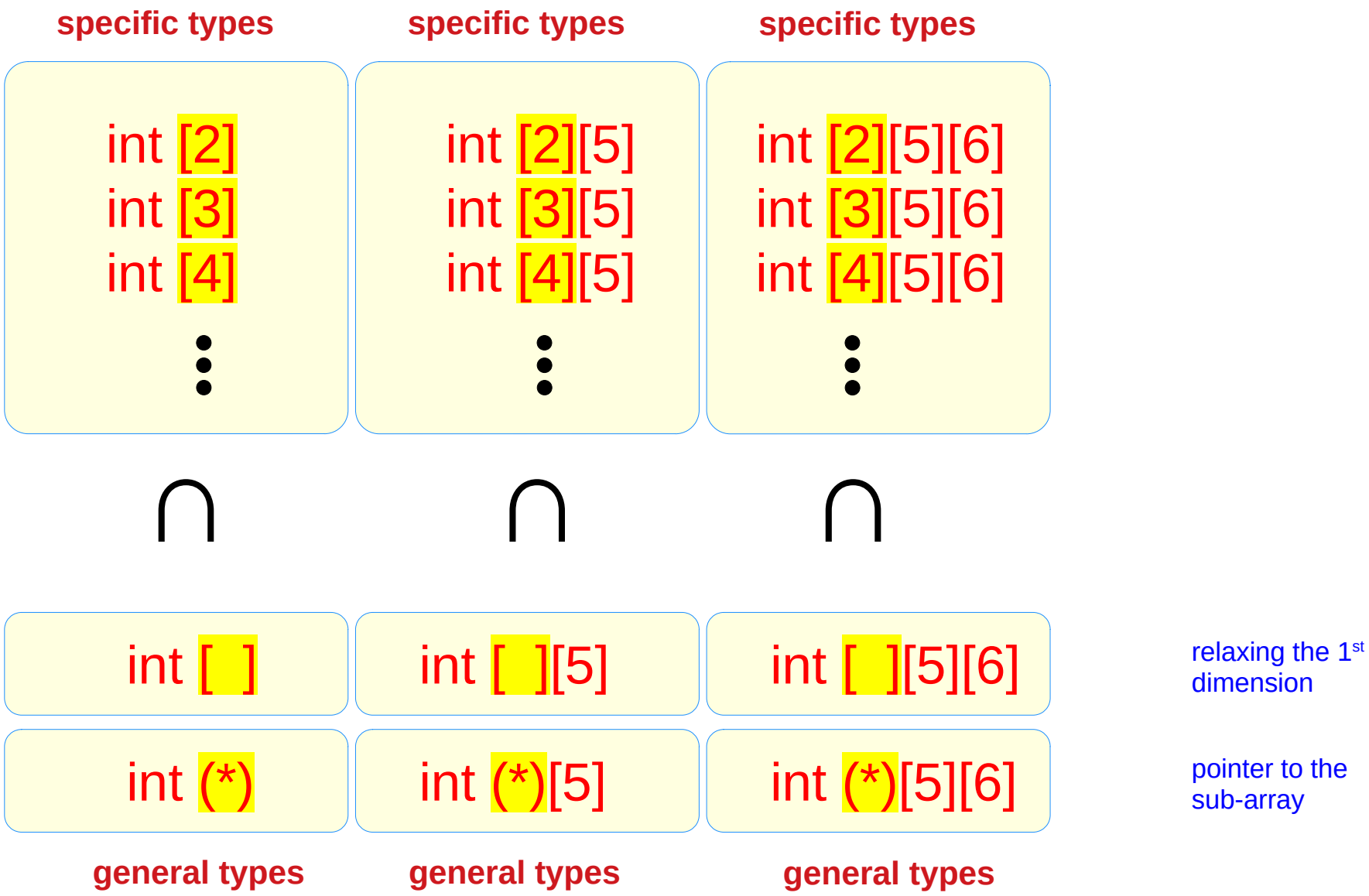
Multi-dimensional array types

array types	function calls	possible function prototypes		
int a [4];	funa(a);	funa(int [4]) ;	array type	*specific type*
		funa(int [ ]) ;	relaxed array	*general type*
		funa(int (*)) ;	array pointer	*general type*
int b [4][5];	funb(b);	funb(int [4][5]) ;	array type	*specific type*
		funb(int [ ][5]) ;	relaxed array	*general type*
		funb(int (*)[5]) ;	array pointer	*general type*
int c [4][5][6];	func(c);	func(int [4][5][6]) ;	array type	*specific type*
		func(int [ ][5][6]) ;	relaxed array	*general type*
		func(int (*)[5][6]) ;	array pointer	*general type*

Multi-dimensional array types



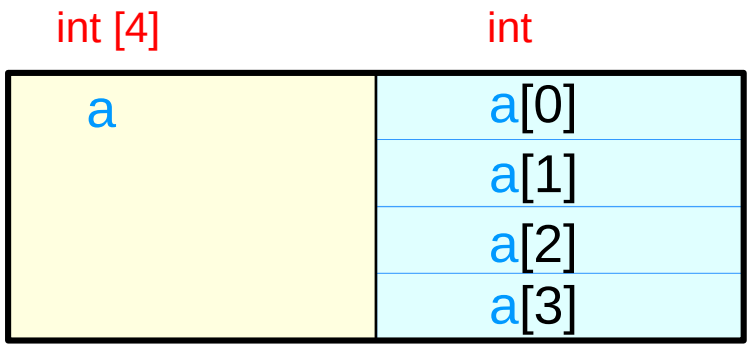
Super types and sub types

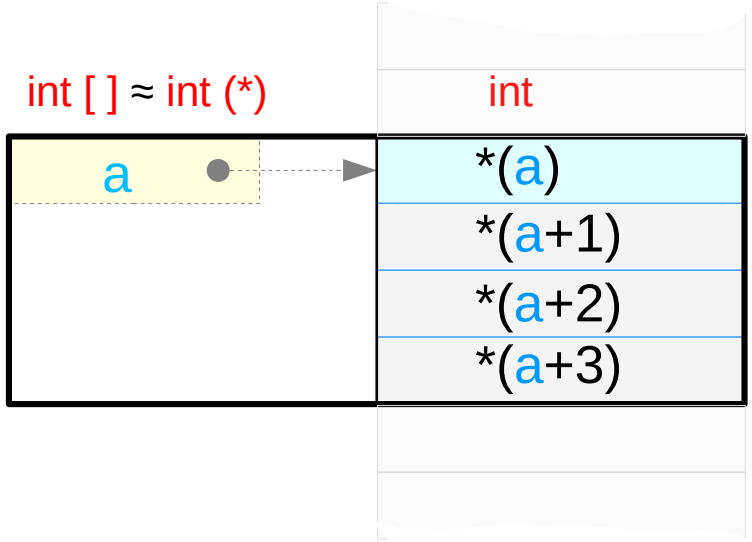


Relaxing array types

int [3][4][5] **3-d** array	int [ ][4][5] the 1st dimension relaxed	int (*)[4][5] **2-d** array pointer
int [4][5] **2-d** array	int [ ][5] the 1st dimension relaxed	int (*)[5] **1-d** array pointer
int [5] **1-d** array	int [ ] the 1st dimension relaxed	int (*) **0-d** array pointer
	undefined size	pointer size

Passing an individual element by reference





int [4] ⊂

int [ ]

≀≀

int (*)

|||

int *

relaxing the 1st dimension

pointer to a **0-d array**

pointer to an **integer**

- Viewing an **array** as a **pointer**

- Viewing a **pointer** as an **array**

int **a**[4] ;

int (*)

1-d array **a**

view **a** as
a **0-d** array pointer

virtual pointer
- no real memory location
- constraints :
  value(&**a**) = value(**a**)

int (***p**) ;

int [N]

0-d array pointer **p**

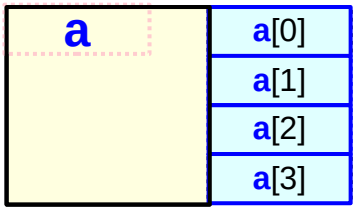
view **p** as
a **1-d** array

N is not fixed to 4

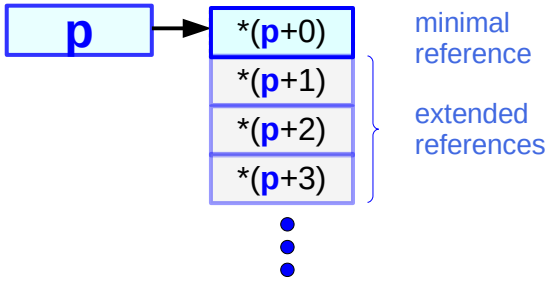
sizeof(**p**) is not
the size of the array
but the size of a
pointer variable

Array **a** and array pointer **p**

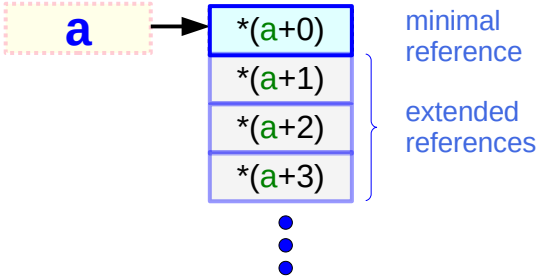
int **a**[4] ; **1-d** array **a**



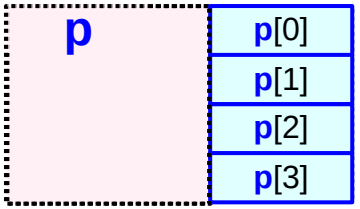
int (***p**) ; **0-d** array pointer **p**



int (*) **a** as a **0-d** array pointer



int [N] **p** as a **1-d** array



Array **a** and array pointer **p**

int **a**[4] ; **1-d** array **a**

- sizeof(**a**) = an array size
  = 4 * 4 bytes
- no. of elements = fixed
  = 4

int (*p) ; **0-d** array pointer **p**

- sizeof(**p**) = a pointer size
  = 4 / 8 bytes
- no. of elements = not fixed
  = at least 1

int (*) **a** as a **0-d** array pointer

a is not a real pointer

- sizeof(**a**) = an array size
- value(a) = value(&**a**)

int [N] **p** as a **1-d** array

p is not a real array

- sizeof(**p**) ≠ an array size
  = a pointer size
- value(**p**) ≠ value(&**p**)
  = allocated address

Determining types of sub-arrays

from the declaration of an array

Types of array names

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

a is the name of the **1-d** array **int [4]**

sizeof(**a**) = 4 * 4

[3] is declared;
[0], [1], [2] are used

```
int  c [3]  [4];
```

c[i] is the name of the **1-d** subarray **int [4]**

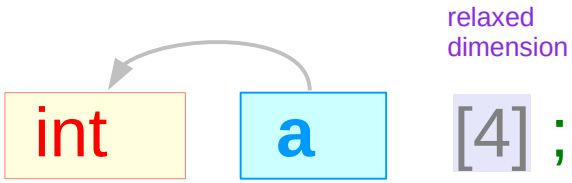
sizeof(**c[i]**) = 4 * 4

```
int  c  [3]  [4]
```

c is the name of the **2-d** array **int [3][4]**

sizeof(**c**) = 3 * 4 * 4

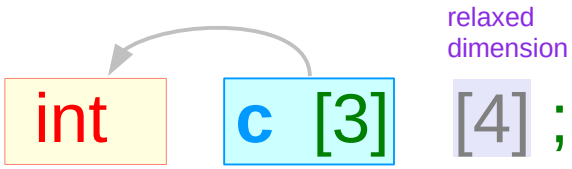
Values of array names



the value of **a** is the starting address of an array with 4 elements of **int** type — **int (*)**

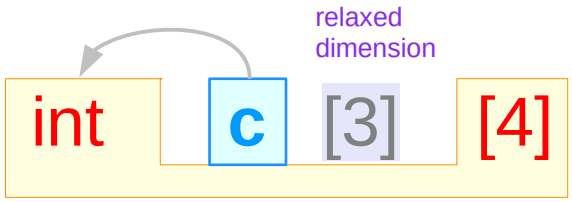
a: pointer to the first element — **a = &a[0]**

