Array Pointers (1A)

Young Won Lim
7/16/21

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

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);
```

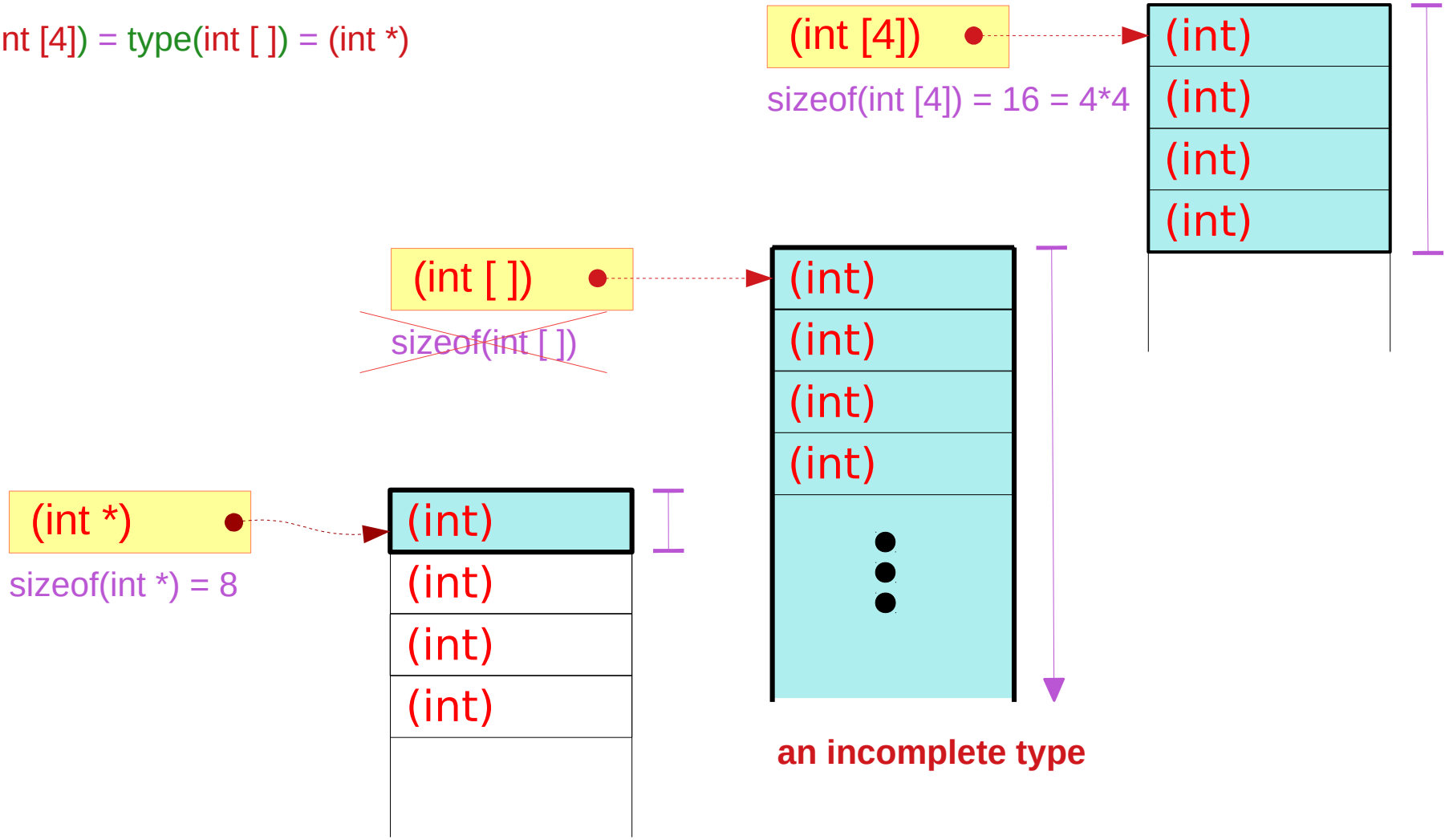
int *

int [N]

int [ ]

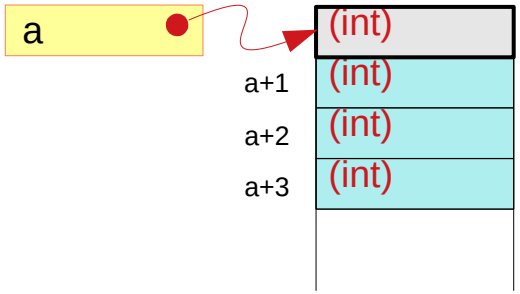
Differences in pointer types – **int [4], int [ ], int ***

type(int [4]) = type(int [ ]) = (int *)



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

int *a;

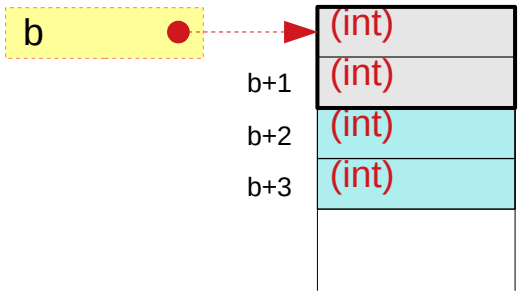


a[0] = *a
a[1] = *(a+1)
a[2] = *(a+2)
a[3] = *(a+3)

syntactically legitimate

programmers must ensure their validity

int b[2]

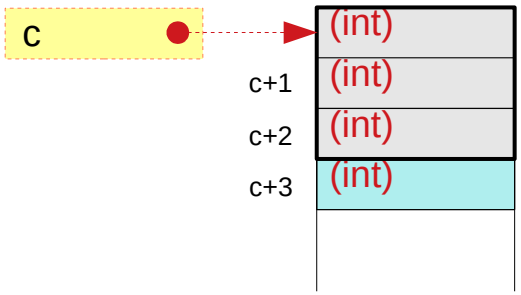


b[0] = *b
b[1] = *(b+1)
b[2] = *(b+2)
b[3] = *(b+3)

syntactically legitimate

programmers must ensure their validity

int c[3];



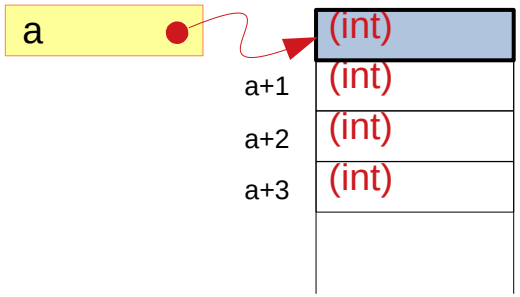
c[0] = *c
c[1] = *(c+1)
c[2] = *(c+2)
c[3] = *(c+3)

syntactically legitimate

programmers must ensure their validity

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

int *a;



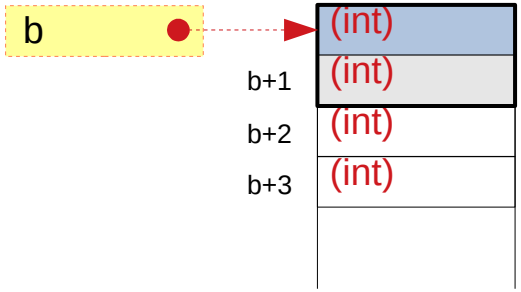
a[0] = *a

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

value(&a) ≠ value(a)

