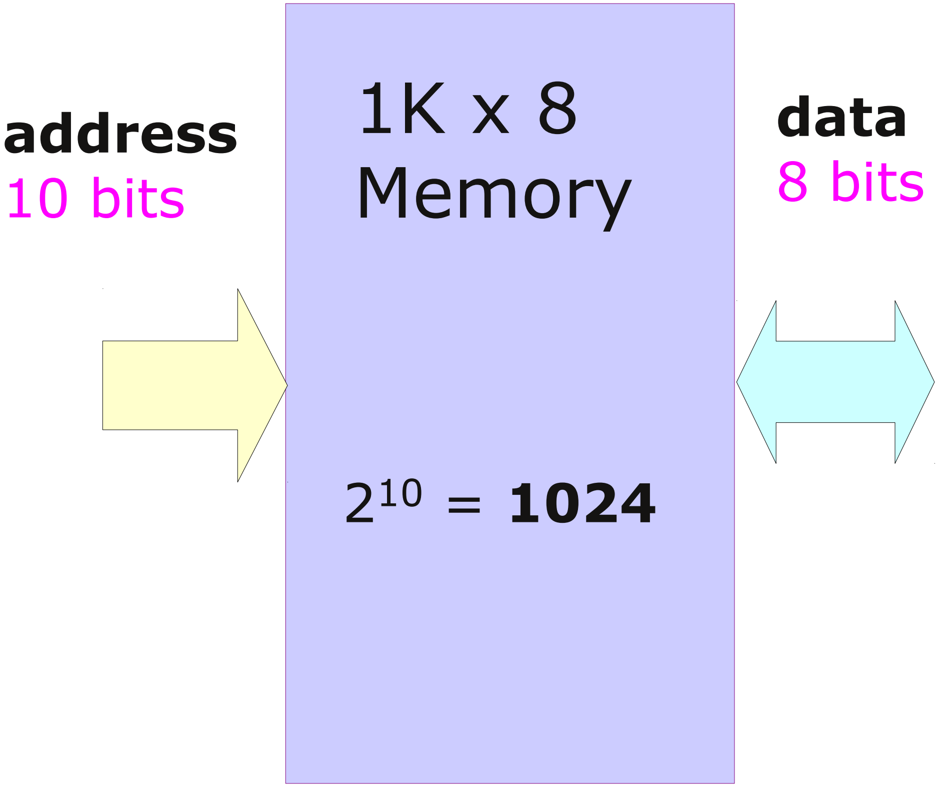
Pointer (1A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

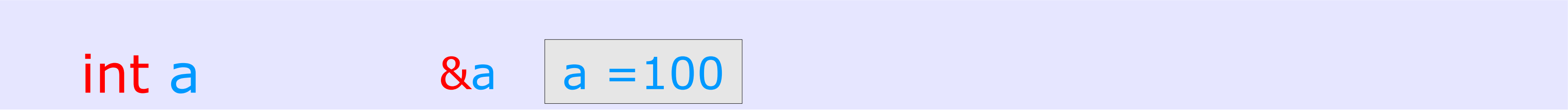
This document was produced by using OpenOffice.

Address and Data in a Memory

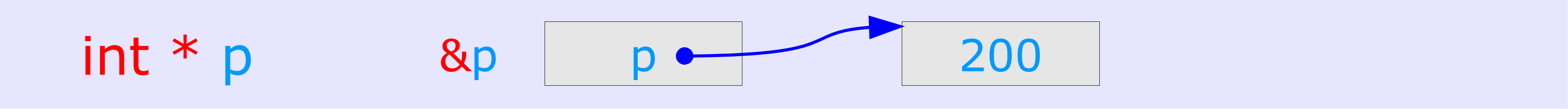


	address	data
1023	0x3FF	
1022	0x3FE	
1021	0x3FD	
1020	0x3FC	
		• • •
0003	0x003	
0002	0x002	
0001	0x001	
0000	0x000	
DEC address	HEX address 10 bits	data 8 bits

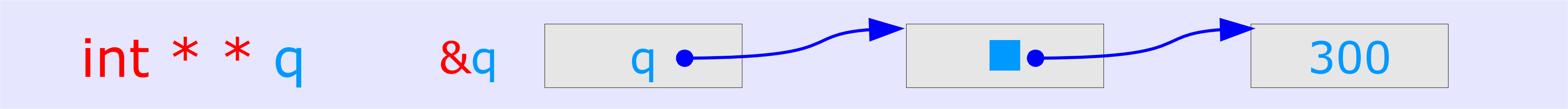
Variable



The variable a holds an integer data



The **pointer** variable p holds an address,
which is the address of an integer data



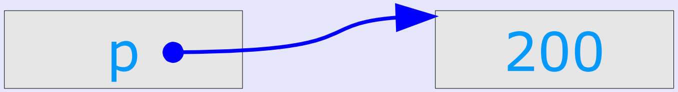
The **pointer** variable q holds an address,
where another address is stored,
which is the address of an integer data