[3] is declared;
[0], [1], [2] are used



each value of **c[i]** is the starting address of an array with 4 elements of **int** type — **int (*)**

c[i]: pointer to the first element — **c[i] = &c[i][0]**



the value of **c** is the starting address of an array with 3 elements of **int [4]** type — **int (*) [4]**

c: pointer to the first element — **c = &c[0]**

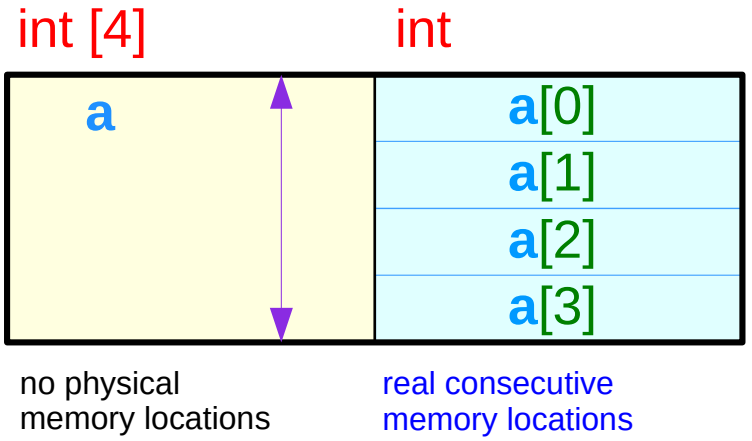
Array and pointer types in a **1-d** array

int a [4]

a **1-d array**

type : int [4]

size : 4 * 4



relaxed dimension

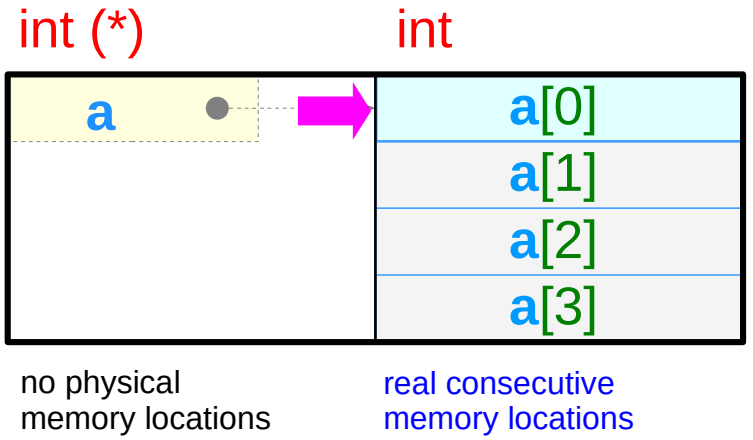
int a [4]

a **0-d array pointer**

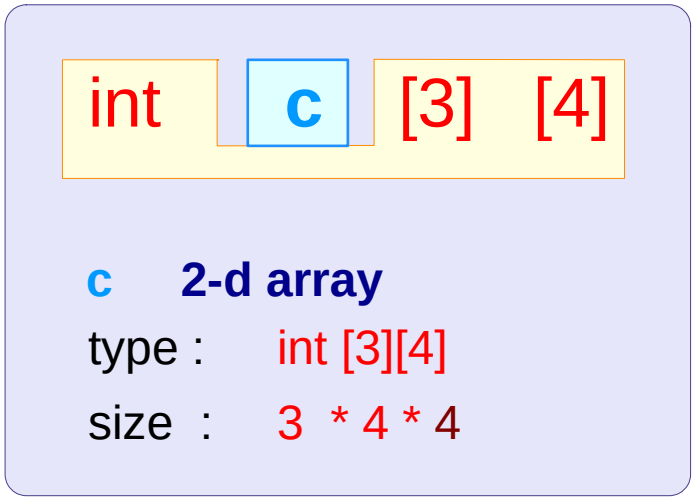
type : int (*)

value : &a[0]

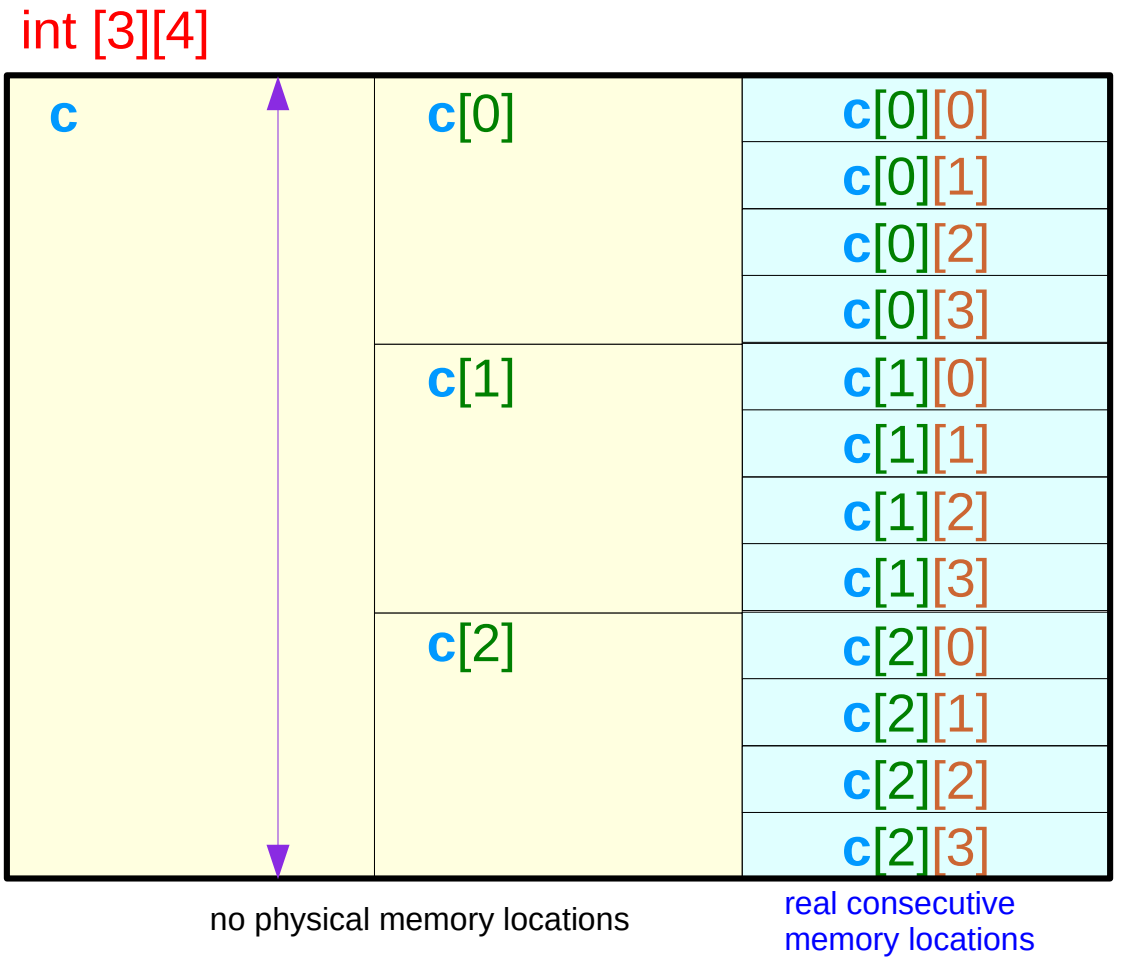
a points to the 1st **int** element
there are 4 **int** elements



2-d array type



&c

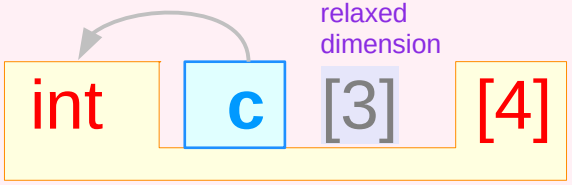


1-d array pointer type

c **2-d array**

type : int [3][4]

size : 3 * 4 * 4

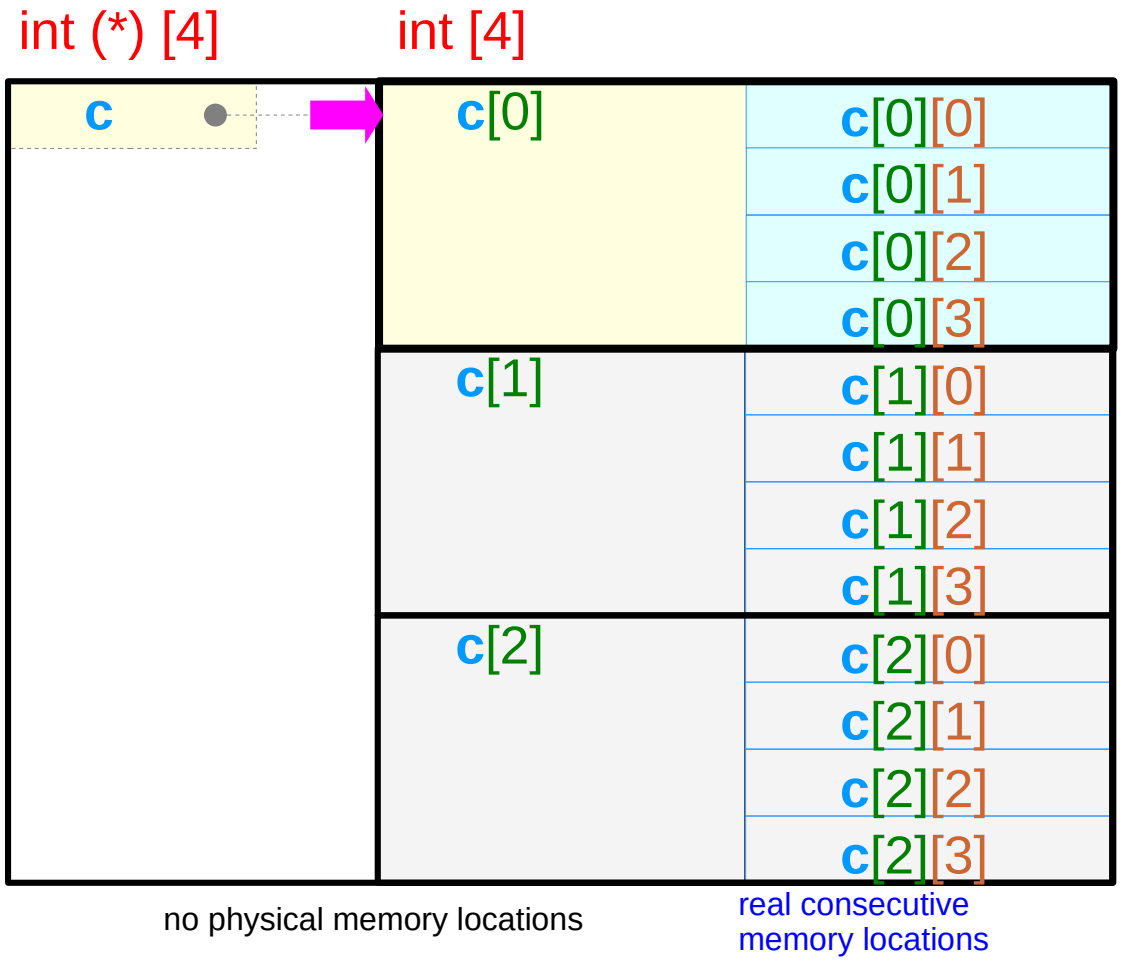


c **1-d array pointer**

type : int (*) [4]

value : c = &c[0][0]

c points to the 1st **int [4]** element
There are 3 **int [4]** elements



1-d array type

int c [3] [4]

c[i] 1-d array

type : int [4]

size : 4 * 4

int c [3] [4]

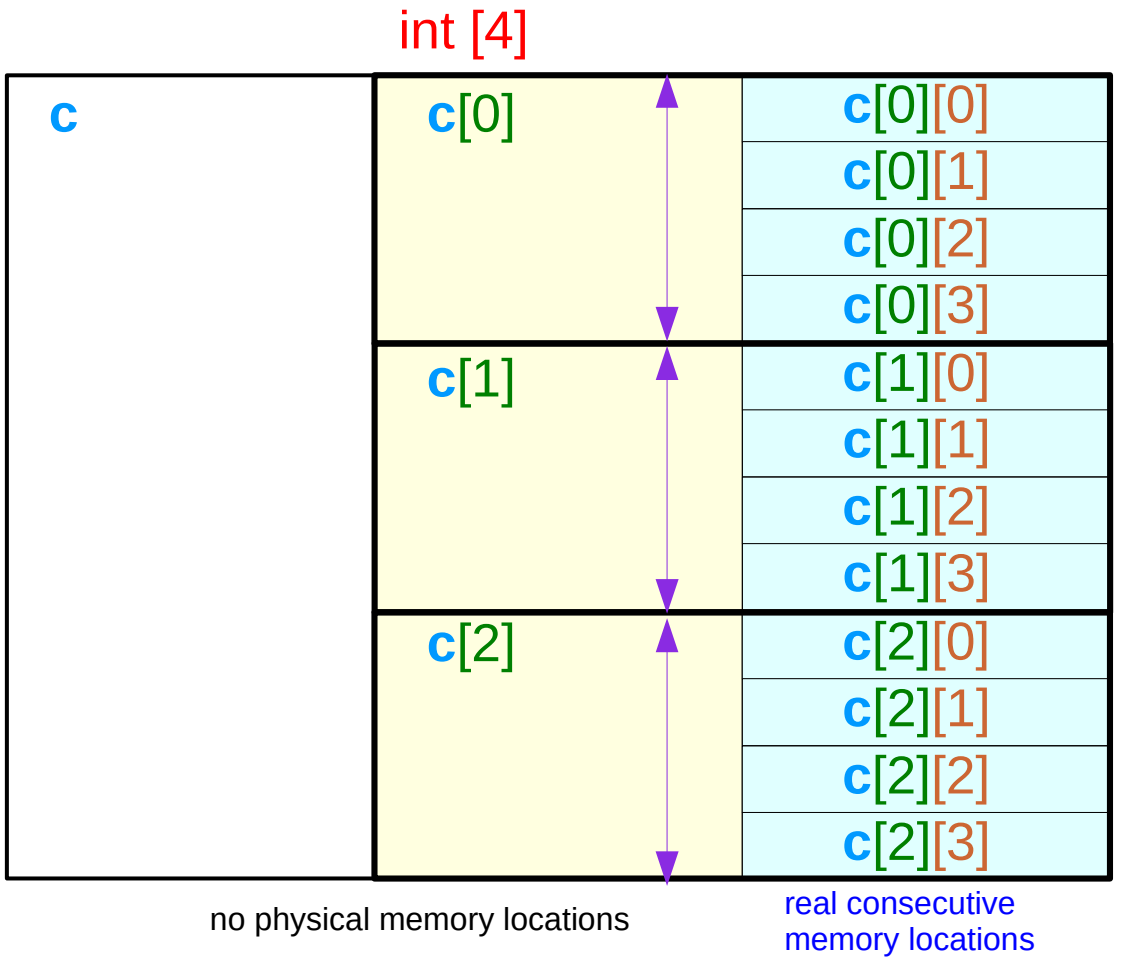
relaxed dimension

c[i] 0-d array pointer

type : int (*)