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

int b[2]



b[0] = *b

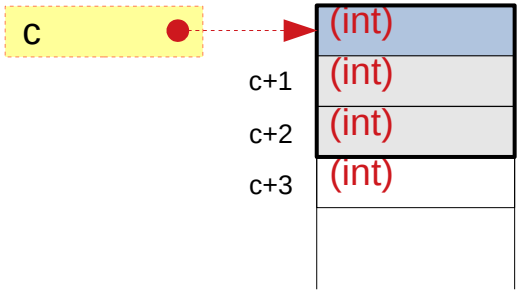
type(b) = int *
type(&b) = int (*) [2]

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];



c[0] = *c

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

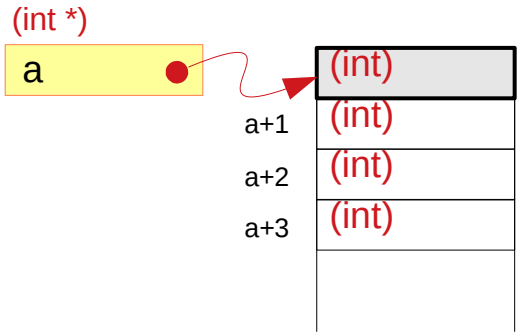
value(&c) = value(c)

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

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

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

int *a;



sizeof (a) = pointer size

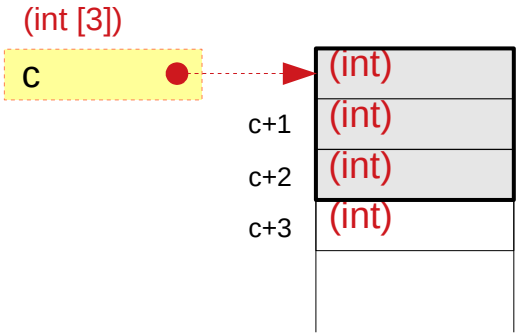
value(&a) ≠ value(a)

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

real memory location for a

a :: int *
&a :: int **

int c[3];



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

value(&c) = value(c)

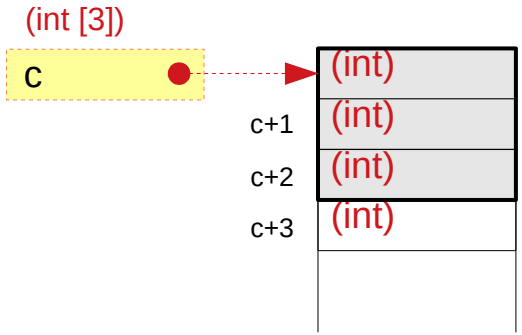
the starting address of array variable c
is equal to the address of the 1st element

no actual memory location for c

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

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

int c[3];



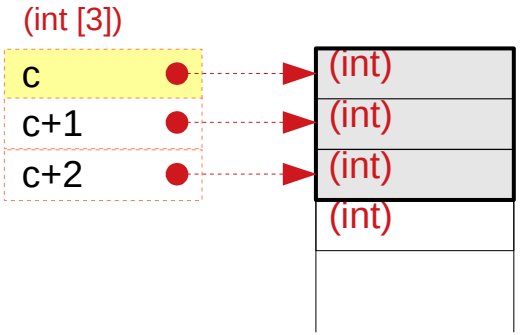