Access Data Via Pointer Variables

int a &a a =100

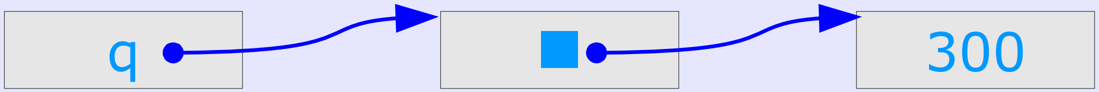
	address	value	
Direct Access	&a	a	integer

int * p &p



Indirect Access	address	value	
Dereference Op *	&p	p	address
content of a pointer	p	*p	integer

int * * q &q

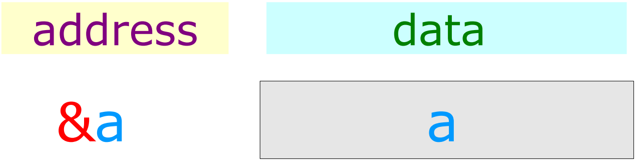


Double Indirect Access	address	value	
	&q	q	address
Dereference Op *	q	*q	address
content of a pointer	*q	**q	integer

Variable

```
int    a;
```

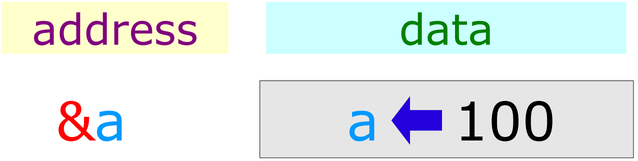
a can hold an *integer*



address	data
&a	a

```
a = 100;
```

a holds an *integer* 100

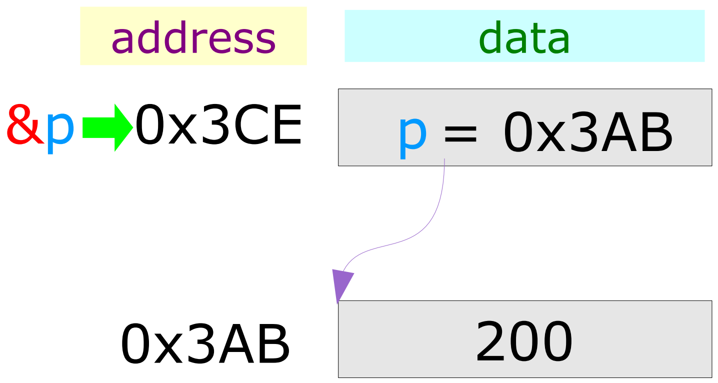


address	data
&a	a ⬅ 100

Pointer Variable

```
int *     p;
```

p holds an *address*



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

p holds an *address*
of Int type variable

$\underbrace{\texttt{int}}_{\textit{int}}\ \underbrace{\texttt{* p;}}$

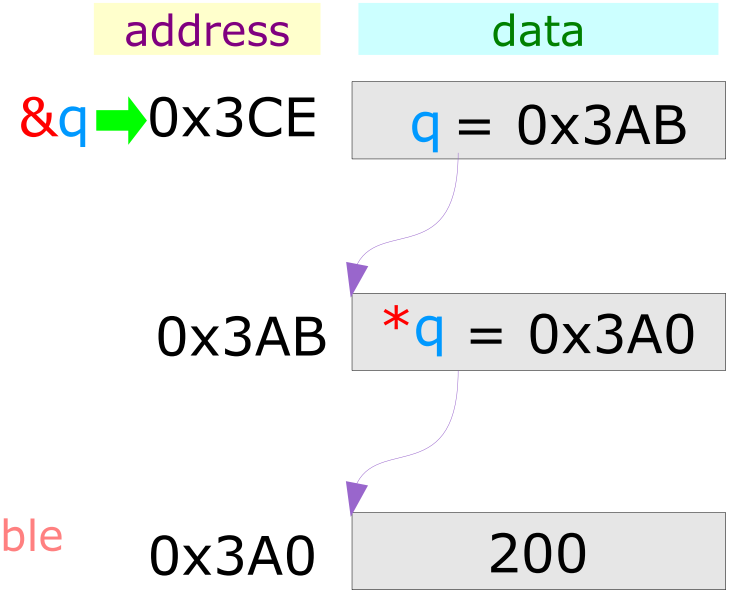
*p holds an *integer*

&p → 0x3CE
p → 0x3AB