value : c[*i*] = &c[*i*][0]

c[i] points to the 1st **int** element
There are 4 **int** elements



0-d array pointer type

int c [3] [4]

c[i] 1-d array

type : int [4]

size : 4 * 4

relaxed dimension

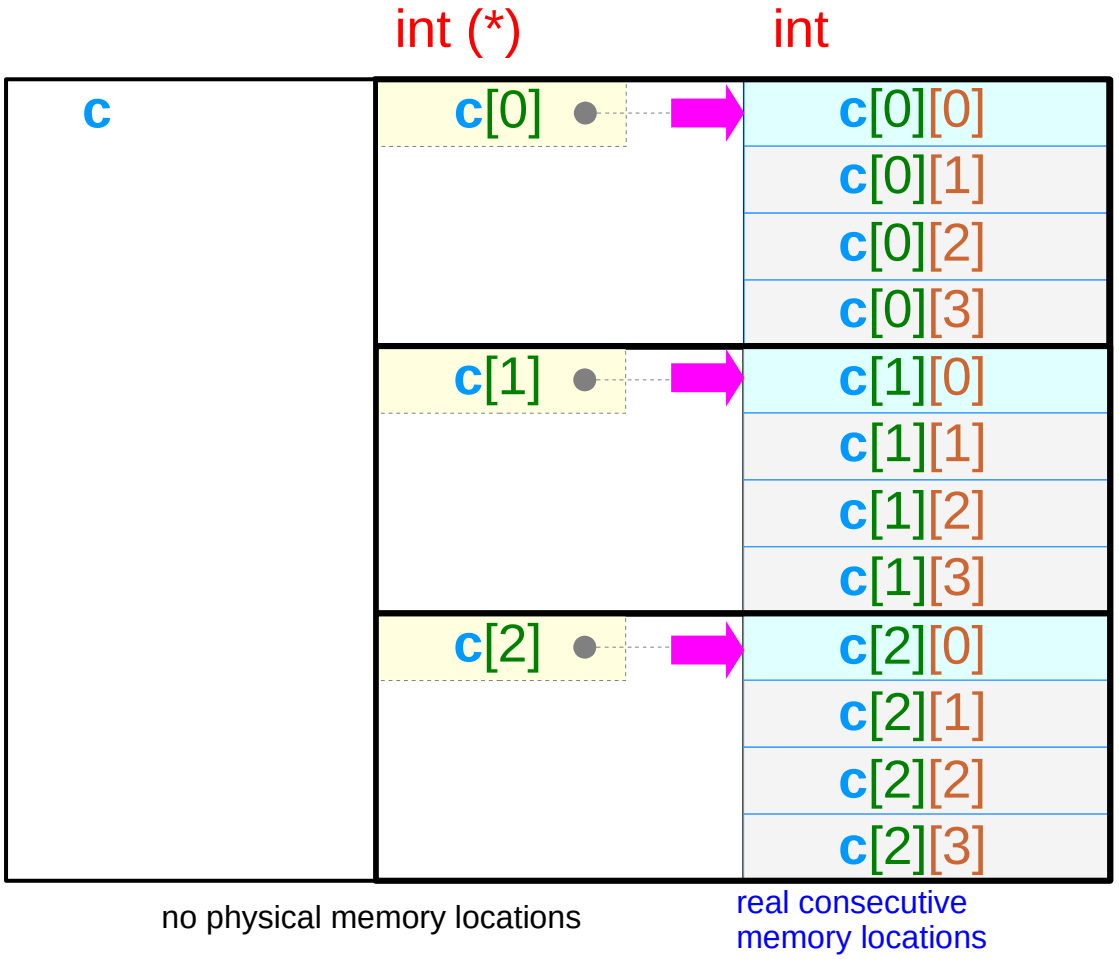
int c [3] [4]

c[i] 0-d array pointer

type : int (*)

value : c[*i*] = &c[*i*][0]

c[i] points to the 1st **int** element
There are 4 **int** elements



Values of virtual array pointer in a **2-d** array

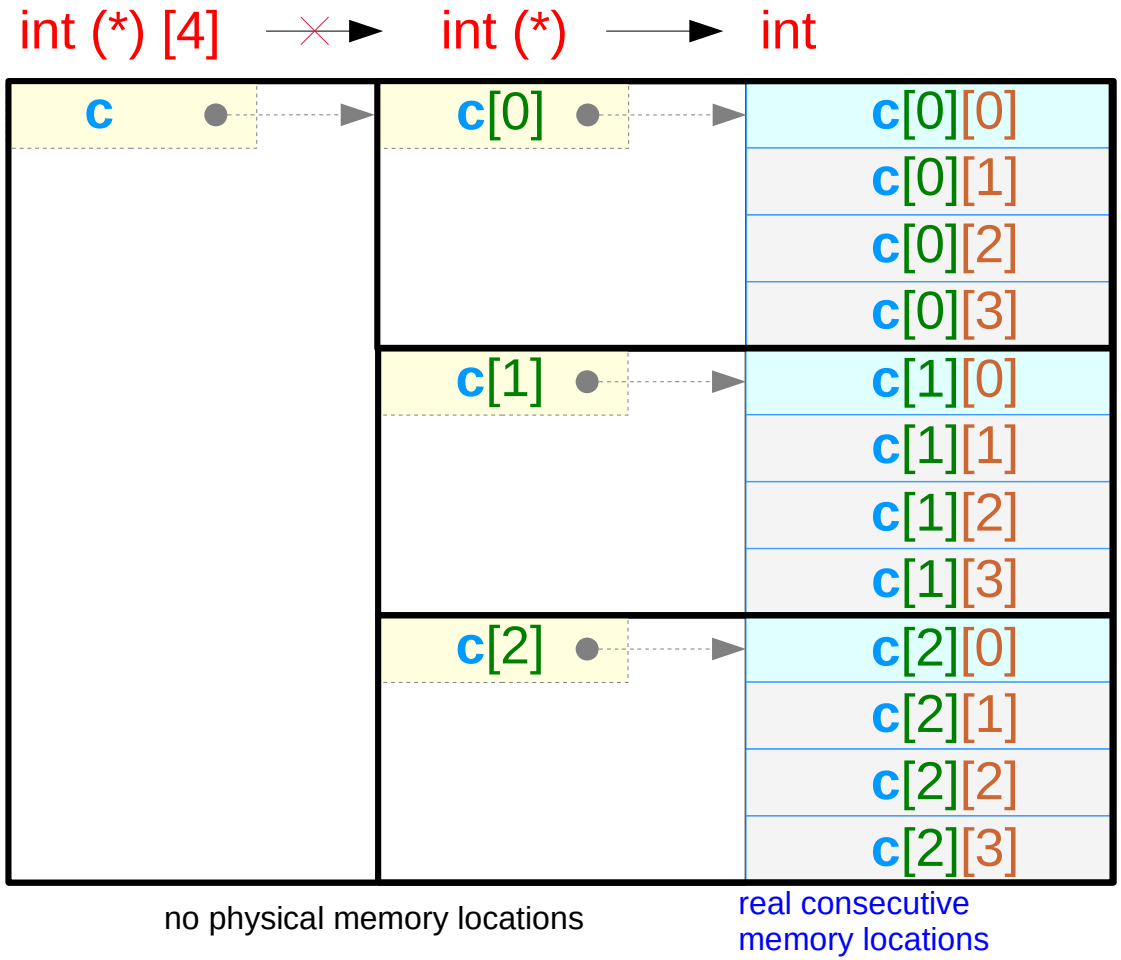
c = c[0] = &c[0][0]

c[1] = &c[1][0]

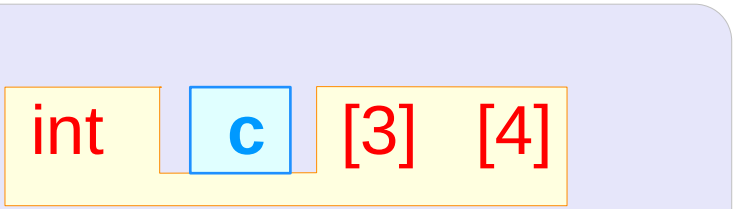
c[2] = &c[2][0]

Only abstract data type can be referenced by a pointer

This figure shows only the same address values of pointers



Types in a **2-d** array

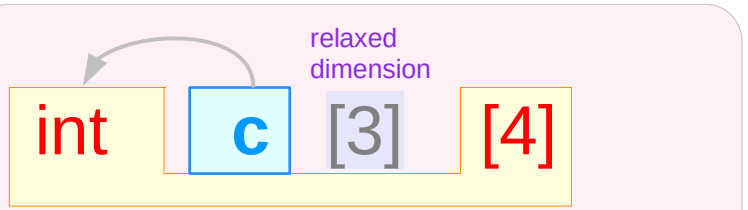


c **2-d array**

type : int [3][4]

size : 3 * 4 * 4

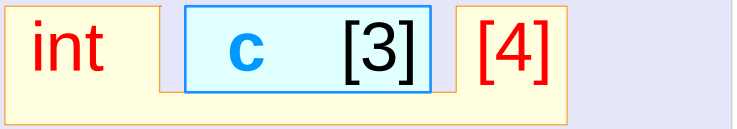
relaxing the 1st dimension



c **1-d array pointer** (virtual)

type : int (*) [4]

value : &c[0][0]

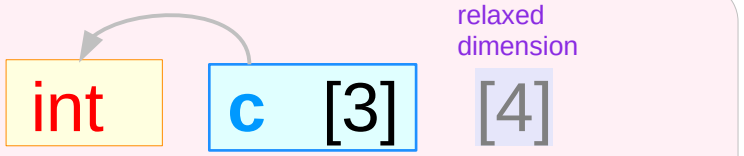


c[i] 1-d array

type : int [4]

size : 4 * 4

relaxing the 1st dimension



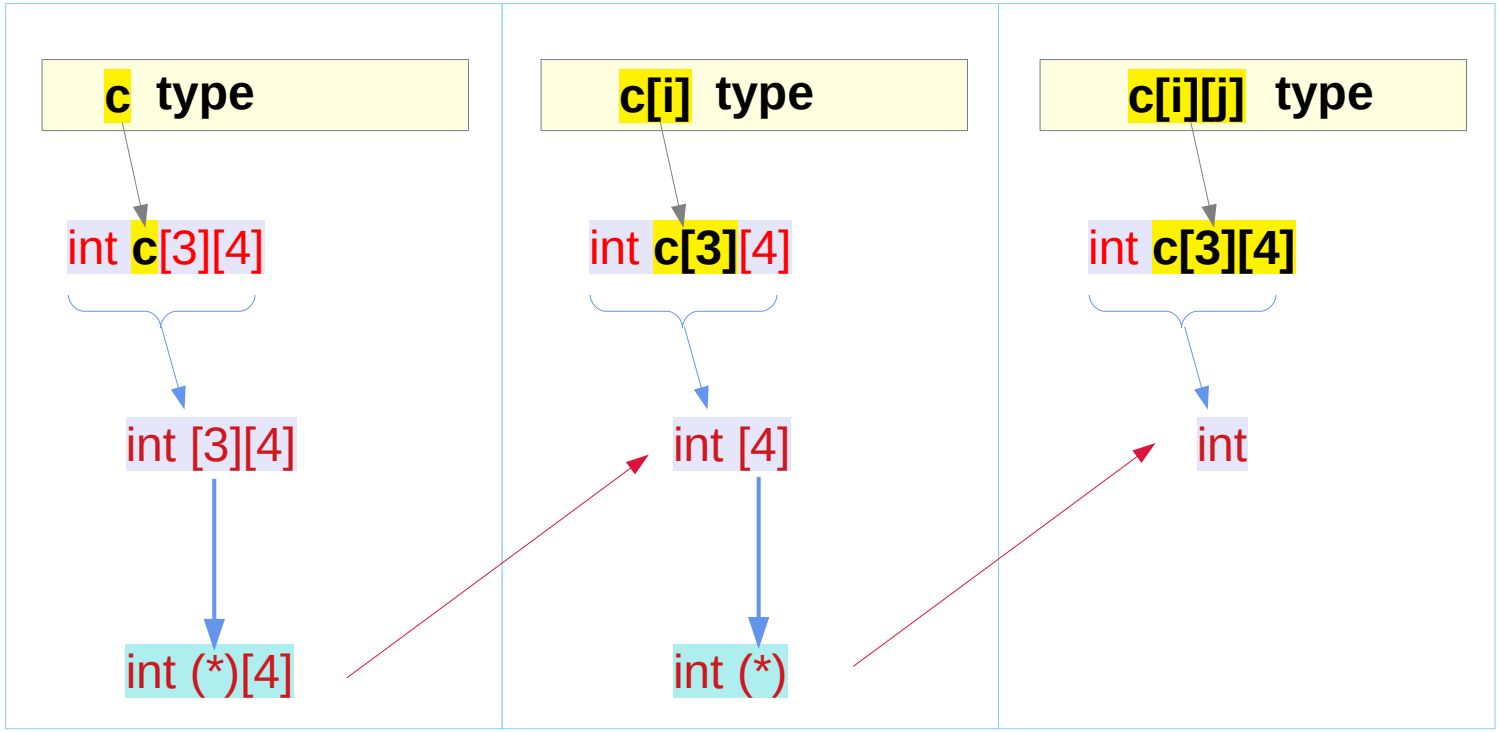
c[i] 0-d array pointer (virtual)

type : int (*)

value : &c[*i*][0]