sizeof (c) = sizeof(int) * 3

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]

Types of multi-dimension array names

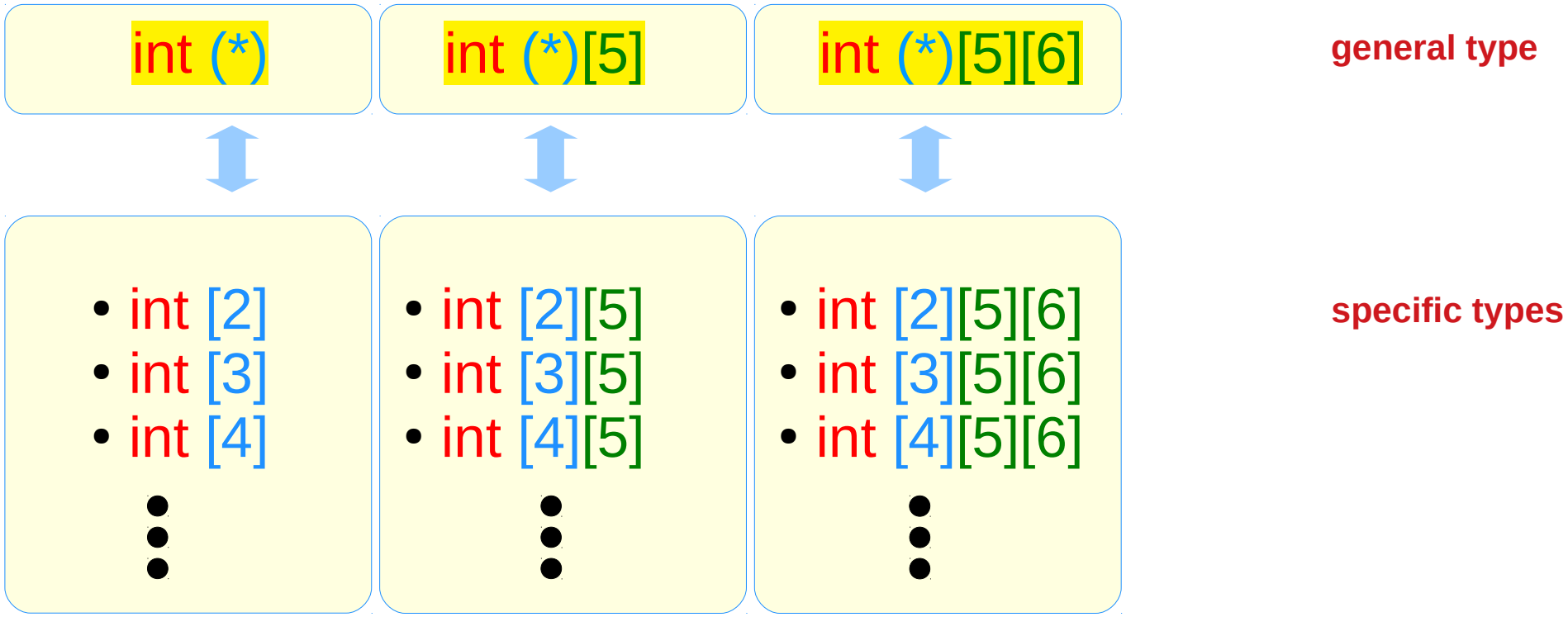
```
int     a ;
int     b [4];
int     c [4][5];
int     d [4][5][6];
```

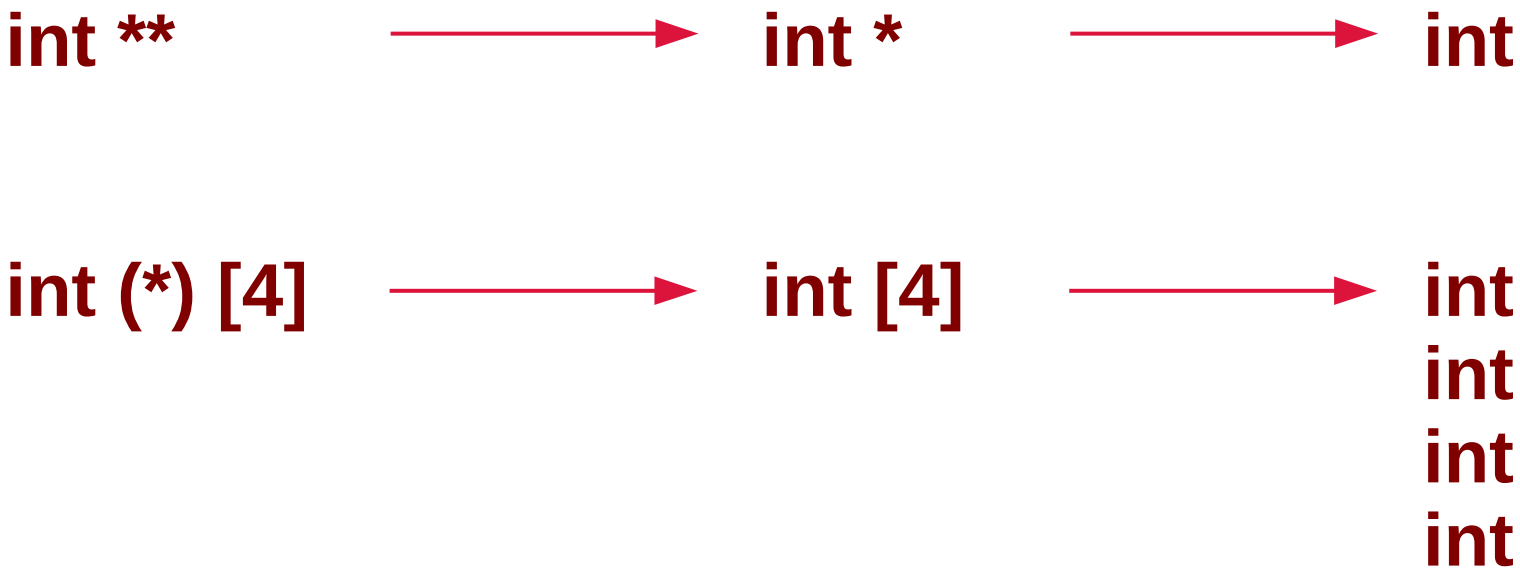
```
a  :: int                 ➡  int
b  :: int [4]             ➡  int (*)          int  *
c  :: int [4][5]          ➡  int (*)[5]
d  :: int [4][5][6]       ➡  int (*)[5][6]
```

array types — *specific types*

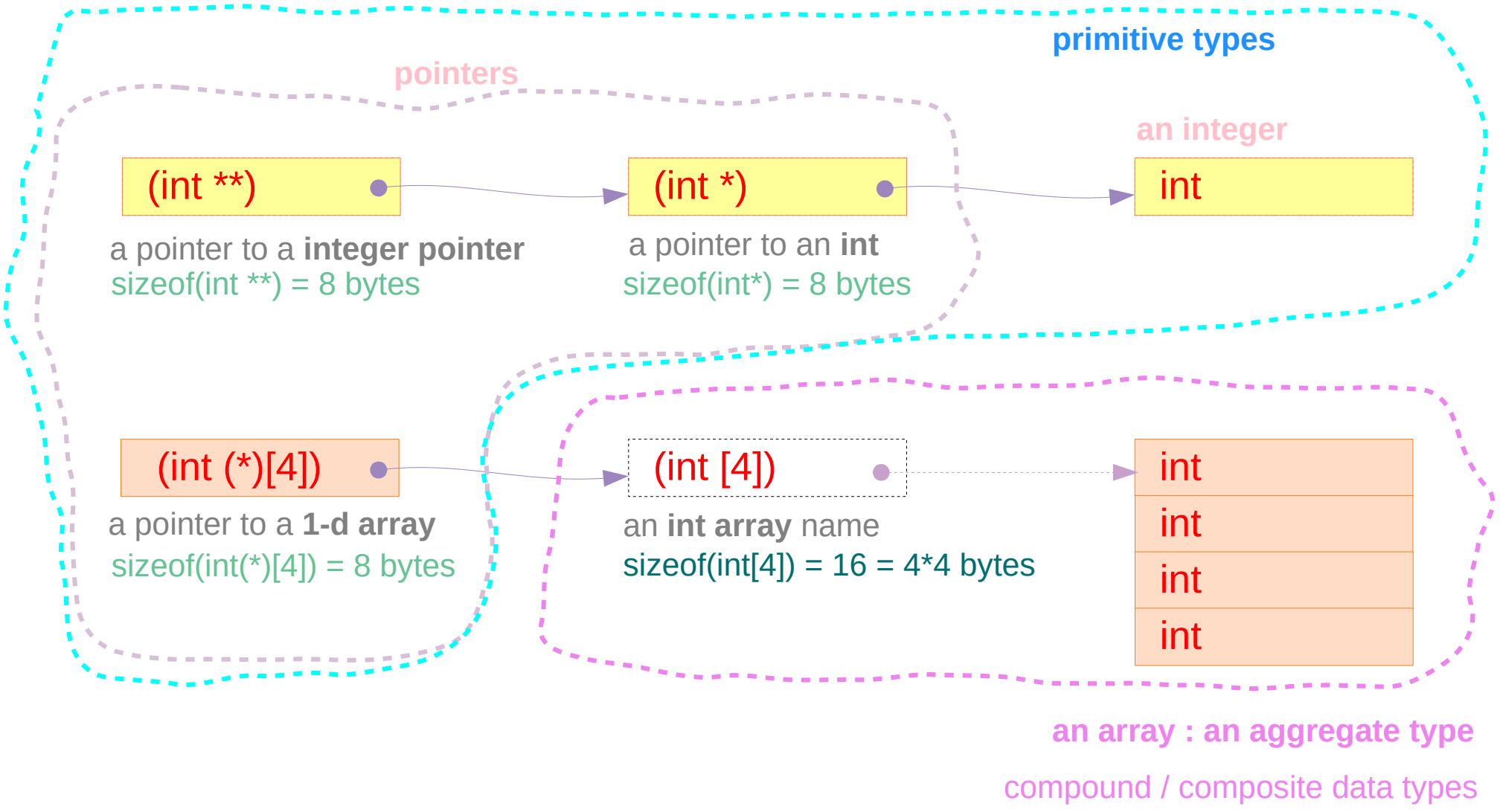
array pointer types — *general type*

Array pointers v.s. Array

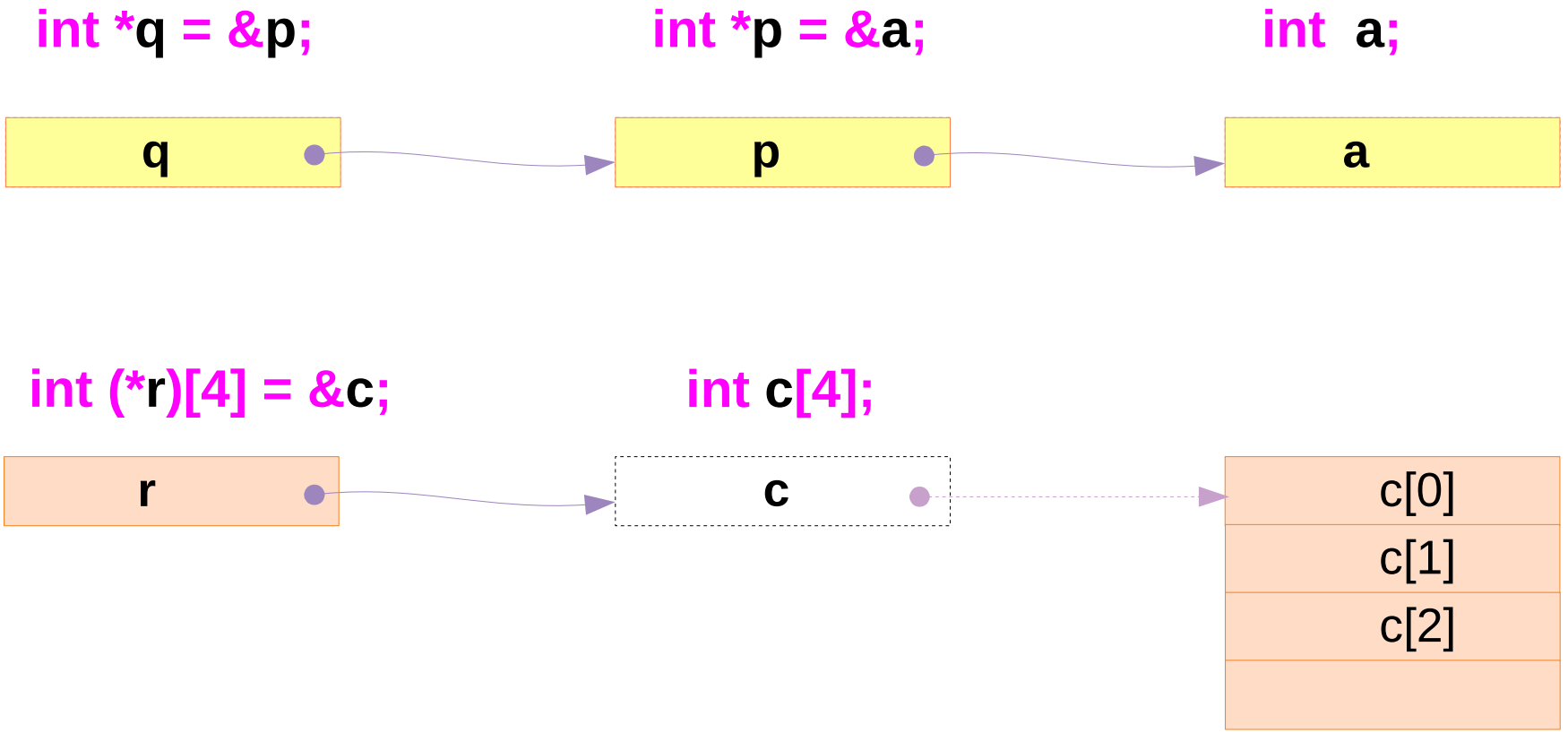



int **
int *
int
int (*) [4]
int [4]
int
int
int
int

Types of integer pointers



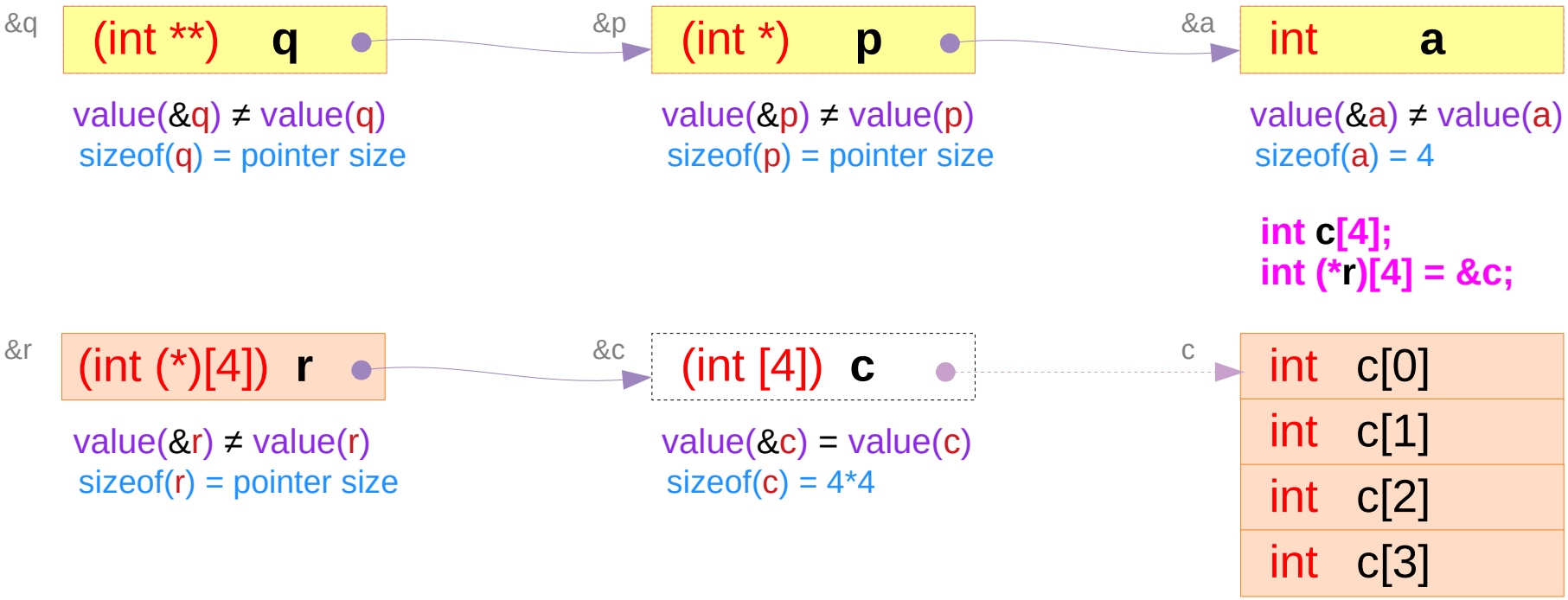
Variable declaration of integer pointers



Types and sizes of integer pointers

type(int [4]) = type(int [ ]) = (int *)

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



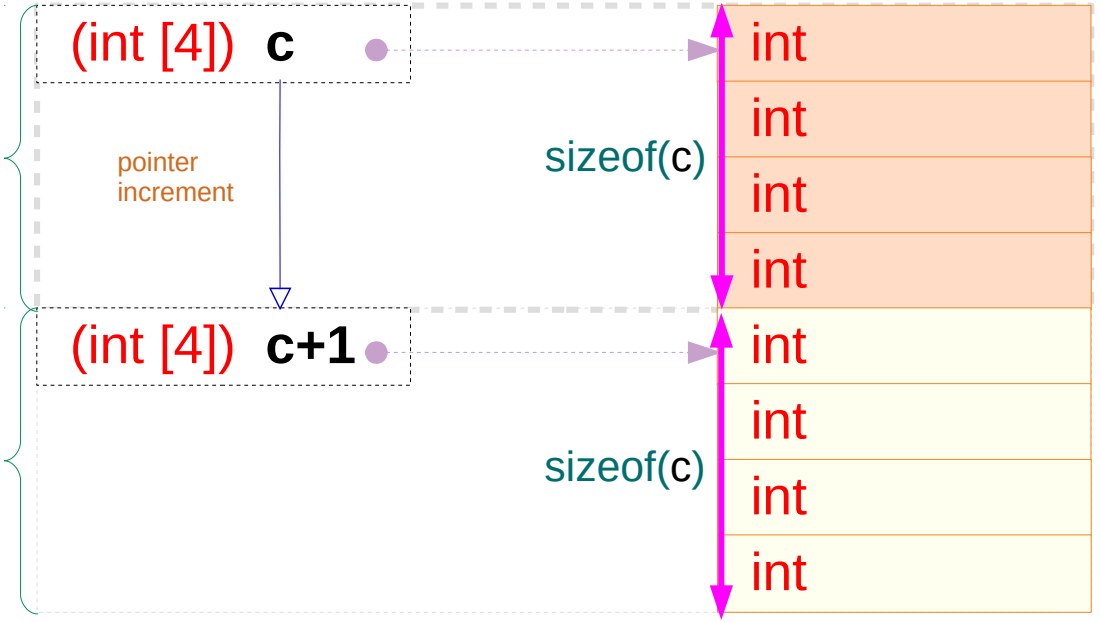
Sizes of integer pointers

a pointer to an **int**

sizeof(p) = pointer size
= 8 bytes on 64-bit machine
= 4 bytes on 32-bit machine