Sub-array types in a **2-d** array

int c[3][4]; **2-d** array c



Dual Types

Four cases of array pointers

Properties of array pointers

virtual pointer c in an array c — **implicit array pointer**

Abstract data c[N][ ] ... [ ]

sizeof(c) = sizeof(*c) * N

sizeof(*c) = value(c+1) - value(c)

Virtual pointer (*c)[ ] ... [ ]

value(&c) = value(c)

array pointer p — **explicit array pointer**

sizeof(p) = pointer size (4/8 bytes)

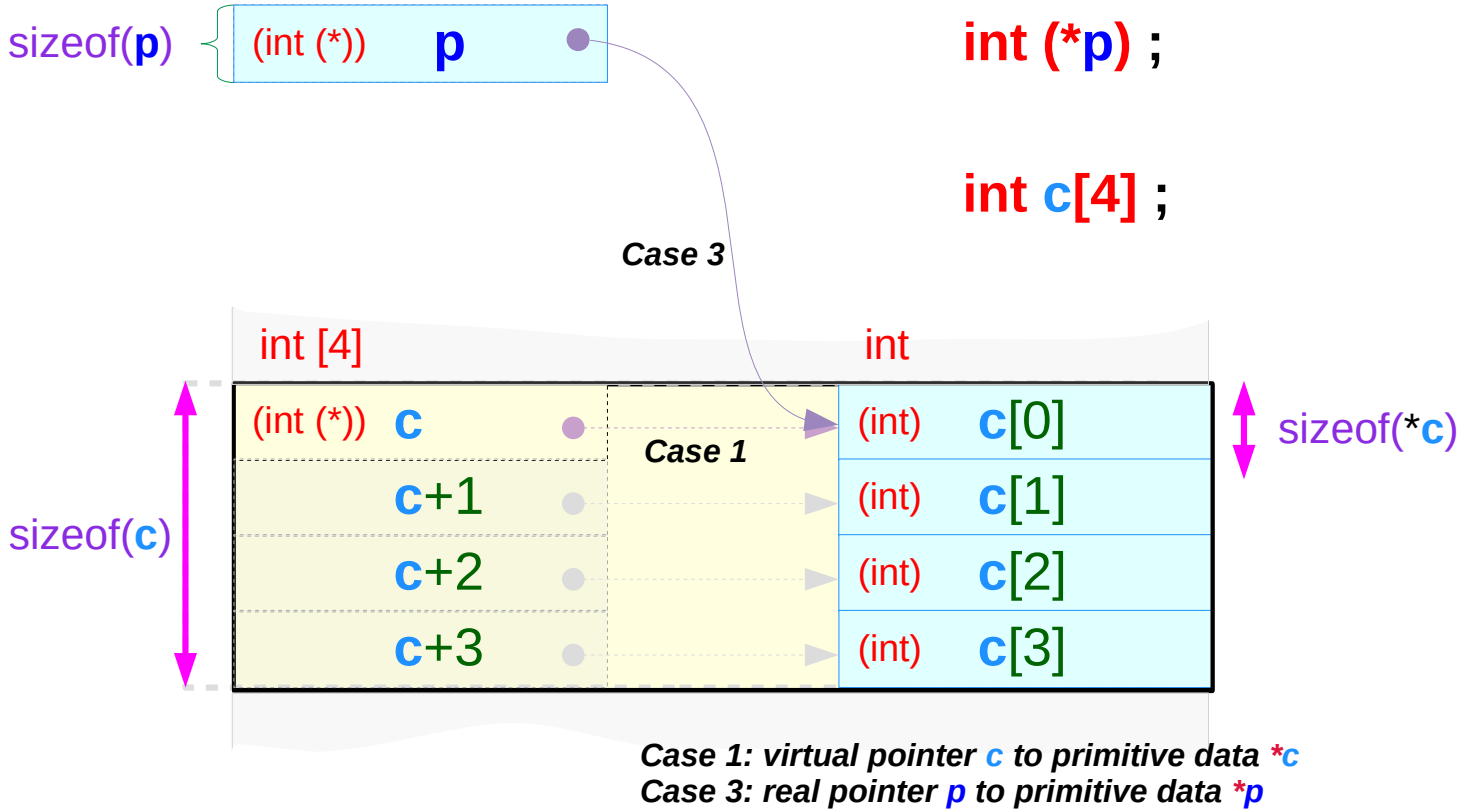
sizeof(*p) = value(p+1) - value(p)

value(&p) ≠ value(p)

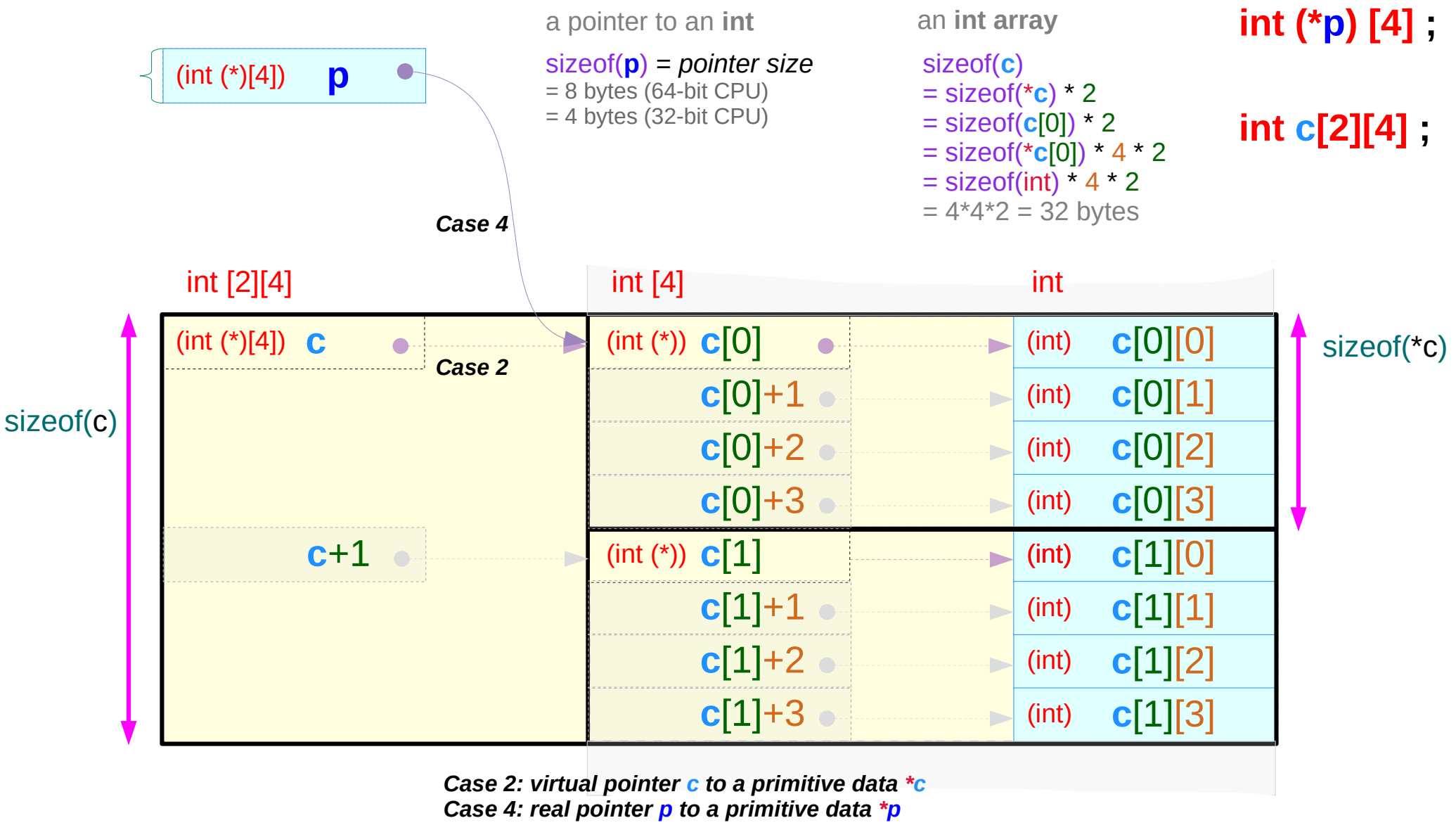
Sizes of integer pointers

a pointer to an **int**

sizeof(**p**) = *pointer size*
= 8 bytes (64-bit CPU)
= 4 bytes (32-bit CPU)

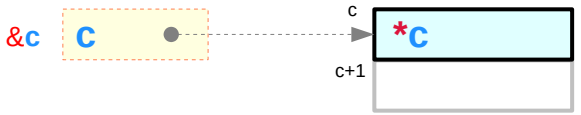


Sizes of integer pointers

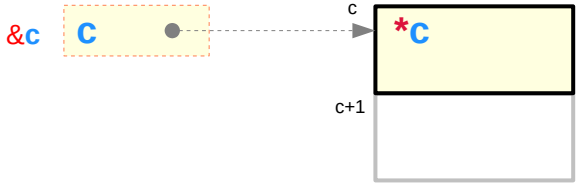


Case 2: virtual pointer c to a primitive data *c
Case 4: real pointer p to a primitive data *p

Case 1 **virtual pointer c** **primitive data *c**



Case 2 **virtual pointer c** **abstract data *c**

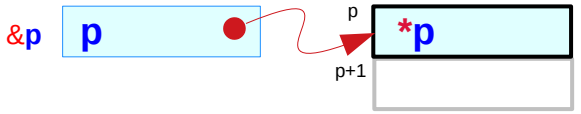


sizeof(**c**) = sizeof(***c**) * N

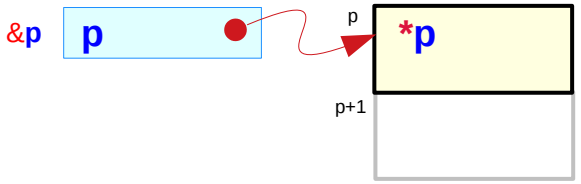
sizeof(***c**) = value(**c**+1) - value(**c**)

value(&**c**) = value(**c**)

Case 3 **real pointer p** **primitive data *p**



Case 4 **real pointer p** **abstract data *p**

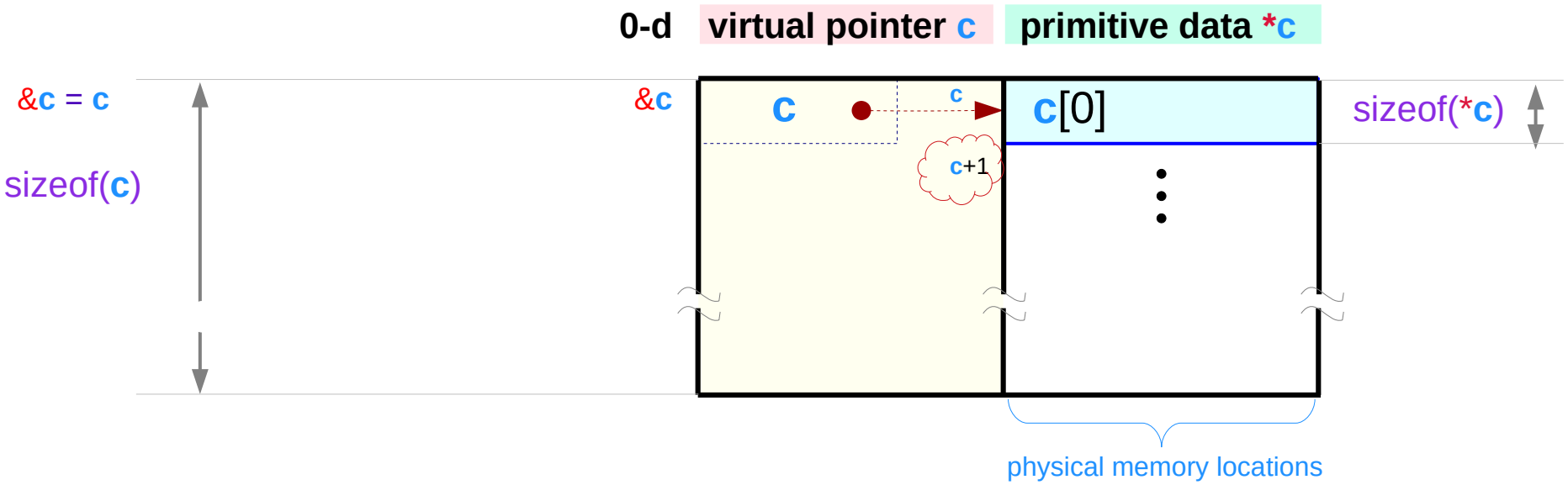


sizeof(**p**) = pointer size (4/8 bytes)

sizeof(***p**) = value(**p**+1) - value(**p**)

value(&**p**) ≠ value(**p**)

Case 1: virtual pointer **c** to a primitive data ***c**



Abstract data c

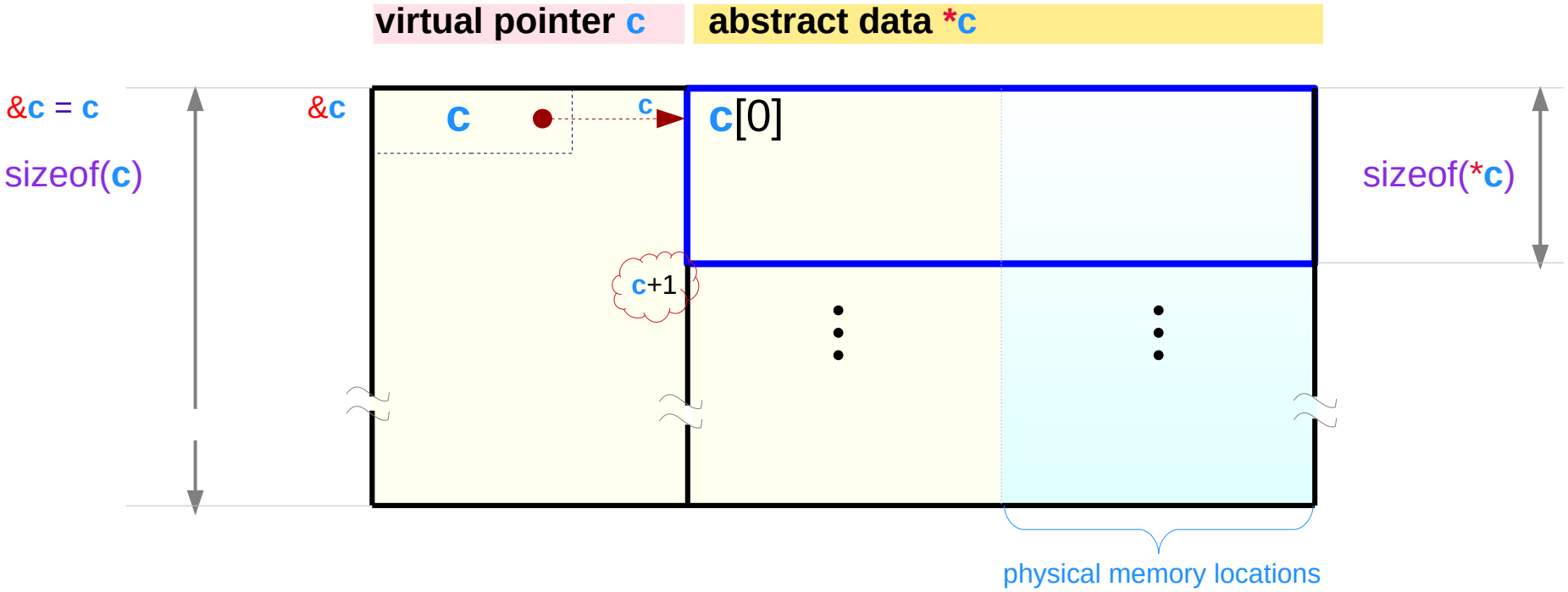
sizeof(**c**) = sizeof(***c**) * N

sizeof(***c**) = value(**c**+1) - value(**c**)

★ **Virtual pointer c**

value(&**c**) = value(**c**)

Case 2: virtual pointer **c** to an abstract data ***c**



Abstract data c, *c

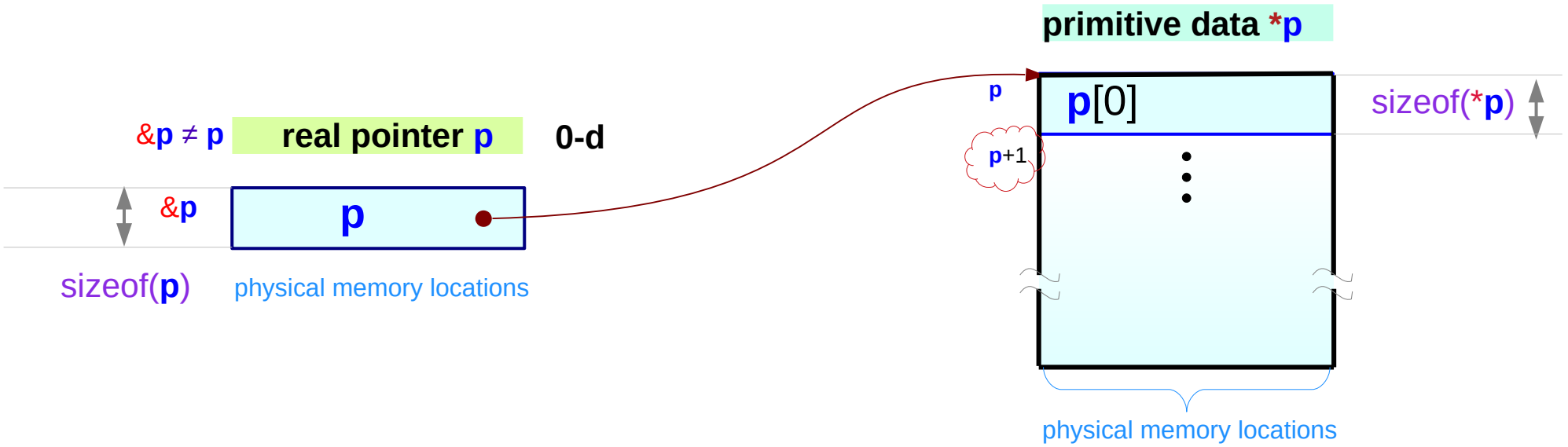
sizeof(**c**) = sizeof(***c**) * N

sizeof(***c**) = value(**c**+1) - value(**c**)

Virtual pointer c

value(&**c**) = value(**c**)

Case 3: real pointer **p** to a primitive data ***p**



real pointer size

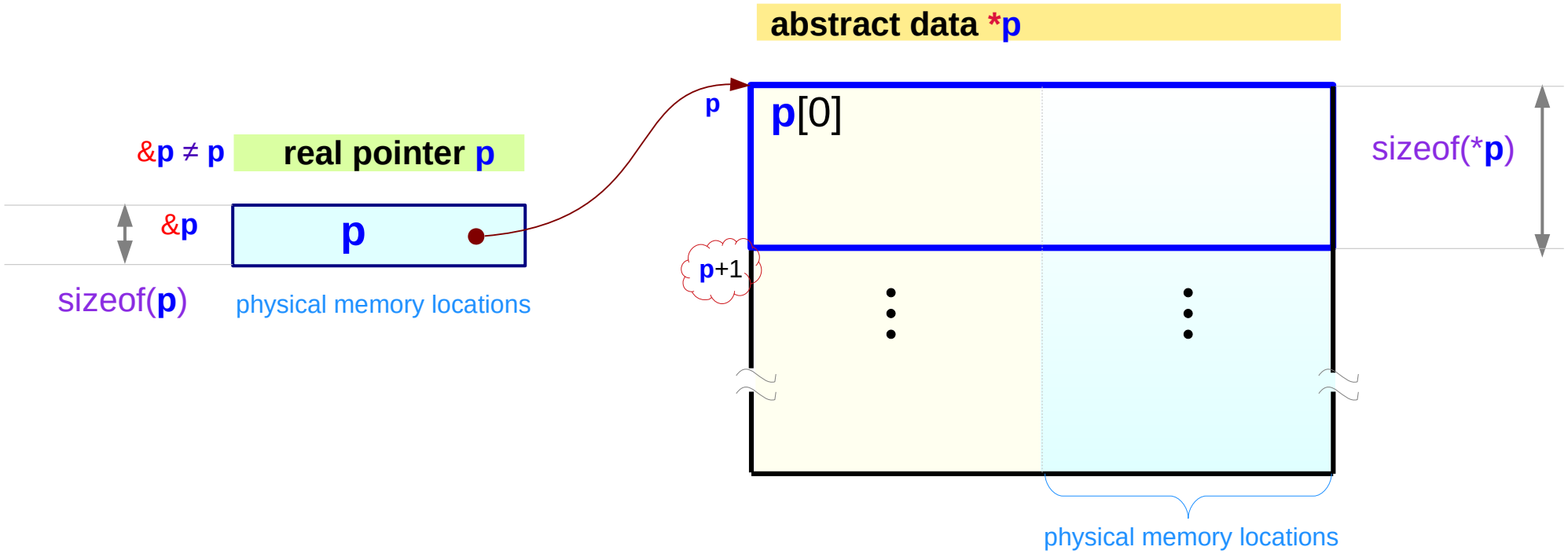
sizeof(**p**) = pointer size (4/8 bytes)

sizeof(***p**) = value(**p**+1) - value(**p**)

unique pointer value and address

value(&**p**) ≠ value(**p**)

Case 4: real pointer **p** to an abstract data ***p**



real pointer size

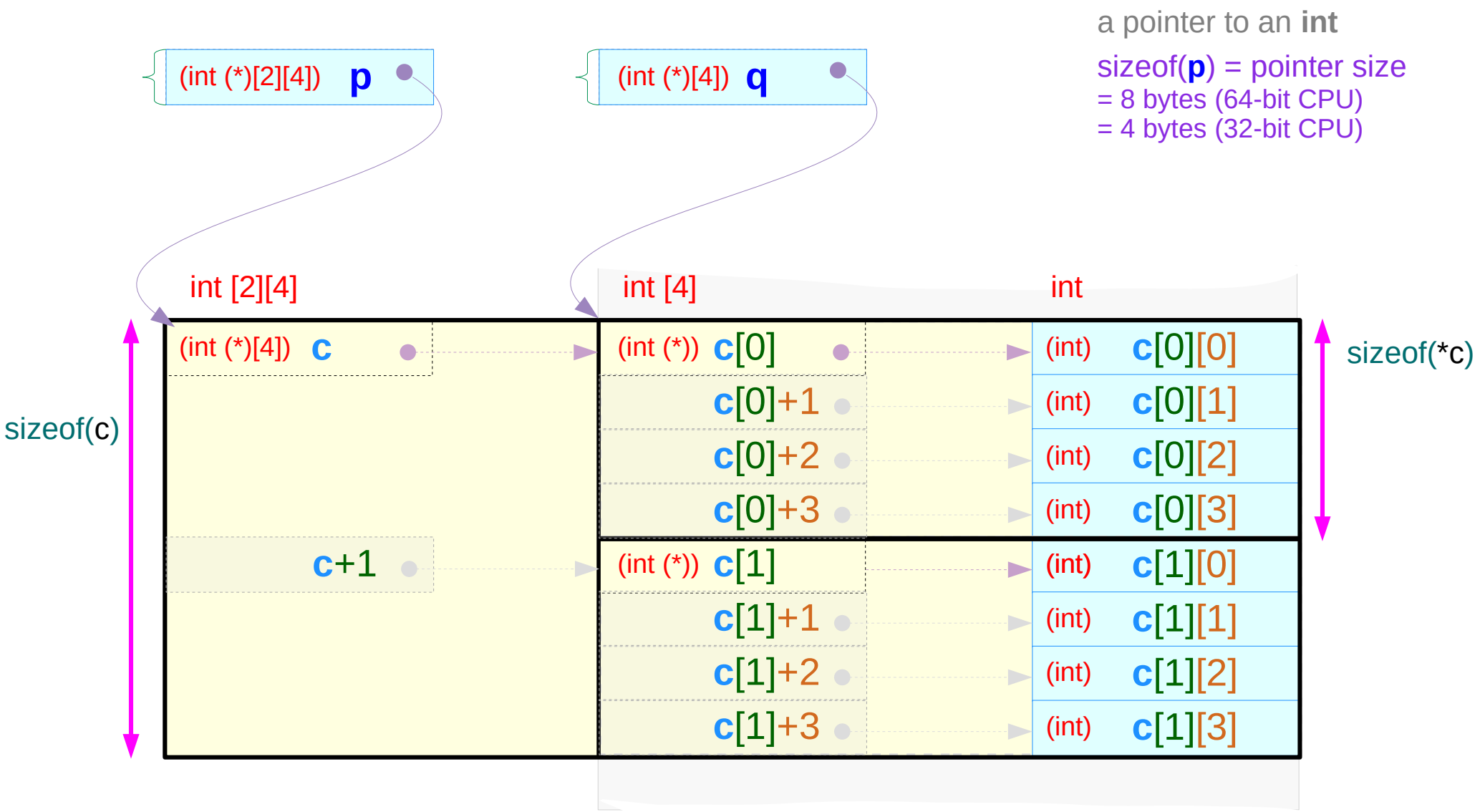
sizeof(**p**) = pointer size (4/8 bytes)

sizeof(***p**) = value(**p**+1) - value(**p**)

unique pointer value and address

value(&**p**) ≠ value(**p**)