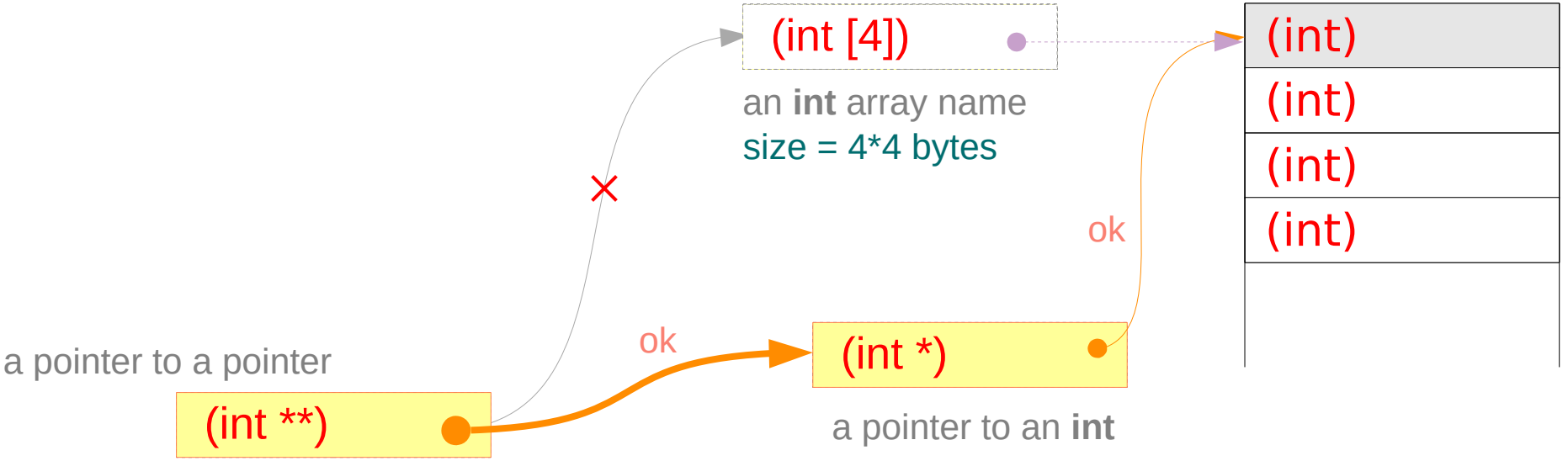
an **int array** name

an array :
an aggregate type
sizeof(c)
= sizeof(*c) * 4
= sizeof(int) * 4
= 4*4 = 16 bytes



type(int [4]) = type(int [ ]) = (int *)

Double integer pointer type – **(int **)**

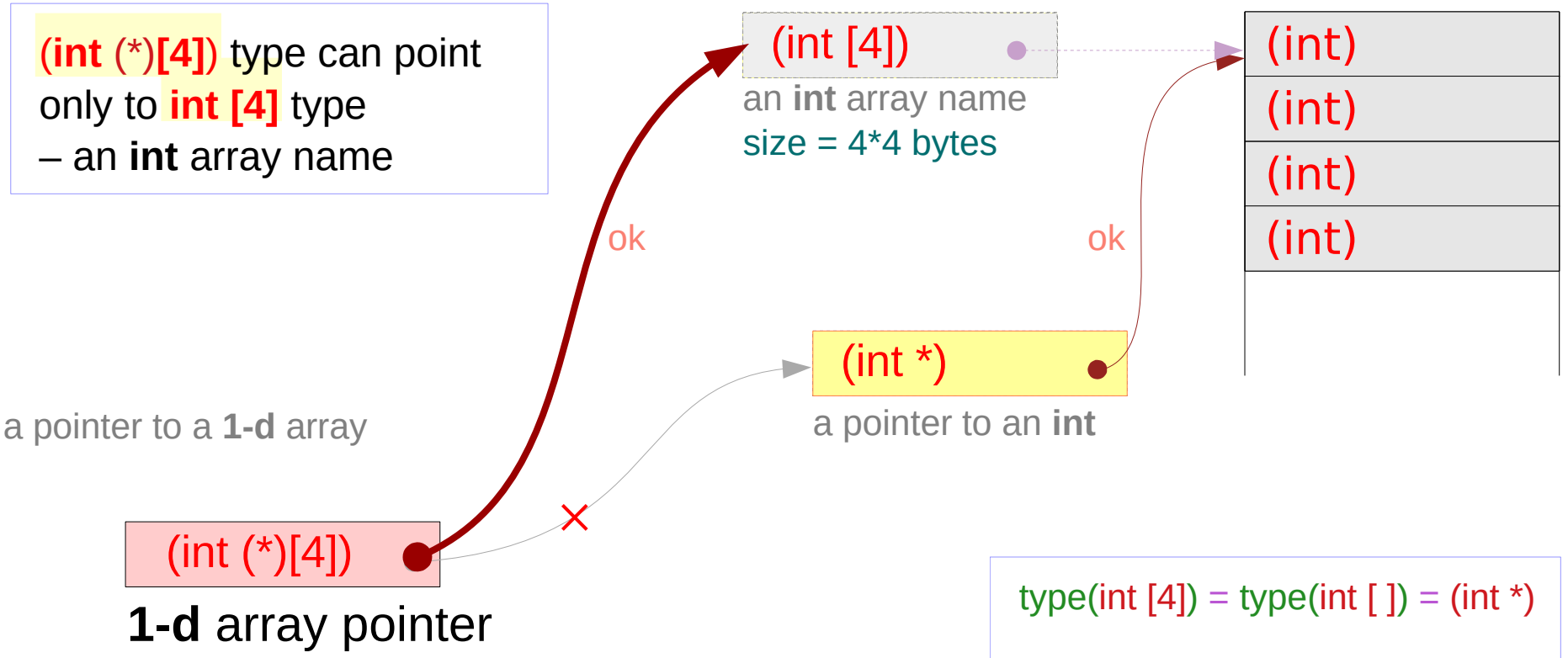


(int **) type can point only to **(int *)** type – an **int** array name (X)

type(int [4]) = type(int [ ]) = (int *)

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

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



Array Pointers

Pointer to an array – variable declarations

```
int   m ;
int  *n ;
```

an integer pointer

Array Pointer Approach
(pointer to arrays)

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

an array pointer

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

a function pointer

Pointer to an array – a type view

int 4 byte data

int *

an integer pointer

array pointer:
a pointer to an array

pointer array:
an array of pointers

int [4] 4*4 byte data

int (*) [4]

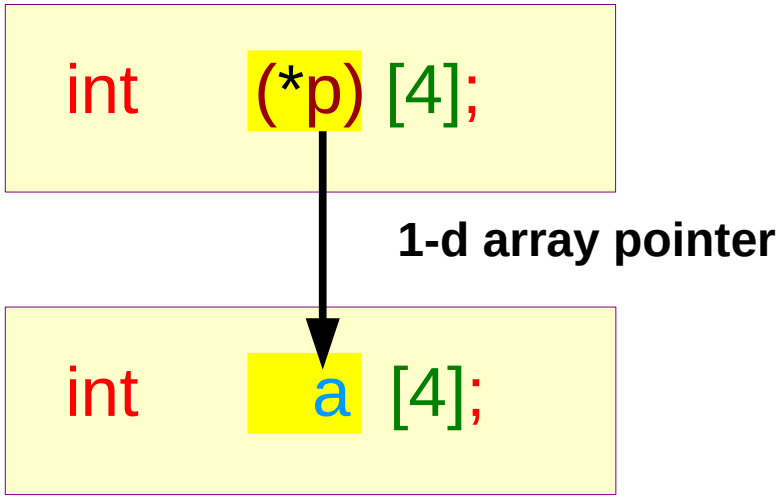
an array pointer

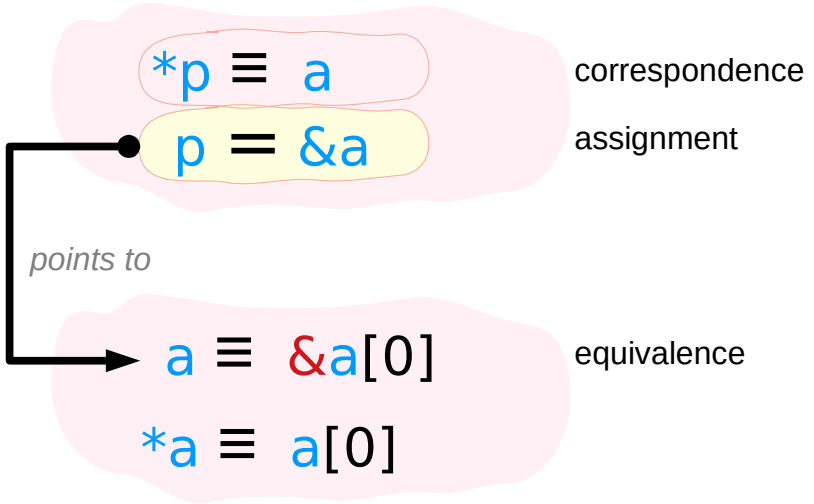
int (int, int) instructions

int (*) (int, int)