Sizes of integer pointers



Incrementing a 2-d array pointer

Incrementing a 1-d array pointer

Subarray sizes referenced by array pointers **p** and **q**

```
int (*p) [3][4] ;
```

2-d array pointer p

sizeof(**p**) = **pointer size**
4 or 8 bytes

int (***p**) [3][4] **p**[0]
sizeof((***p**)) = sizeof(int [3][4])

int (***p**) [3][4] **p**[0][**i**] i = 0,1,2
sizeof((***p**) [**i**]) = sizeof(int [4])

int (***p**) [3][4] **p**[0][**i**][**j**] i = 0,1,2 j = 0,1,2,3
sizeof((***p**) [**i**][**j**]) = sizeof(int)

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

1-d array pointer q

sizeof(**q**) = **pointer size**
4 or 8bytes

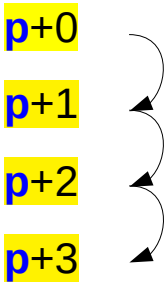
int (***q**) [4] **q**[0]
sizeof((***q**)) = sizeof(int [4])

int (***q**) [**j**] **q**[0][**j**] j = 0,1,2,3
sizeof((***q**) [**j**]) = sizeof(int)

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

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

```
int c [3][4] ;
```



sizeof(*p) sizeof(int [3][4]) sizeof(**2-d array**)

sizeof(*p) sizeof(int [3][4]) sizeof(**2-d array**)

sizeof(*p) sizeof(int [3][4]) sizeof(**2-d array**)

*(p+0) ⟷ p[0]

*(p+1) ⟷ p[1]

*(p+2) ⟷ p[2]

*(p+3) ⟷ p[3]

(*(p+0))[i][j] ⟷ p[0][i][j] used for c[3][4]

(*(p+1))[i][j] ⟷ p[1][i][j]

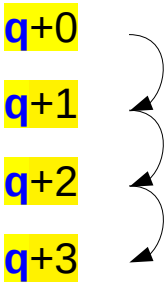
(*(p+2))[i][j] ⟷ p[2][i][j]

(*(p+3))[i][j] ⟷ p[3][i][j]

Incrementing a **1-d** array pointer **q**

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

```
int c [3][4] ;
```



sizeof(*q) sizeof(int [4]) sizeof(**1-d array**)

sizeof(*q) sizeof(int [4]) sizeof(**1-d array**)

sizeof(*q) sizeof(int [4]) sizeof(**1-d array**)

*(q+0) ⟷ q[0]

*(q+1) ⟷ q[1]

*(q+2) ⟷ q[2]

*(q+3) ⟷ q[3]

(*(q+0))[j] ⟷ q[0][j]

(*(q+1))[j] ⟷ q[1][j]

(*(q+2))[j] ⟷ q[2][j]

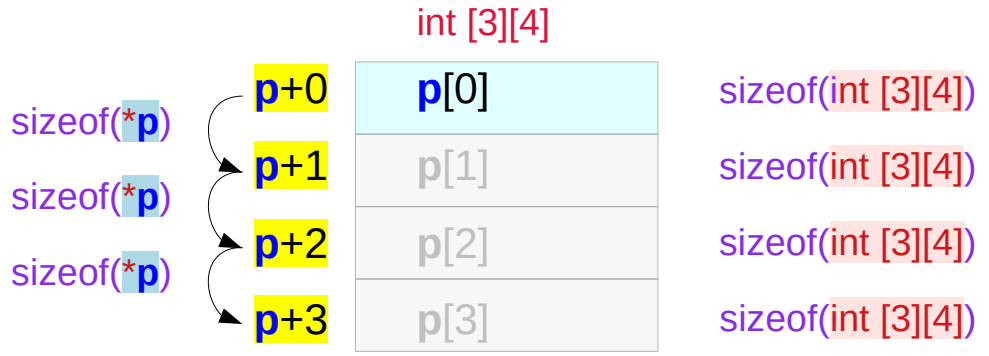
(*(q+3))[j] ⟷ q[3][j]

used for c[3][4]

Subarray sizes referenced by a **2-d** array pointer **p**

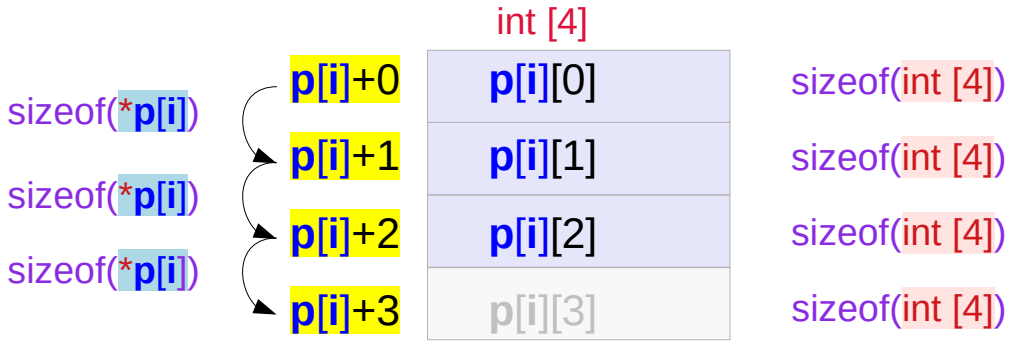
```
int (*p) [3][4] ;
```

(int [3][4]) **p[i]** i = 0



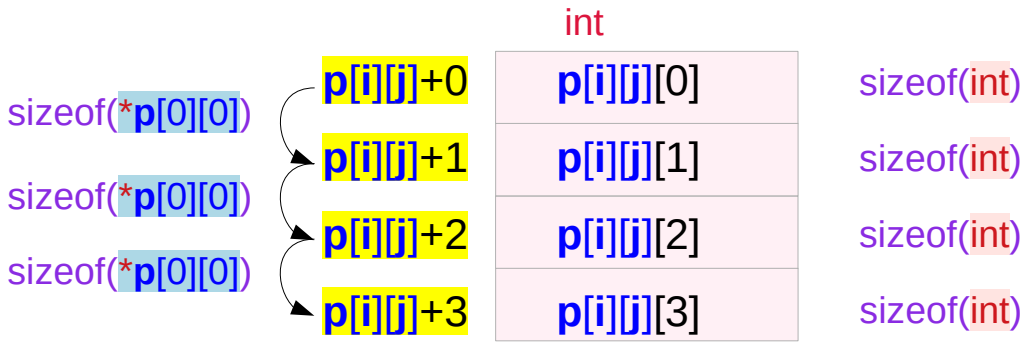

```
int (*p) [3][4] ;
```

(int [4]) **p[i][j]** i = 0 j = 0,1,2




```
int (*p) [3][4] ;
```

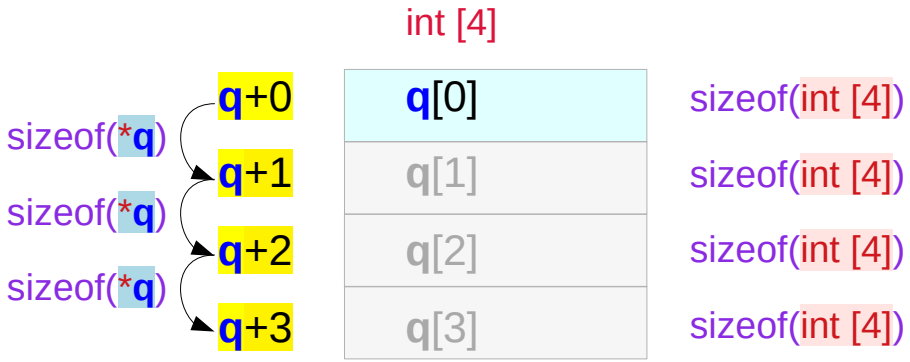
(int) **p[i][j][k]** i = 0 j = 0,1,2 k = 0,1,2,3



Subarray sizes referenced by a **1-d** array pointer **q**

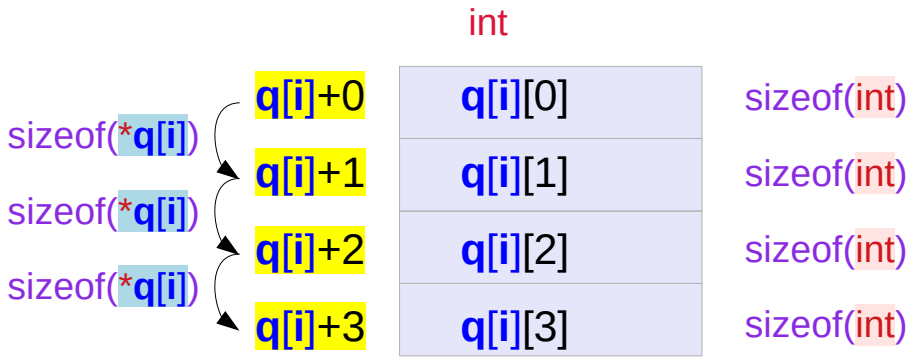
```
int (*q) [4] ;
```

(int [4]) **q[i]** i = 0,1,2




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

(int) **q[i][j]** i = 0,1,2 j = 0,1,2,3



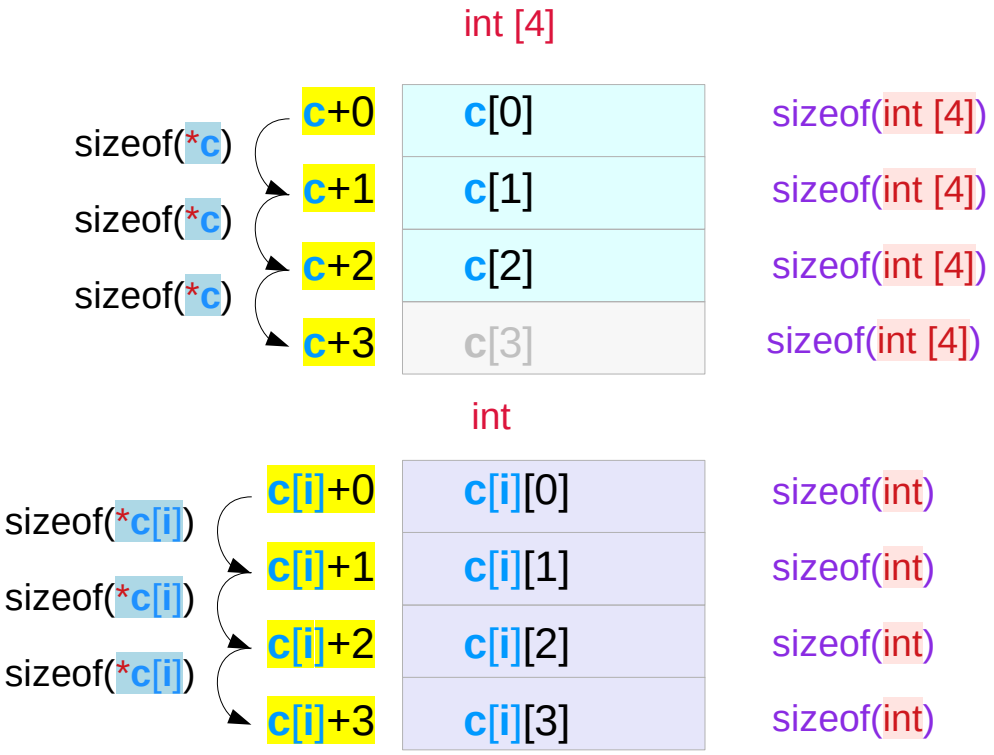
Subarray sizes in a **1-d** array **c**

```
int c [3][4] ;
```

(int [4]) **c[i]** i = 0,1,2

```
int c [3][4] ;
```

(int) **c[i][j]** i = 0,1,2
j = 0,1,2,3



Array element notation p[i]

Dereference notation *(p+i)

```
int a[4];          int (*p) [4]     = &a;
int c[3][4];       int (*p) [3][4] = &c;
int c[3][4];       int ( *   q )[4]  =   c;
```

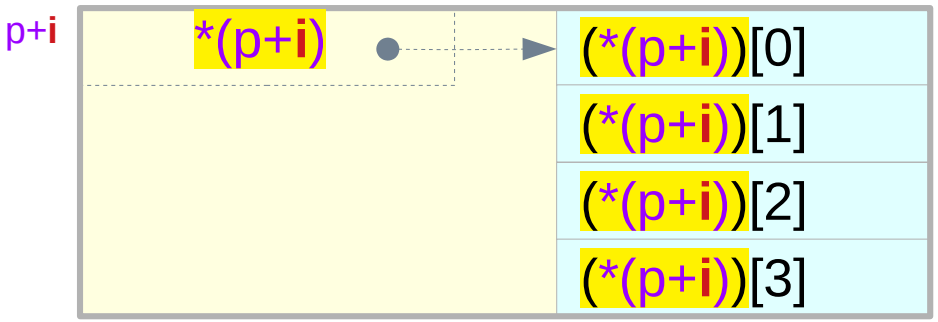
1-d array, 1-d array pointer

2-d array, 2-d array pointer

2-d array, 1-d array pointer

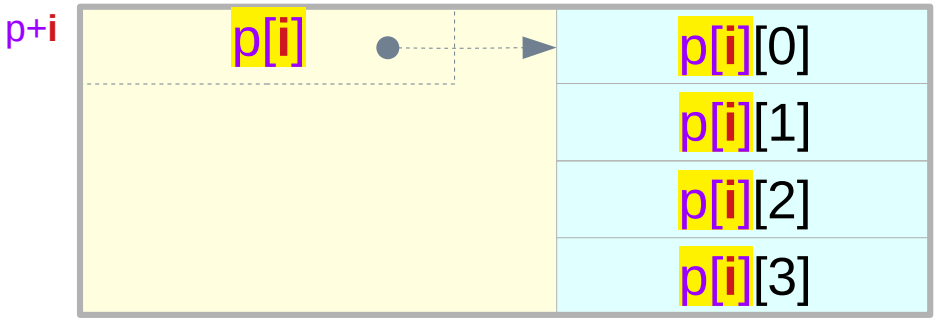
Equivalence : *(p+i) = p[i]

(*(p+**i**)) : **1-d** array name



*(p+i) ≡ p[i]

equivalent



p[**i**] : **1-d** array name

malloc(3*4*4) ;

*(p+i)

***(p+i)**

for i = 0,1,2

p[i]

p[i]

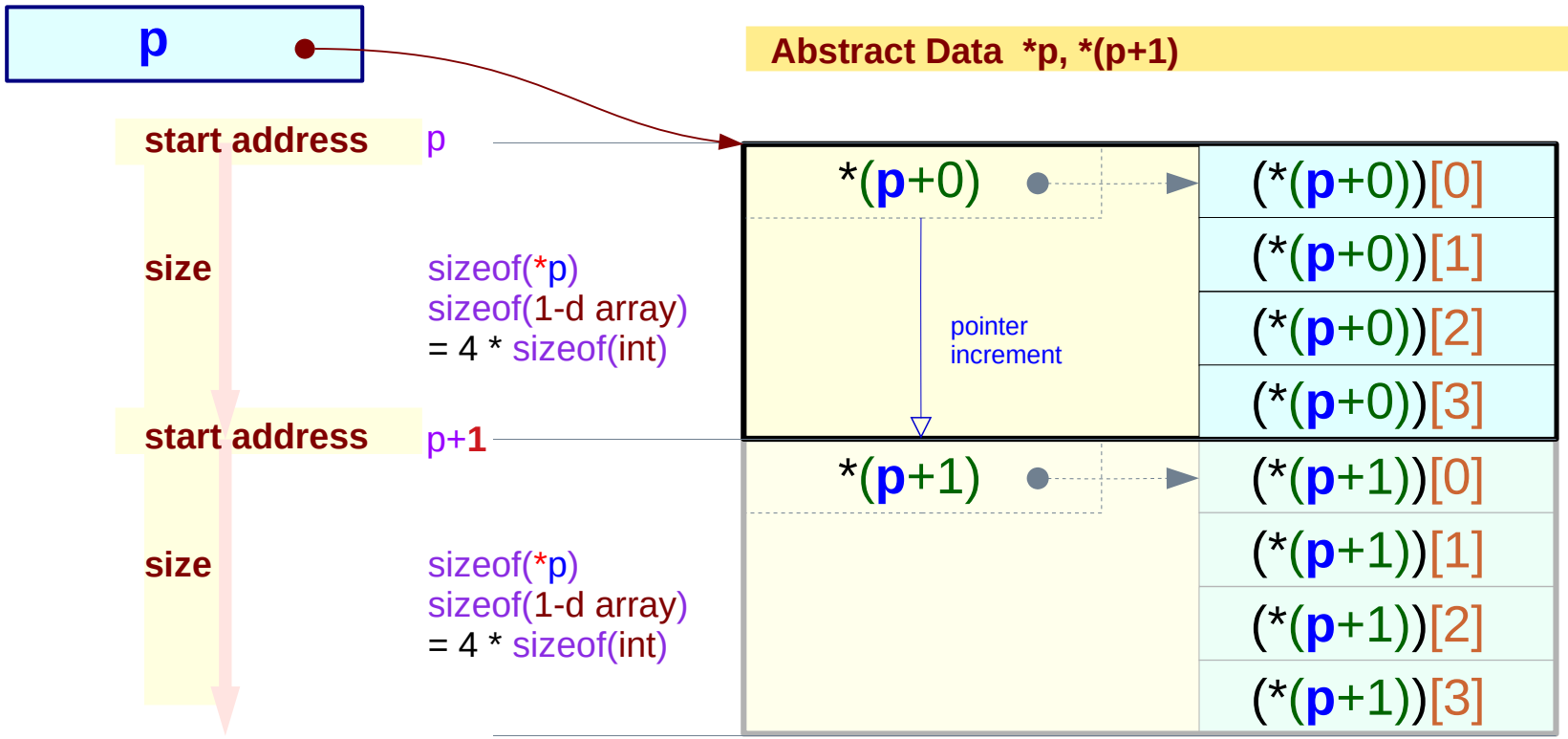
int p [3][4];

1-d array, 1-d array pointer p – using *

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

value(p+1) – value(p) = sizeof(*p)

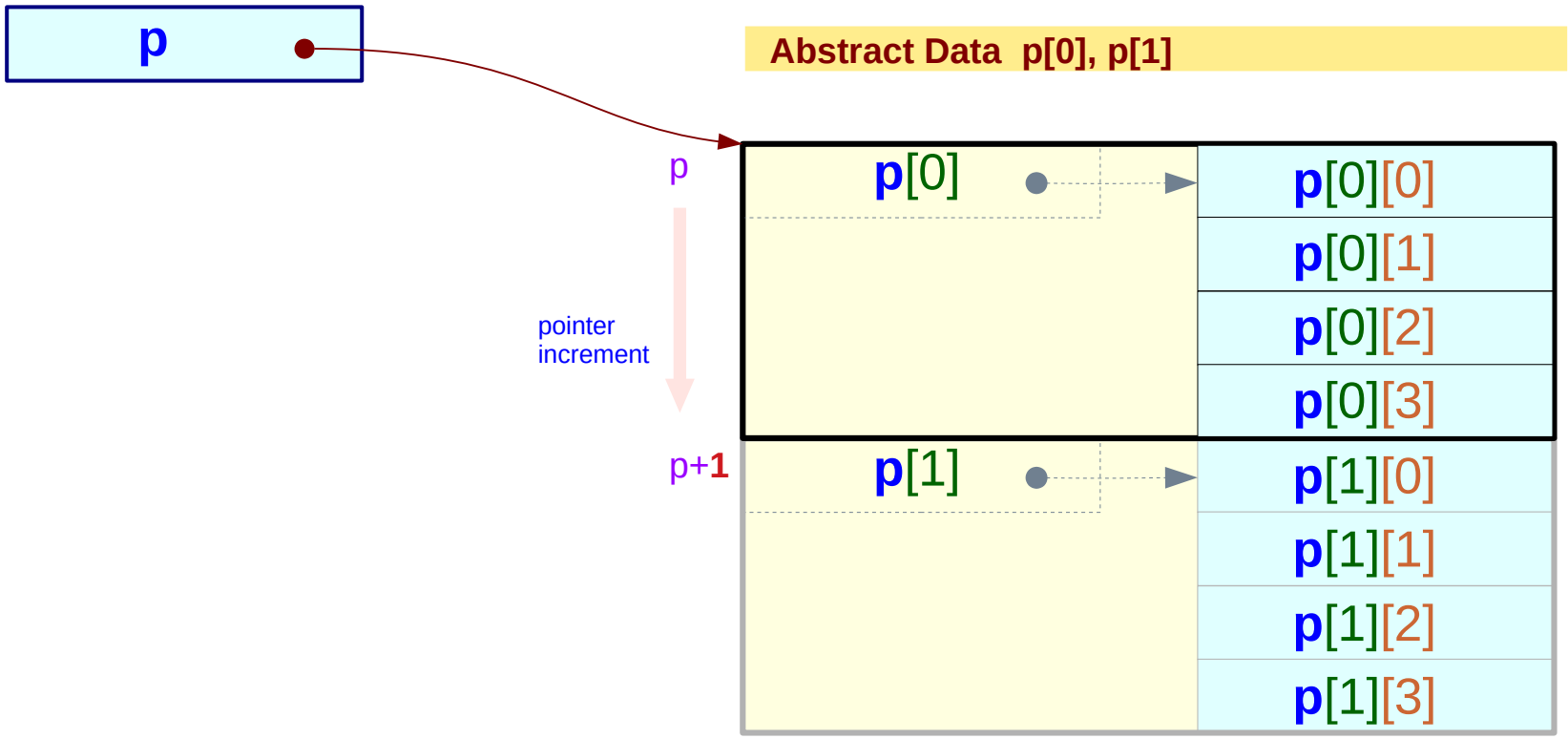
= (long) (p+1) – (long) (p) = 4 * sizeof(int)



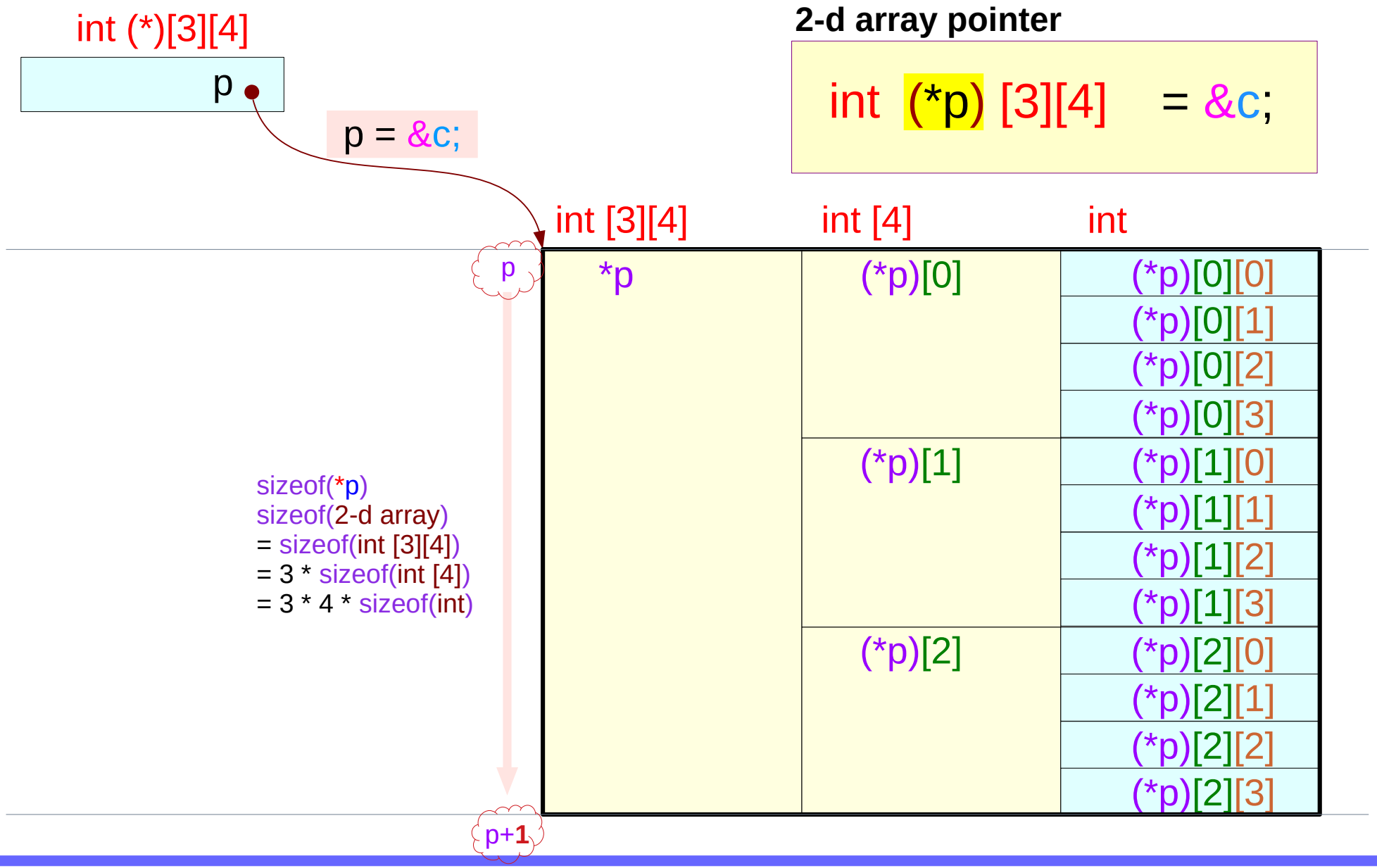
1-d array, 1-d array pointer p – using [ ]