a function pointer

Pointer to a **1-d** array – (1) type declarations





&a and a print
the same address
but have different types

value(&a) = value(a)

type(&a) ≠ type(a)

int (*)[4] ≠ int [4]

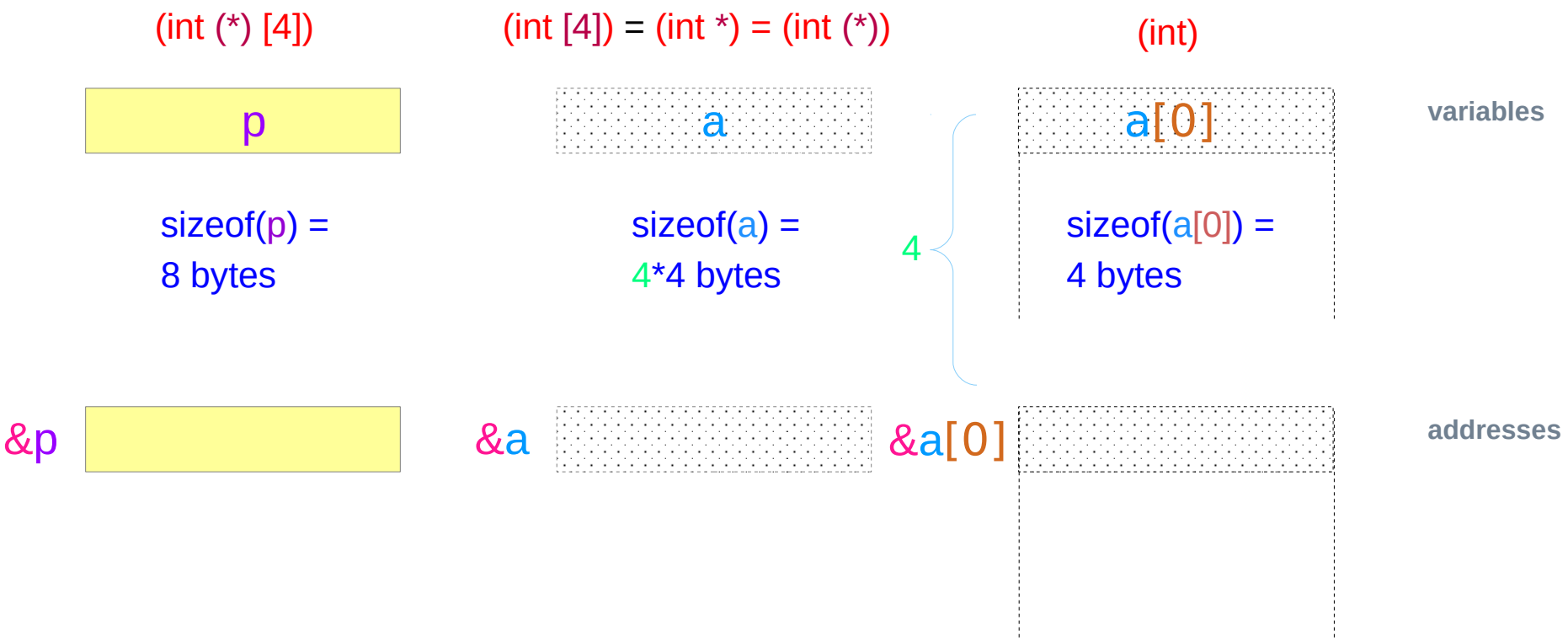
those values are evaluated as addresses

Pointer to a **1-d** array – (2) types and sizes

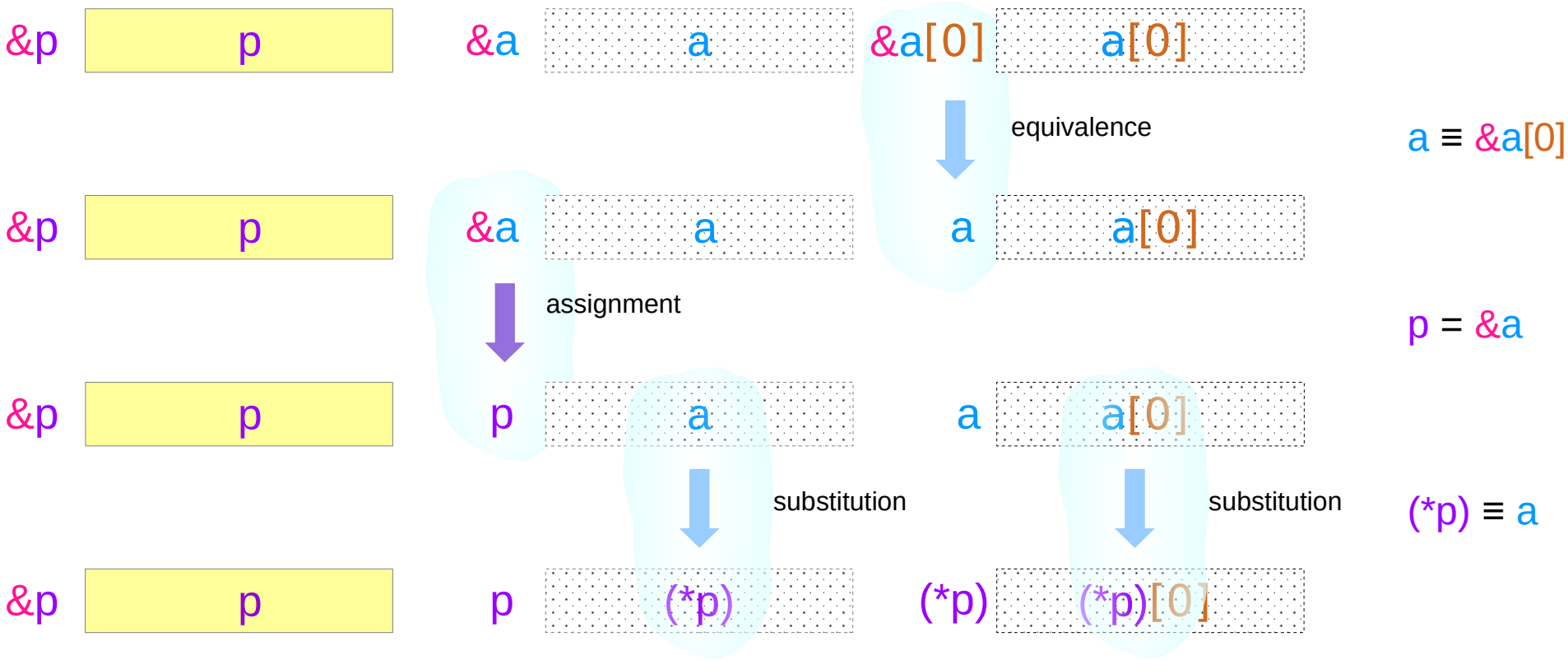
int a [4];
int (*p) [4];

assignment: p = &a

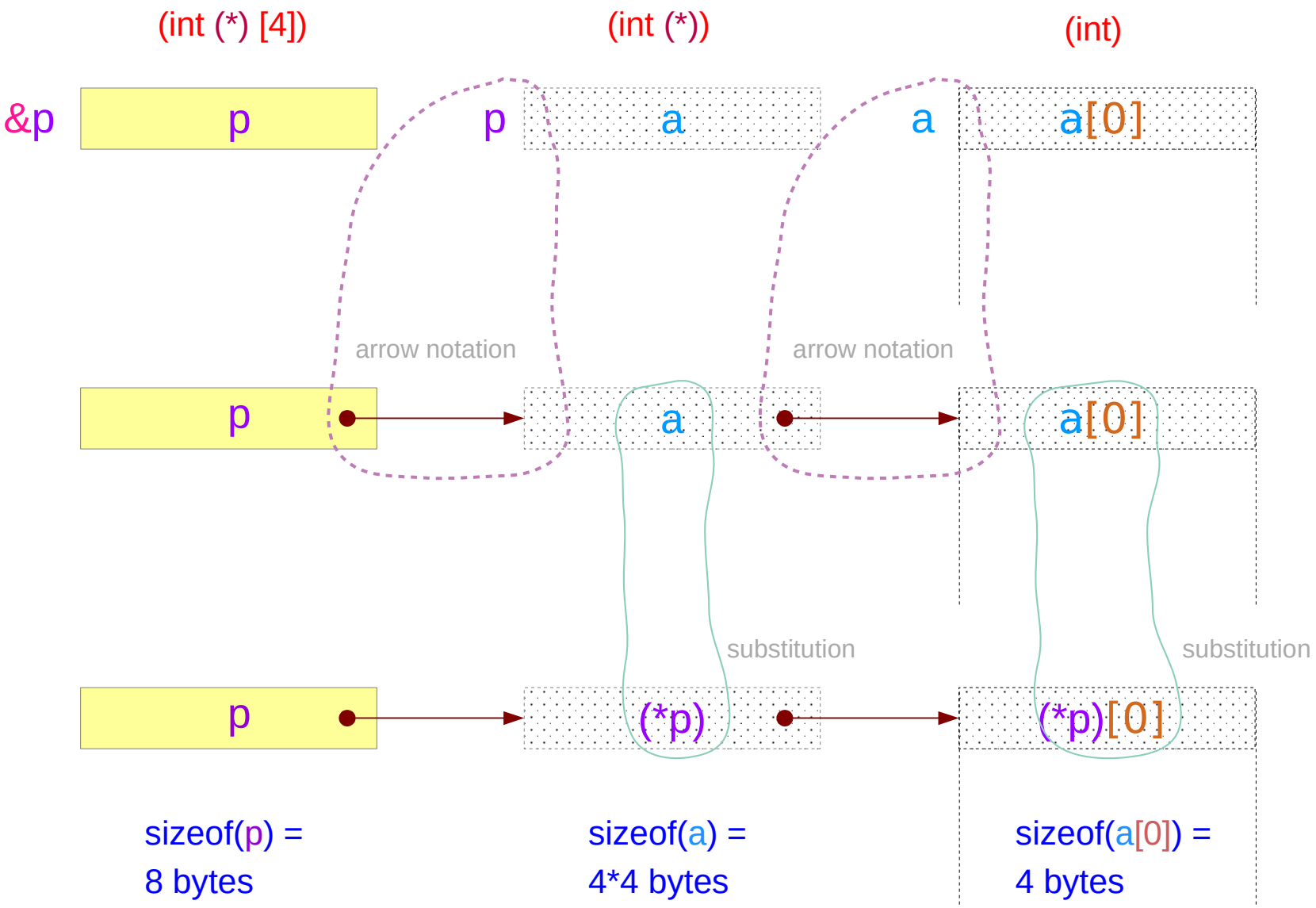
equivalence: a ≡ &a[0]



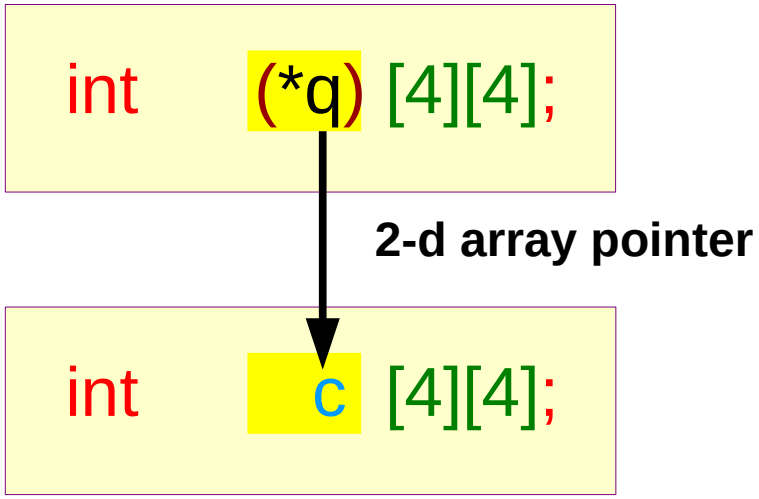
Pointer to a **1-d** array – (3) an assignment & equivalences

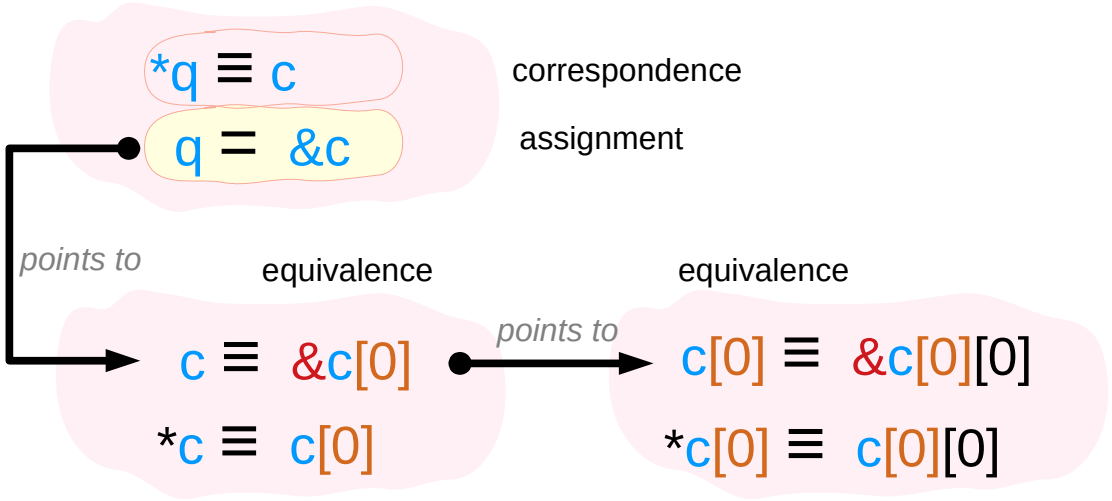


Pointer to a **1-d** array – (4) a chain of pointers view



Pointer to a **2-d** array – (1) type declarations





&c, c, c[0] print
the same address
but have different types

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

type(&c) ≠ type(c) ≠ type(c[0])

int (*)[4][4] ≠ int [4][4] ≠ int [4]

those values are evaluated as addresses

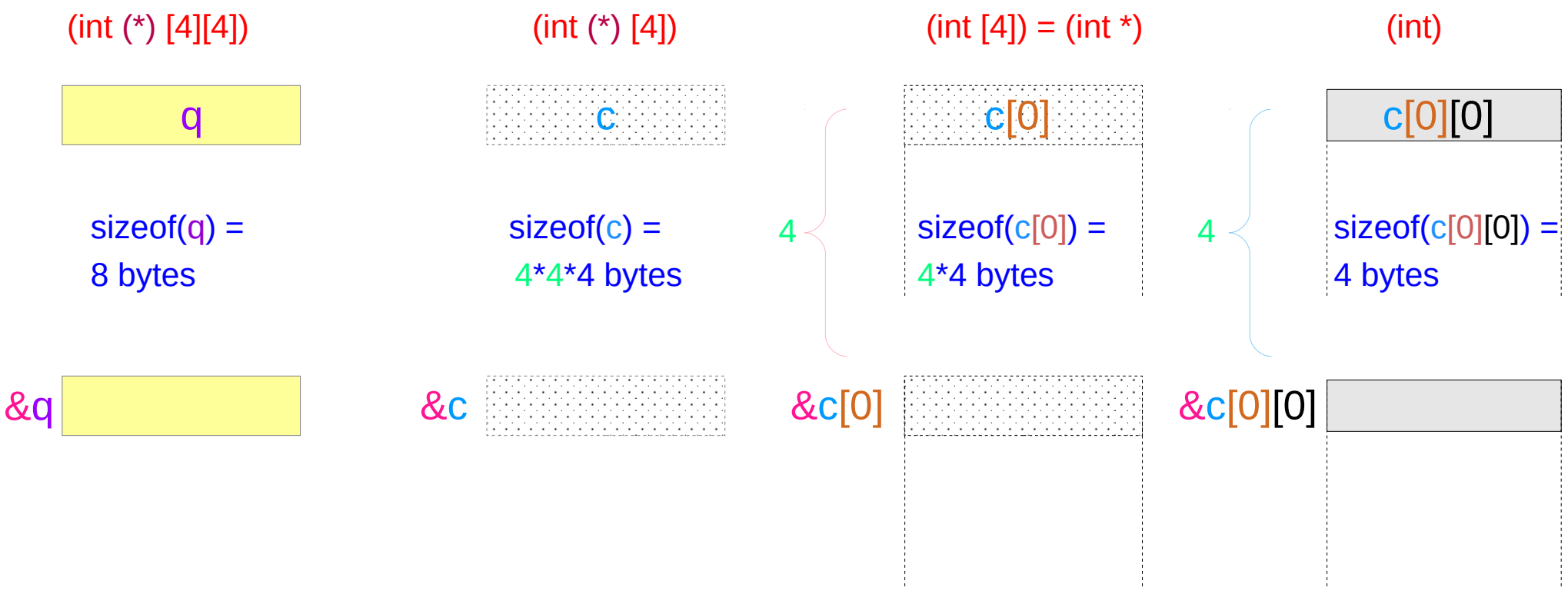
Pointer to a **2-d** array – (2) types and sizes

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

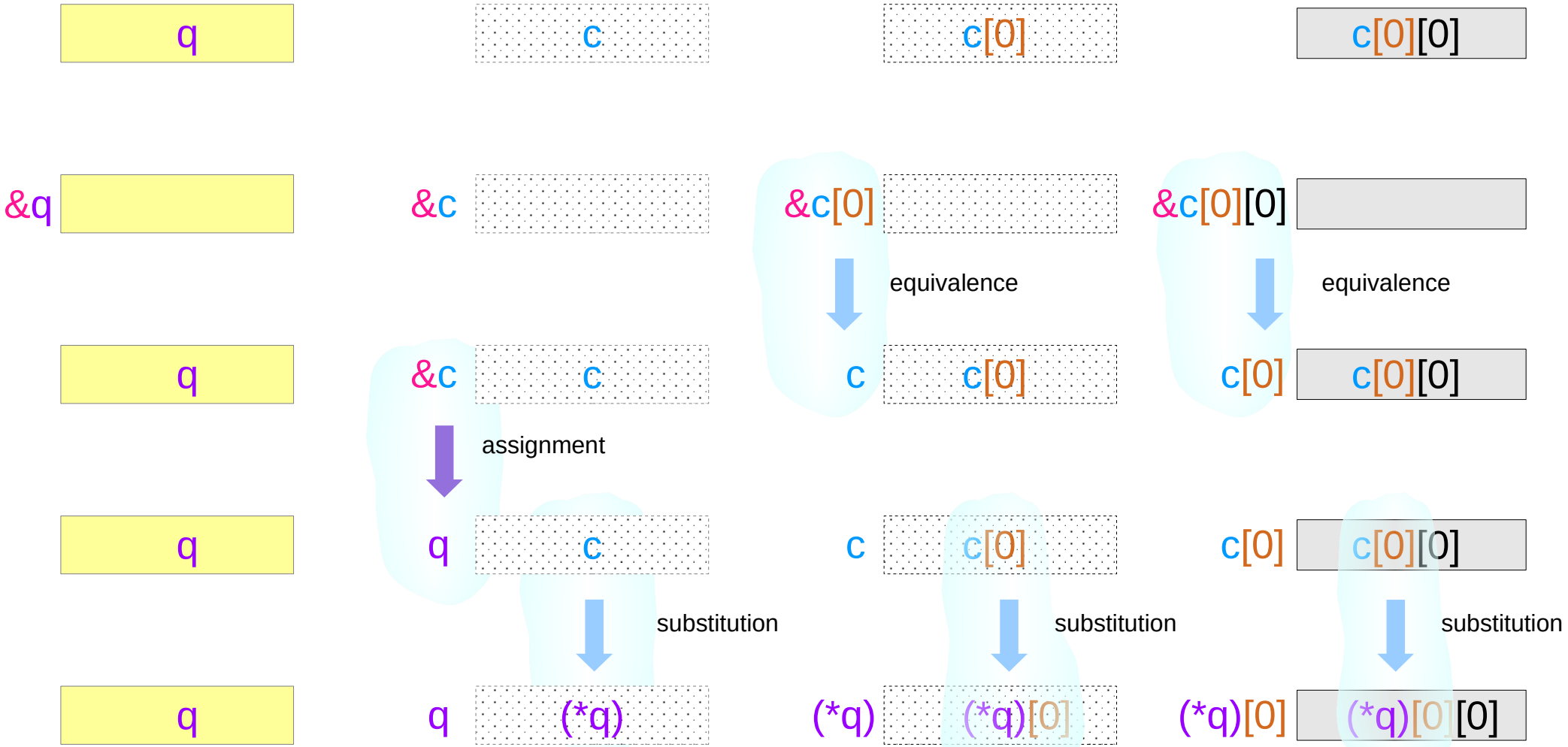
assignment: q = &c

equivalence: c ≡ &c[0]

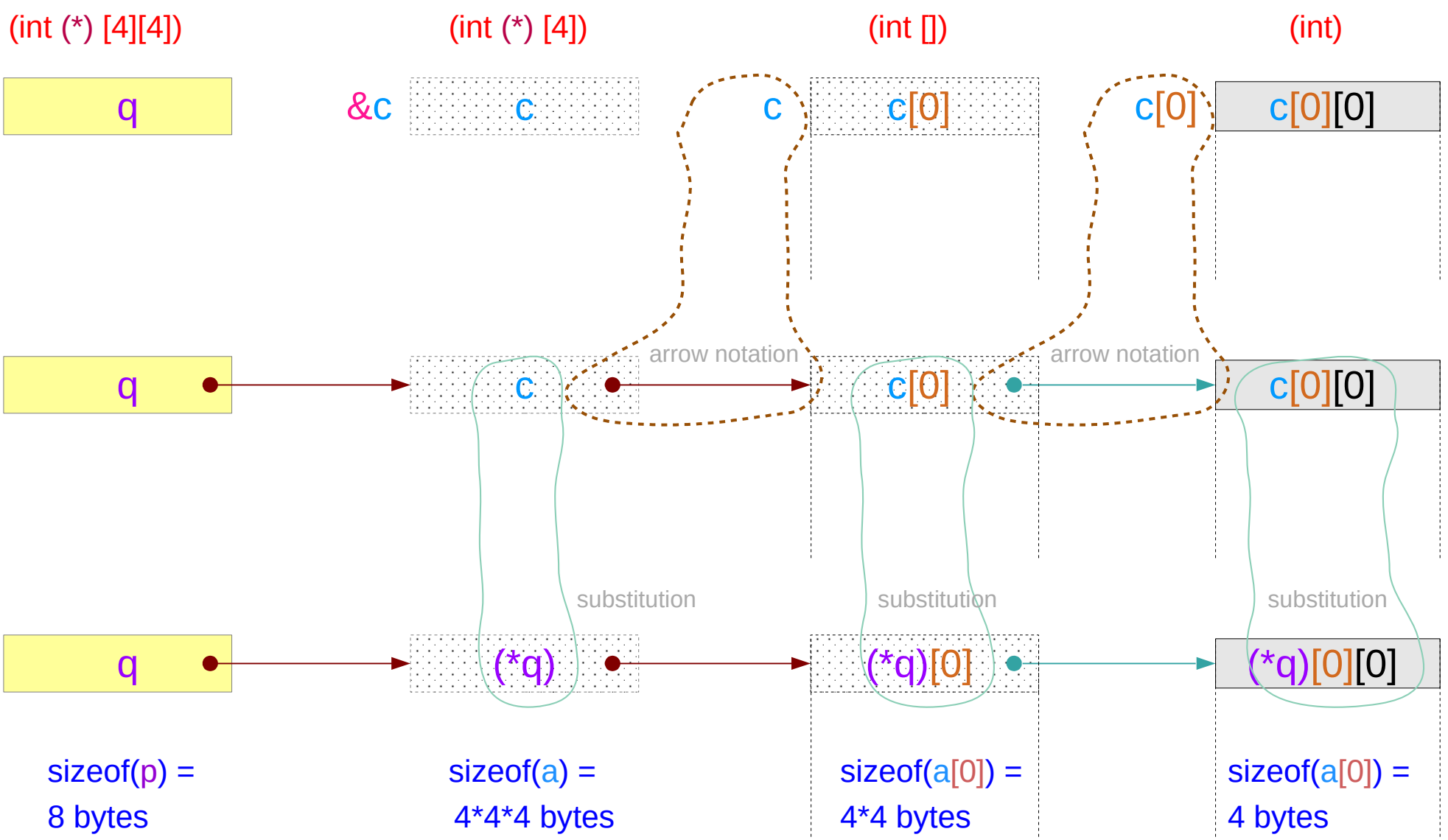
equivalence: c[0] ≡ &c[0][0]



Pointer to a **2-d** array – (3) an assignment & equivalences

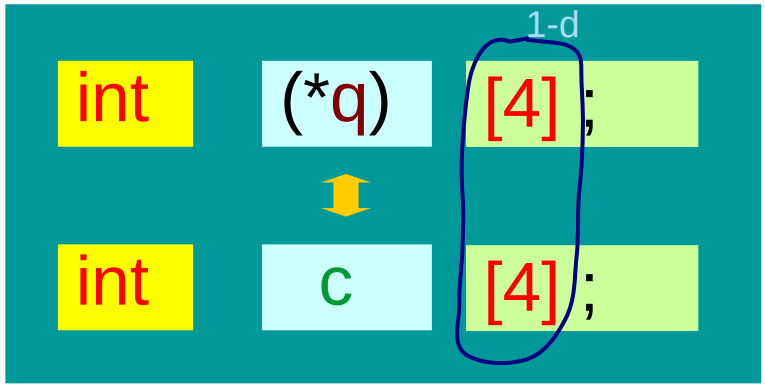


Pointer to a **2-d** array – (4) a chain of pointers view



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

1-d array pointer

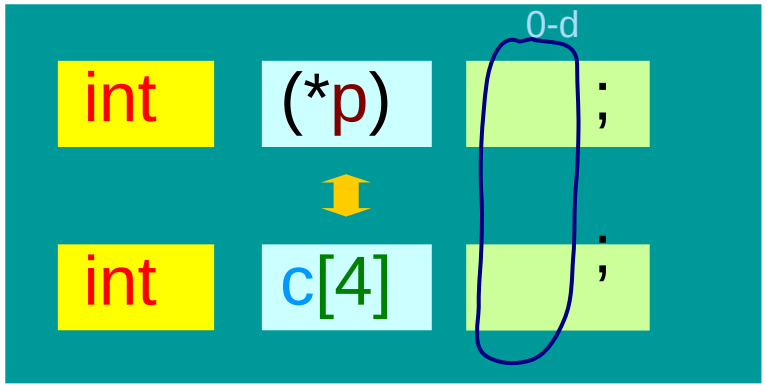


correspondence *q ≡ c;

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

(*q)[**i**] ≡ q[0][**i**] ≡ c[**i**]

0-d array pointer : int pointer



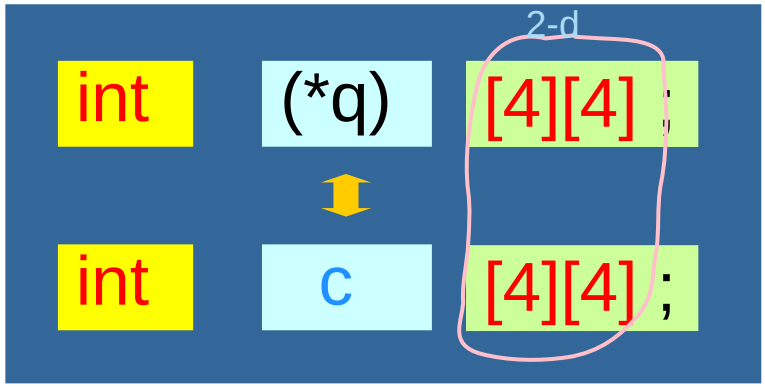
correspondence *p ≡ *c;

(int (*)) p = c;

p[**i**] ≡ c[**i**]

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

2-d array pointer

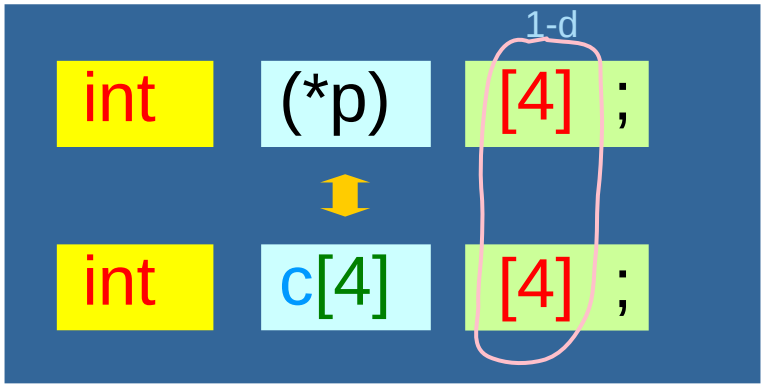


correspondence *q ≡ c;

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

(*q)[i][j] ≡ q[0][i][j] ≡ c[i][j]

1-d array pointer

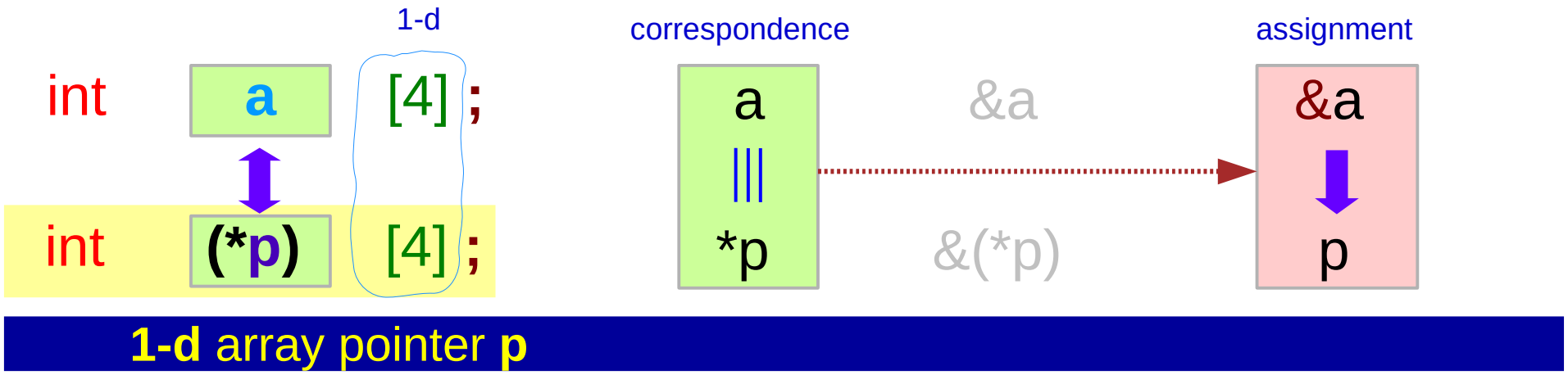


correspondence *p ≡ *c;

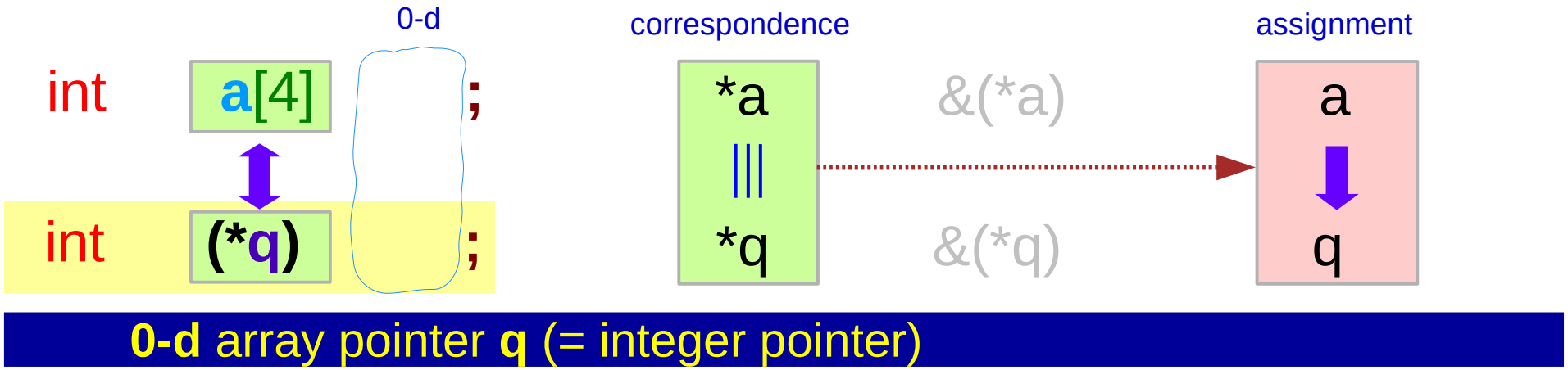
(int (*) [4]) p = c;

p[i] ≡ c[i]

Pointer types to a **1-d** array : 2 cases