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

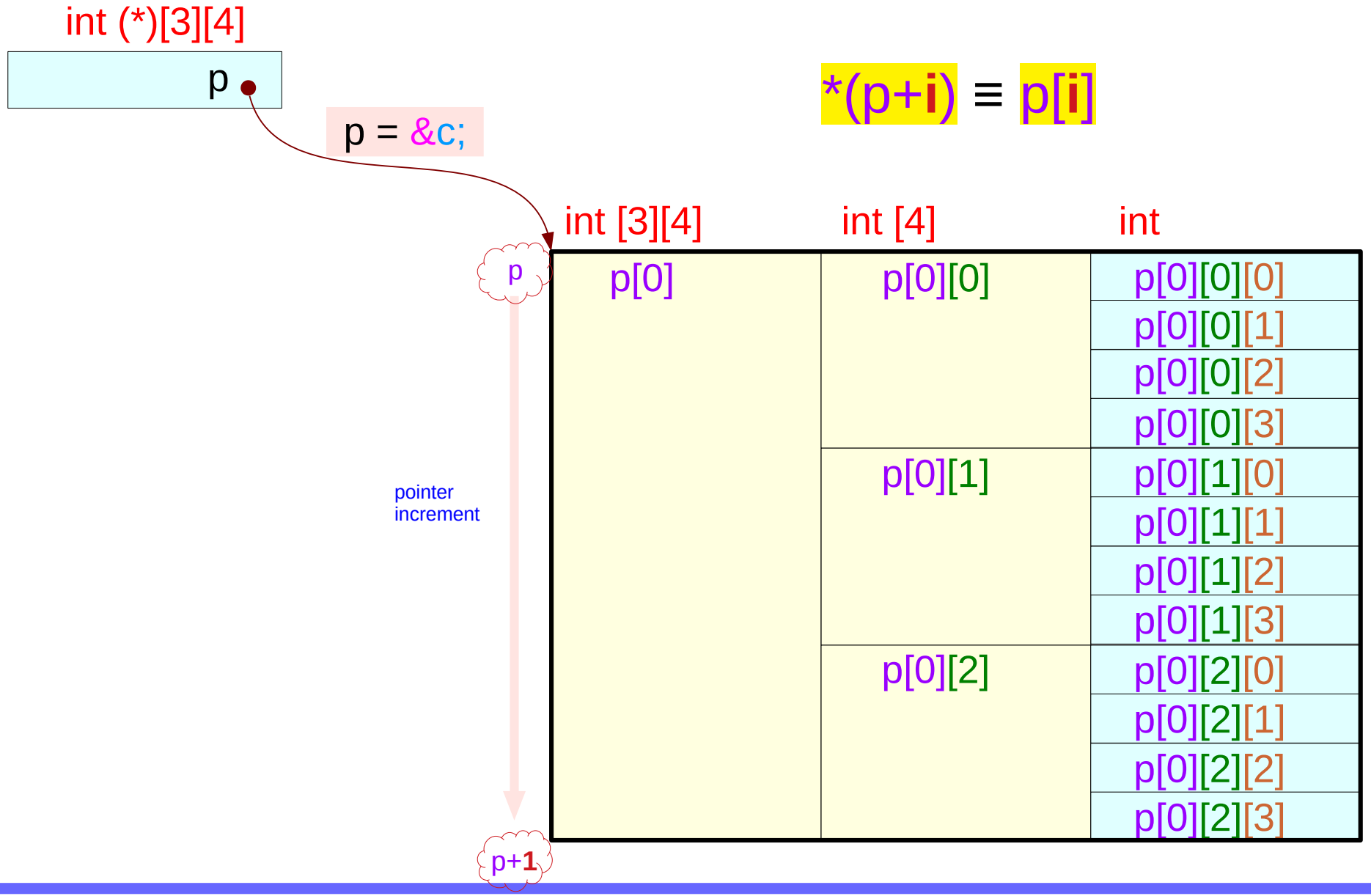
*(p+i) ≡ p[i]



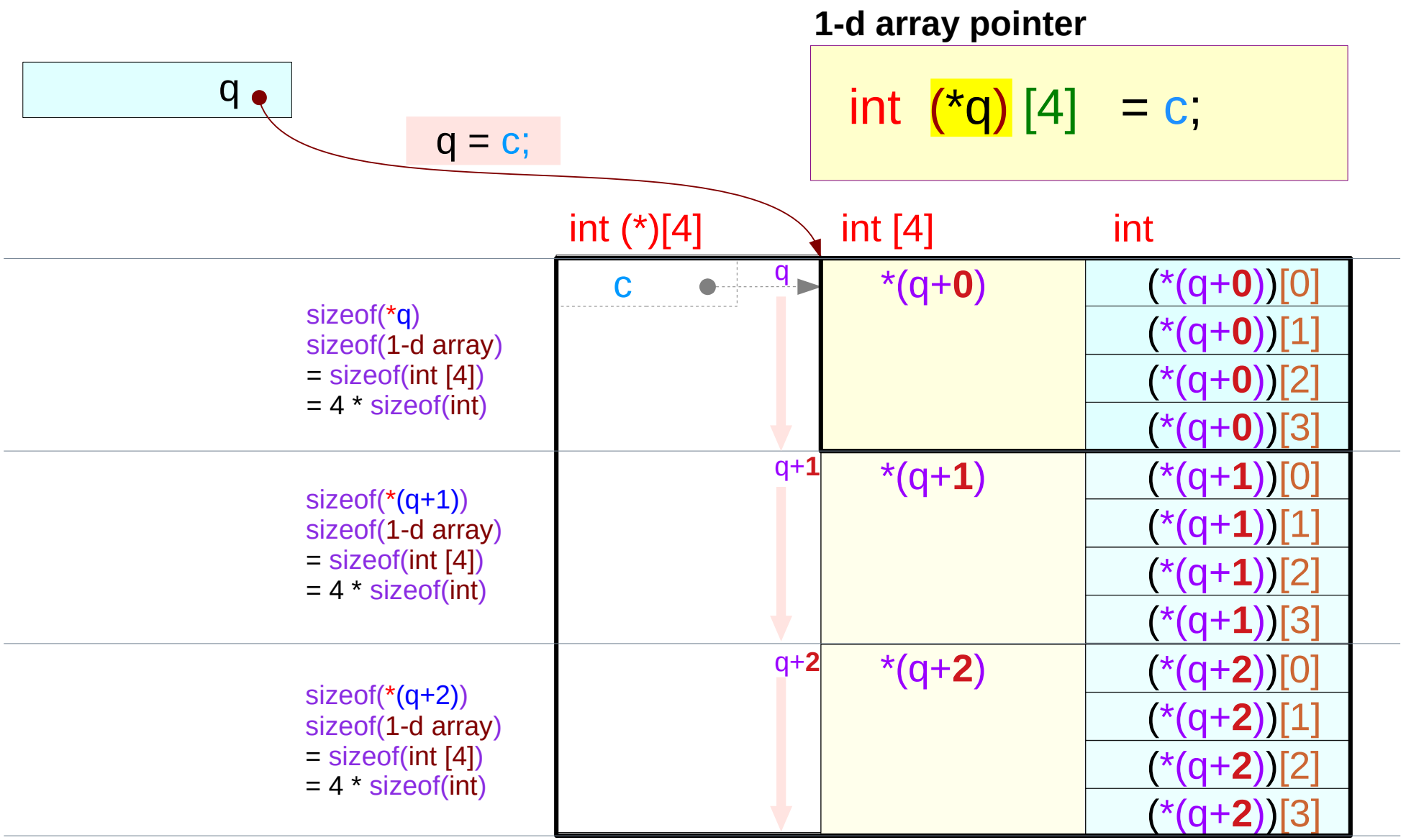
2-d array, 2-d array pointer p – using *



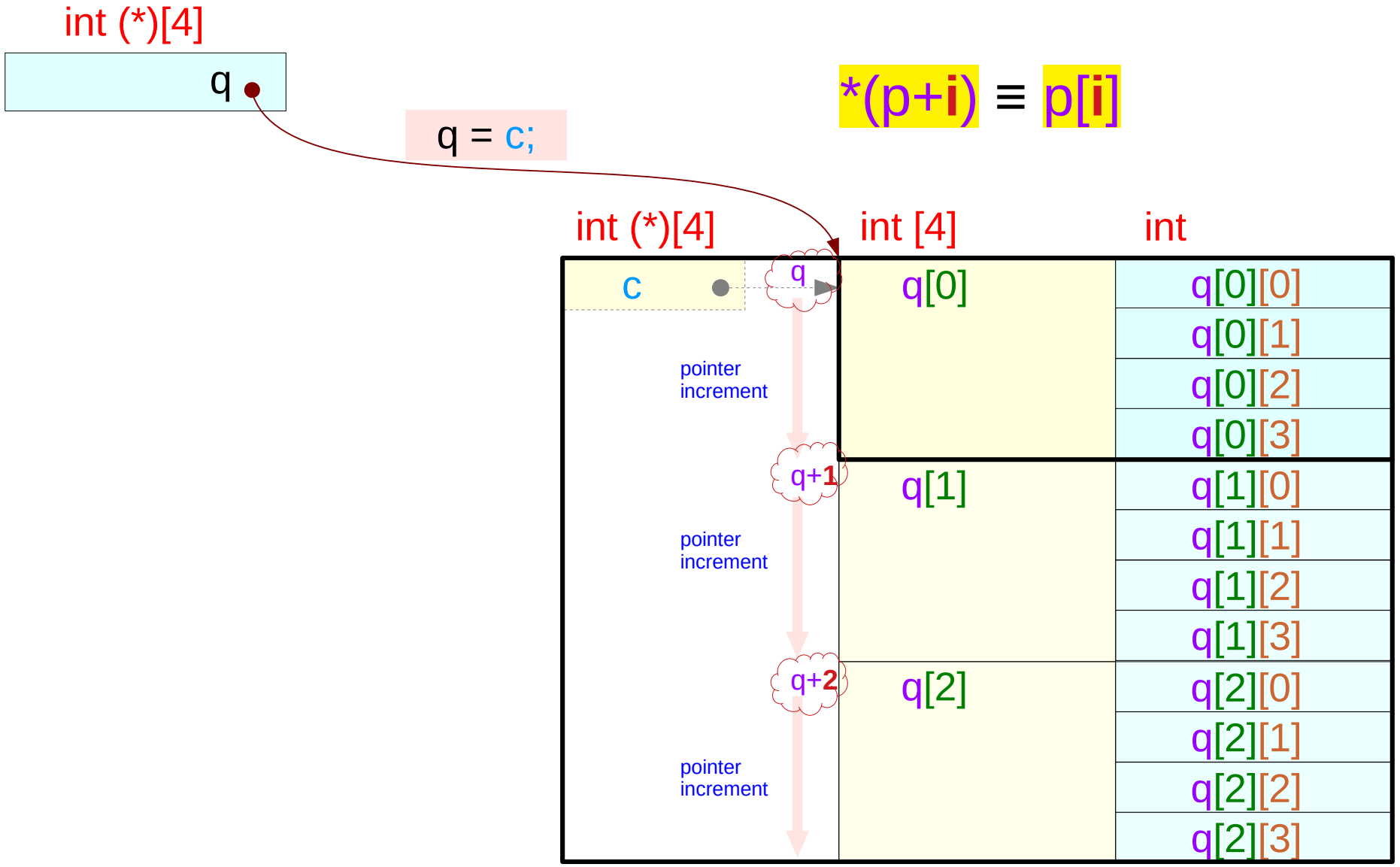
2-d array, 2-d array pointer p – using [ ]



2-d array, 1-d array pointer q – using *



2-d array, 1-d array pointer q – using [ ]



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