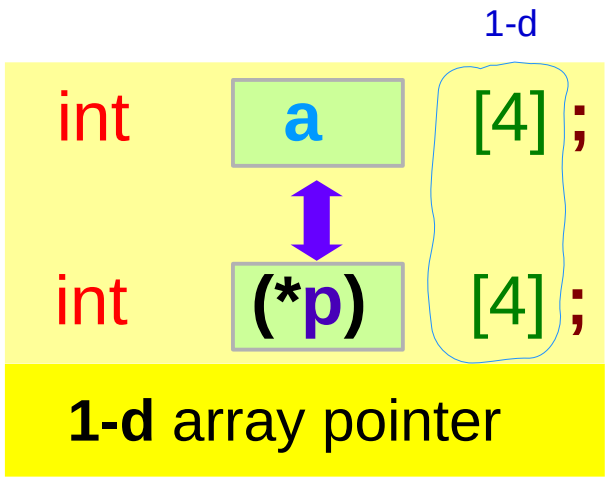


1-d array pointer **p**



0-d array pointer **q** (= integer pointer)

Pointer types to a **1-d** array : sizes of pointer dereferences



assignment

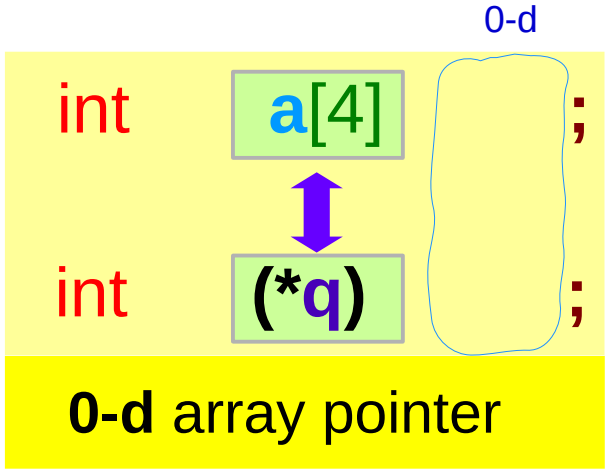
p = &a;

substitution

(*p)[i] ≡ p[0][i] ≡ a[i]

sizeof(p) = 4 or 8 bytes : the size of a pointer

sizeof(*p) = 4*4 bytes : the size of an 1-d array



assignment

q = a;

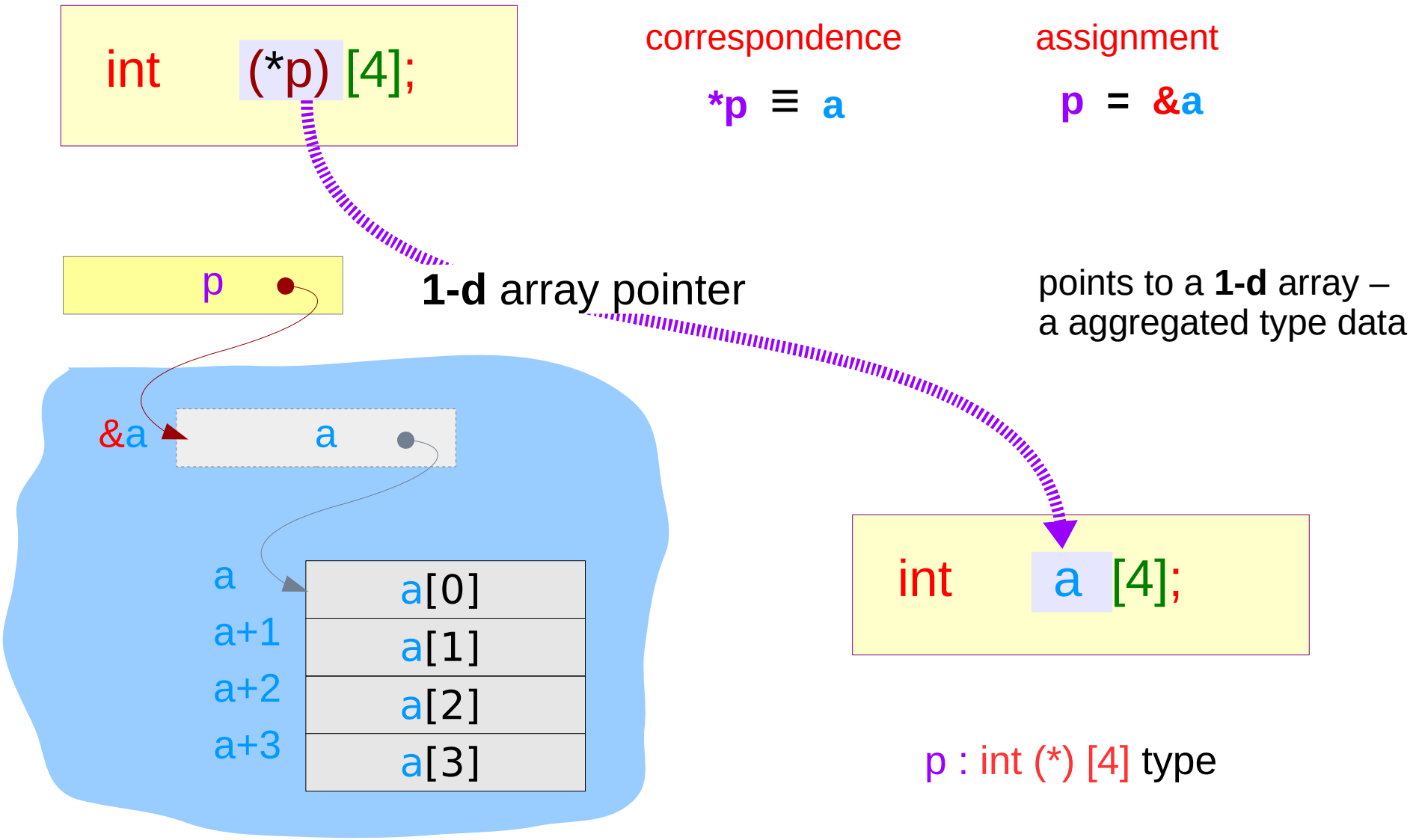
substitution

q[i] ≡ a[i]

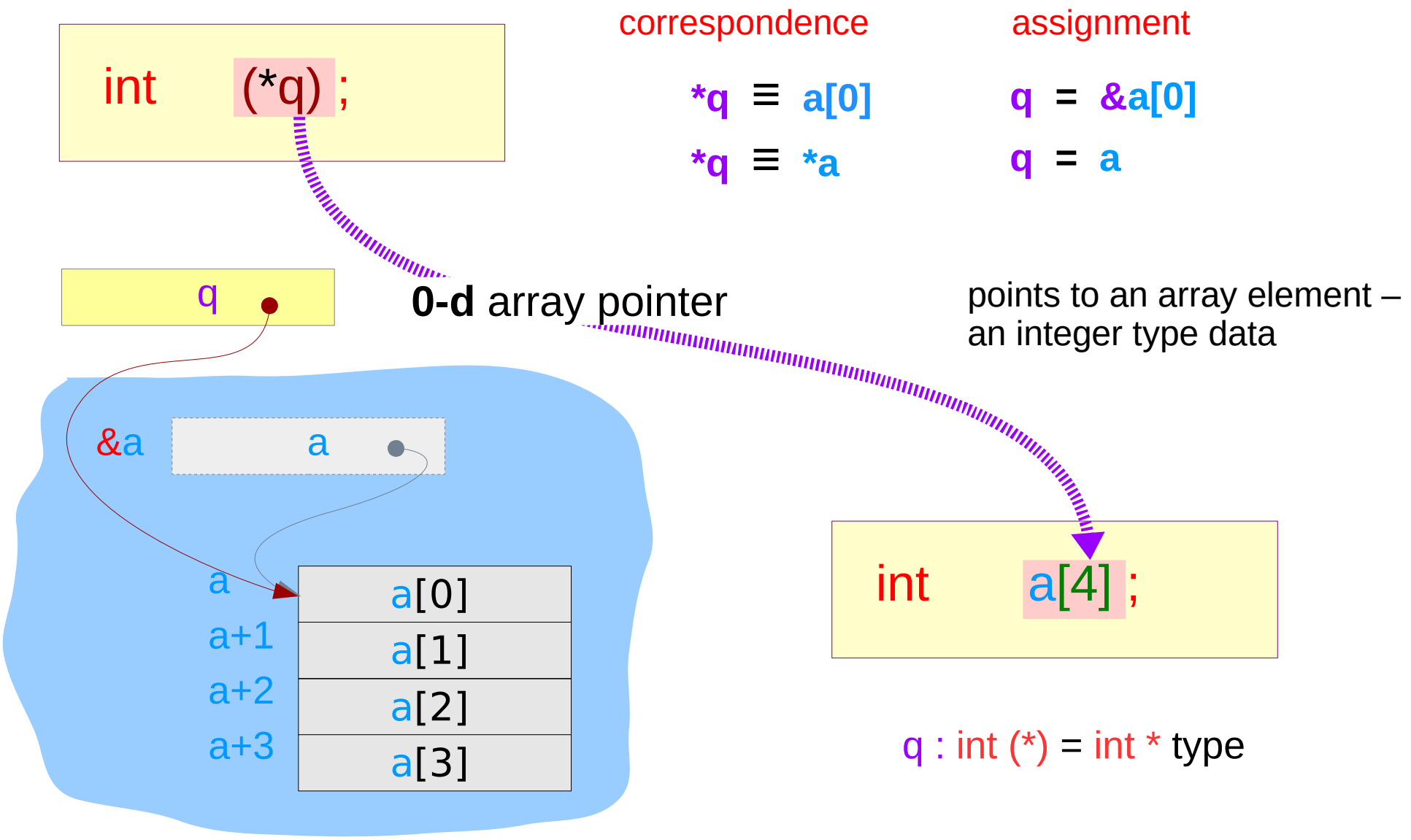
sizeof(q) = 4 or 8 bytes : the size of a pointer

sizeof(*q) = 4 bytes : the size of a 0-d array (int)

1-d pointer to a 1-d array – a variable view



0-d pointer to a 1-d array – a variable view



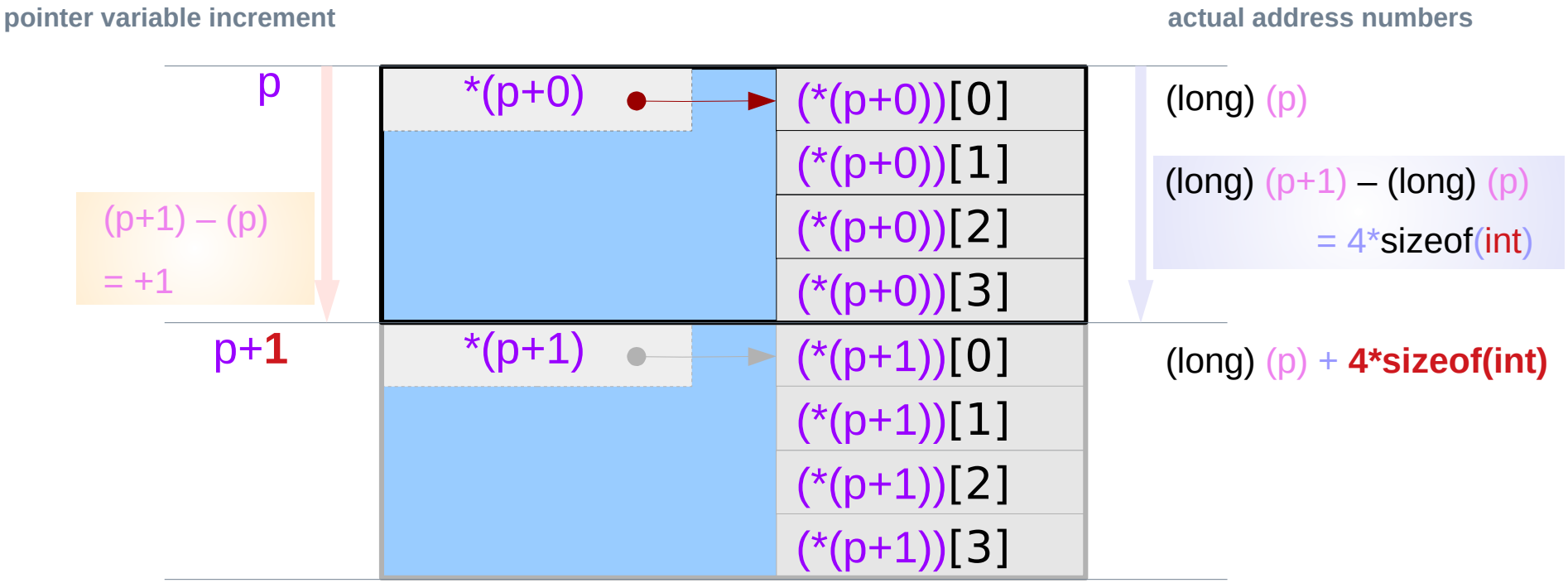
Incrementing a **1-d** array pointer

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

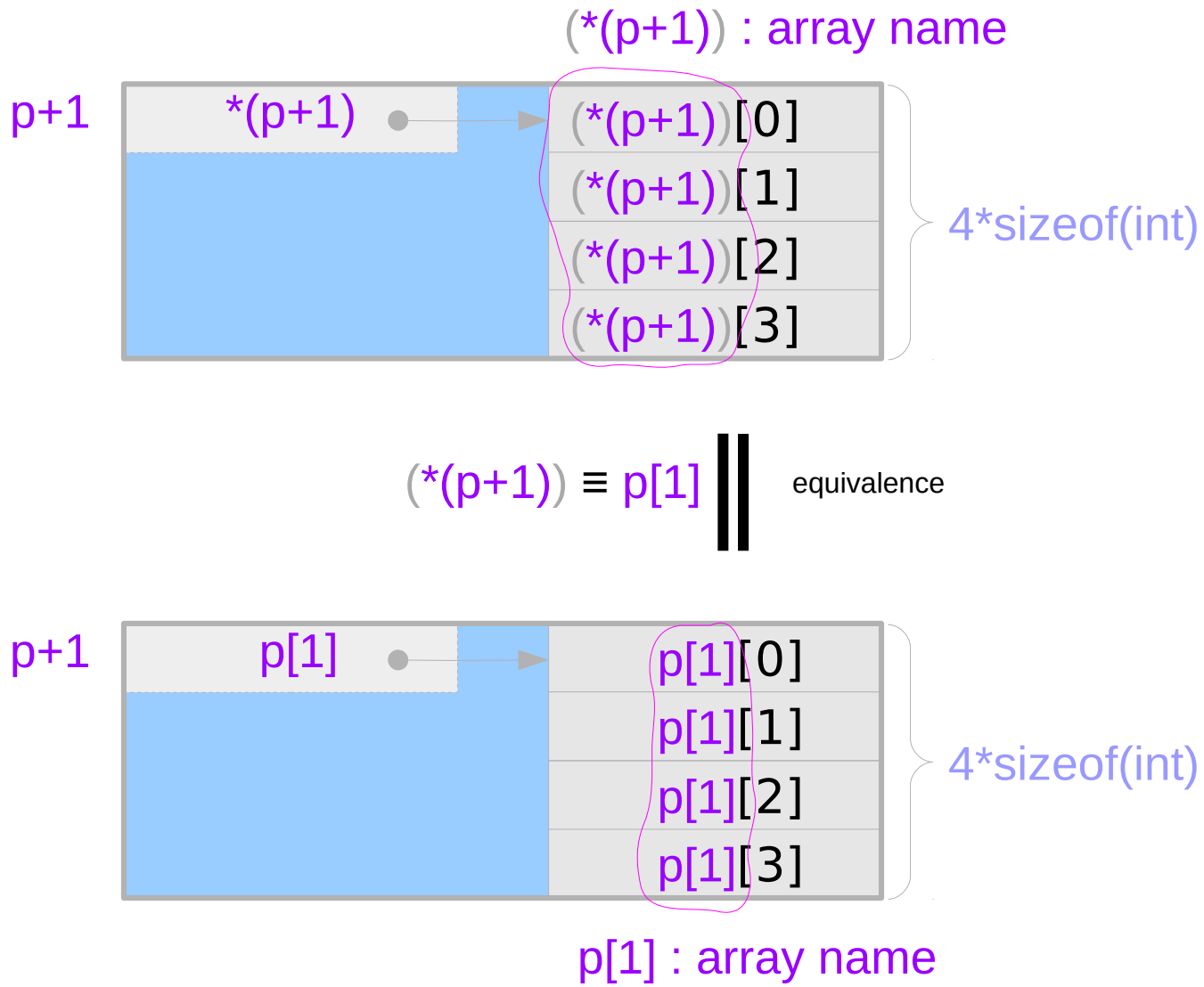
value(p+1) – **value**(p) = sizeof(*p)

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

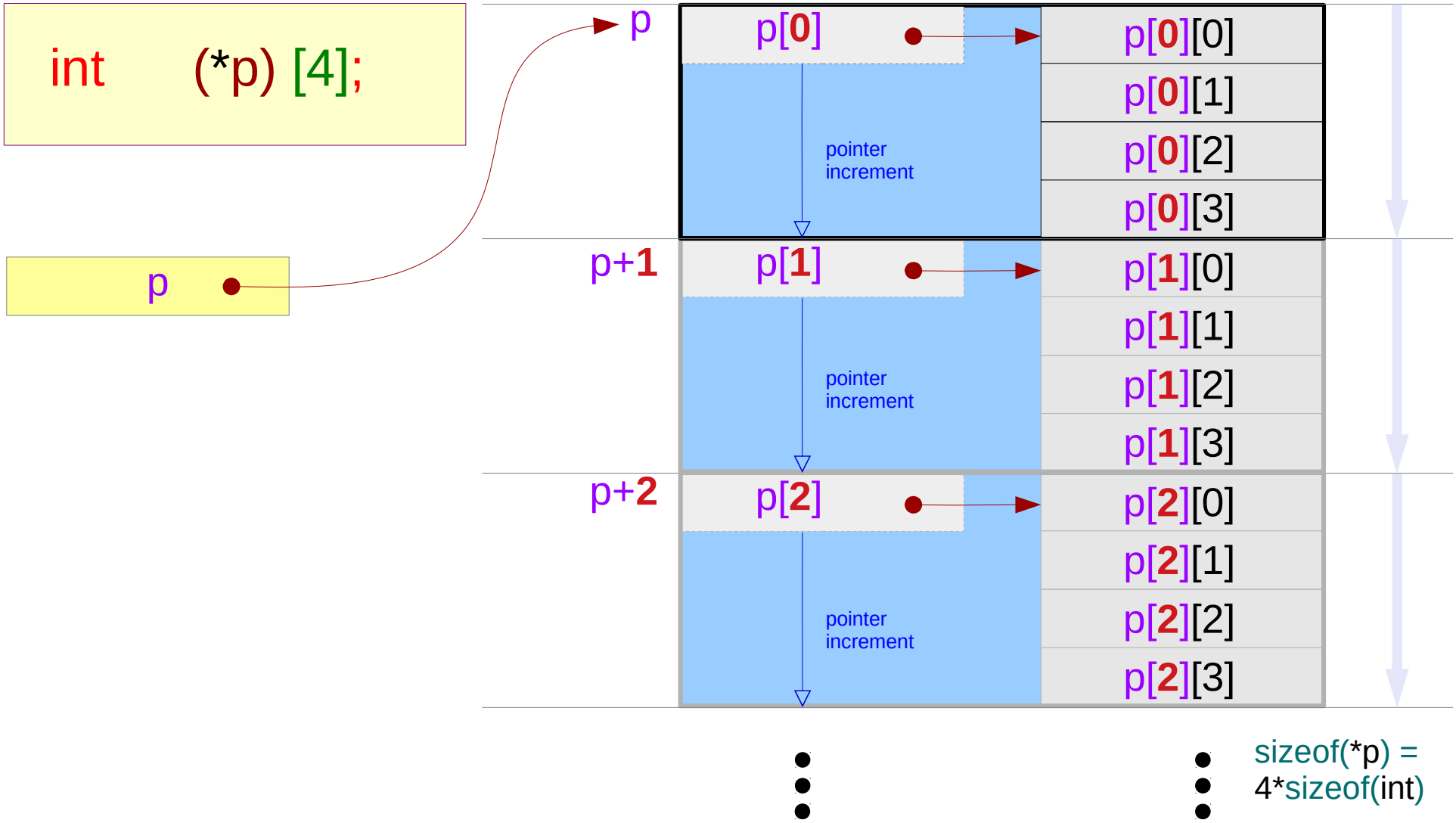
Aggregate Type Size



Incrementing a **1-d** array pointer – extending a dimension



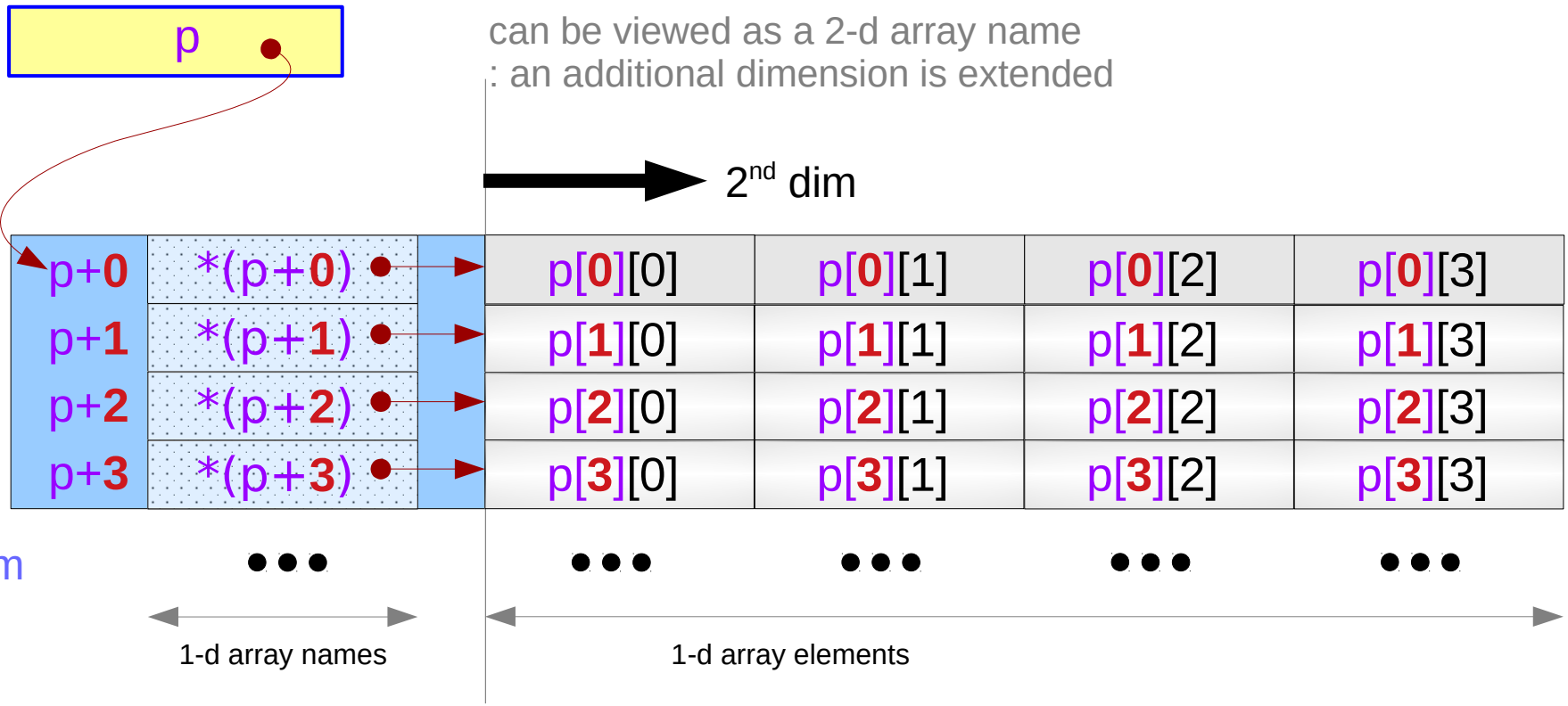
Substitution using a **1-d** array pointer



A **1-d** array pointer – extending a dimension

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

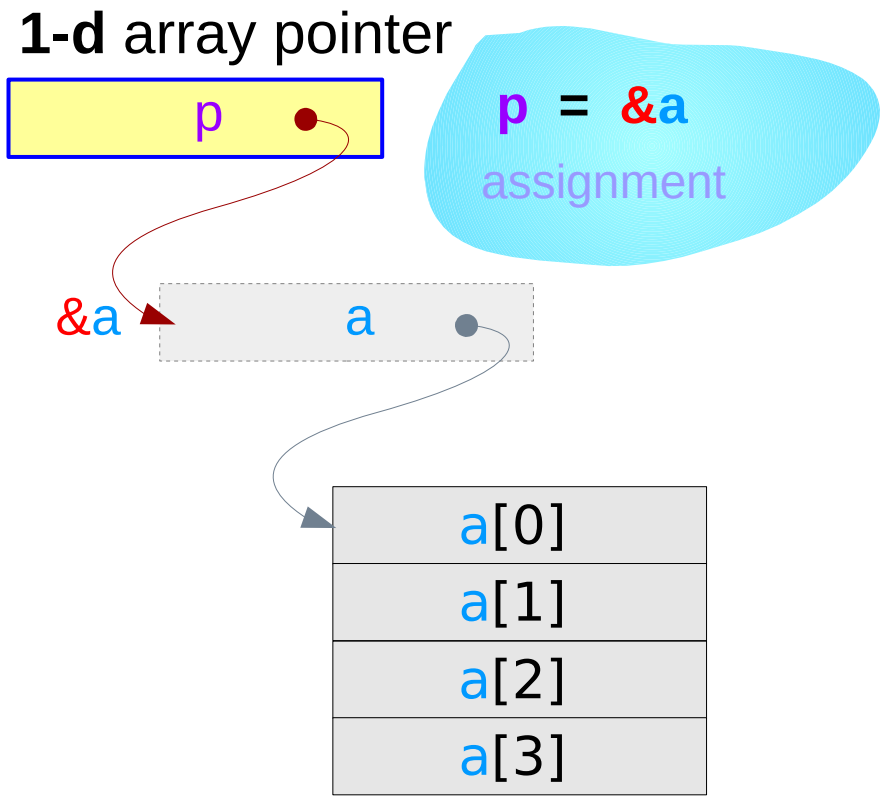
1-d array pointer

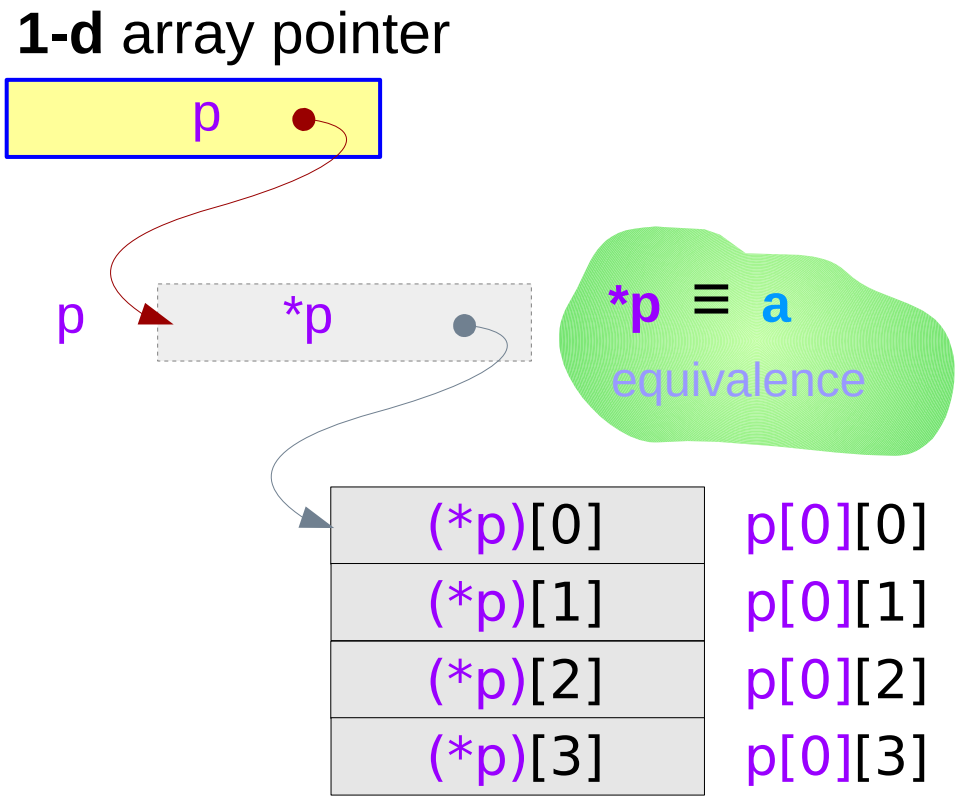


A **1-d** array pointer and a **1-d** array

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

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



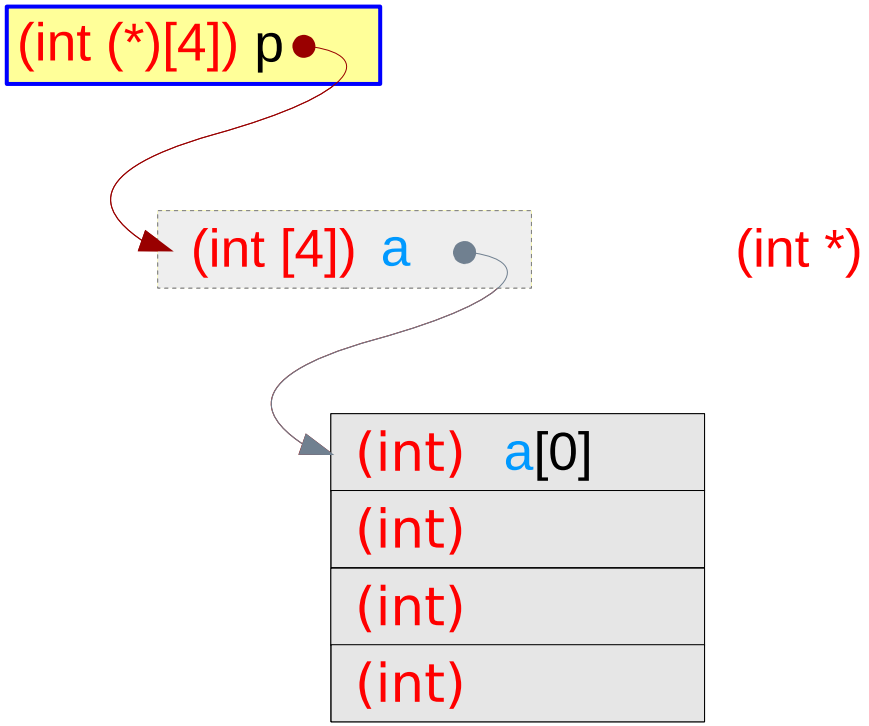


A **1-d** array pointer and a **1-d** array – a type view

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

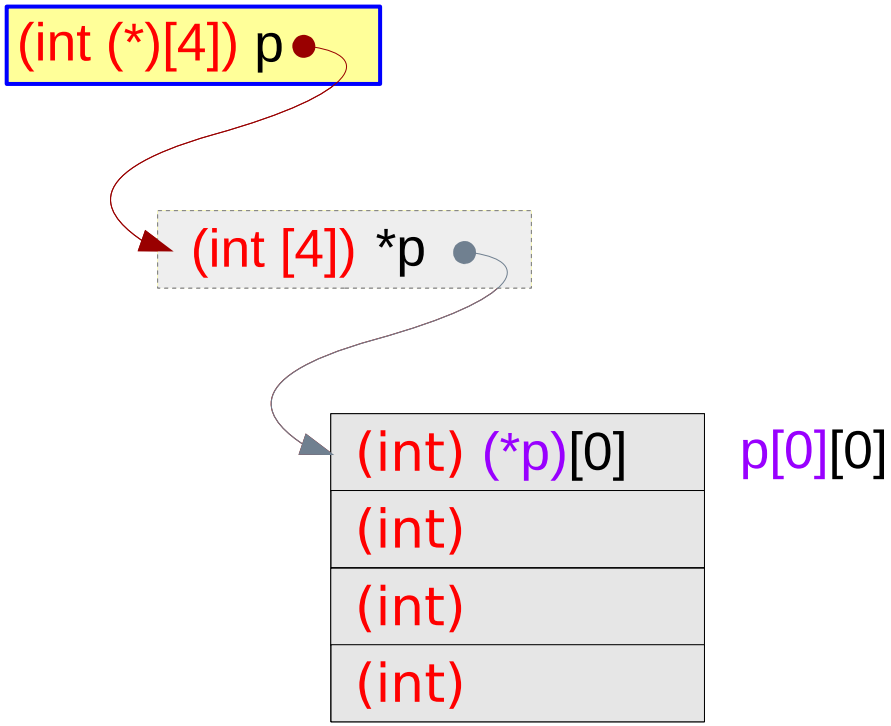
```
int  (*p) [4]  = &a;
```

1-d array pointer



(int *)

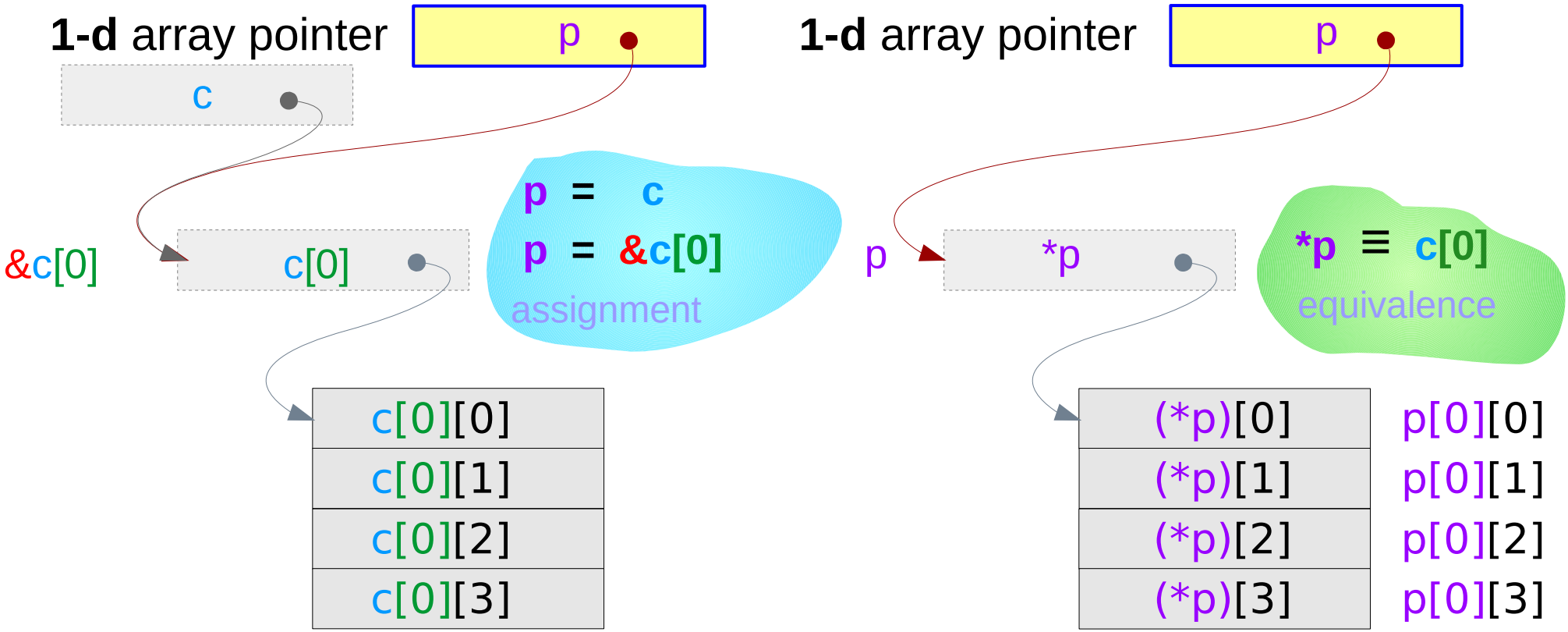
1-d array pointer



A **1-d** array pointer and a **2-d** array

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

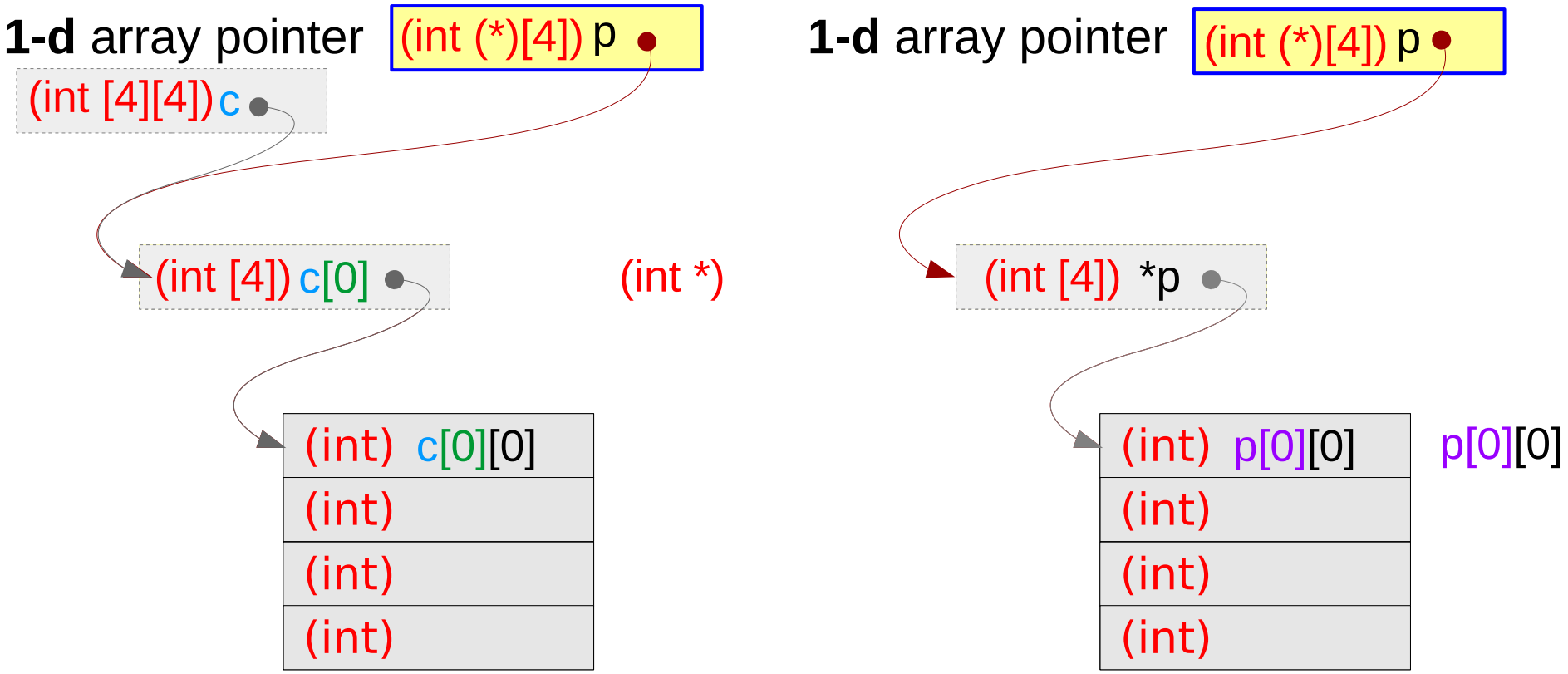
```
int  (*p) [4]  = &c[0];
```



A **1-d** array pointer and a **2-d** array – a type view

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

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



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