*p → 200

Pointer to Pointer Variable

```
int **      q;
```

q holds an *address*



address

data

&q → 0x3CE | q = 0x3AB

0x3AB | *q = 0x3A0

0x3A0 | 200

`int **` `q;`

pointer to pointer to int

q holds an *address* of Pointer to Int type_variable

`int *` `*q;`

pointer to int

*q holds an *address* of Int type variable

`int` `**q;`

int

**q holds an *integer*

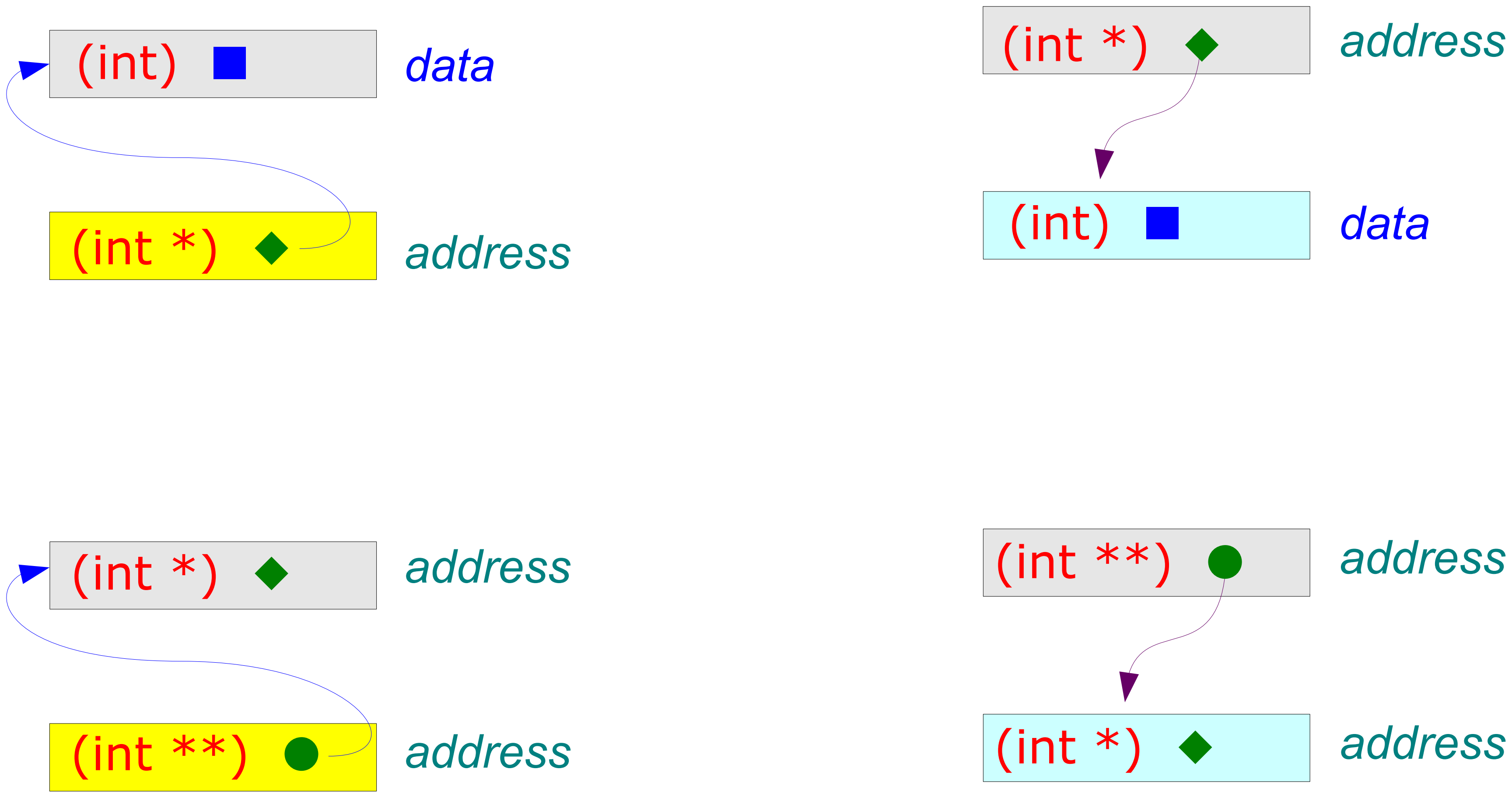
&q → 0x3CE

q → 0x3AB

*q → 0x3A0

**q → 200

Interpretation of Pointer (1)



(int) ■ *data*

(int *) ◆ *address*

(int *) ◆ *address*

(int) ■ *data*

(int *) ◆ *address*

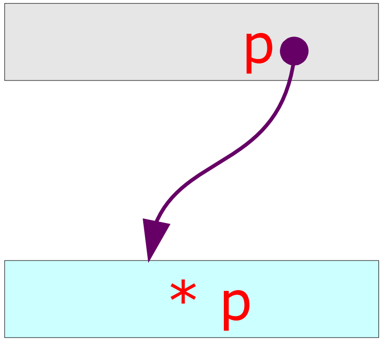
(int **) ● *address*

(int **) ● *address*

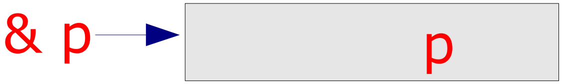
(int *) ◆ *address*

Interpretation of Pointer (2)

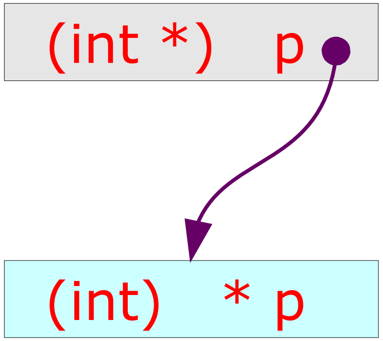
content of a pointer :
*Dereferencing operator **



The address of a variable :
Address of operator &



If p is a pointer to integer type



*If *p is an integer type*

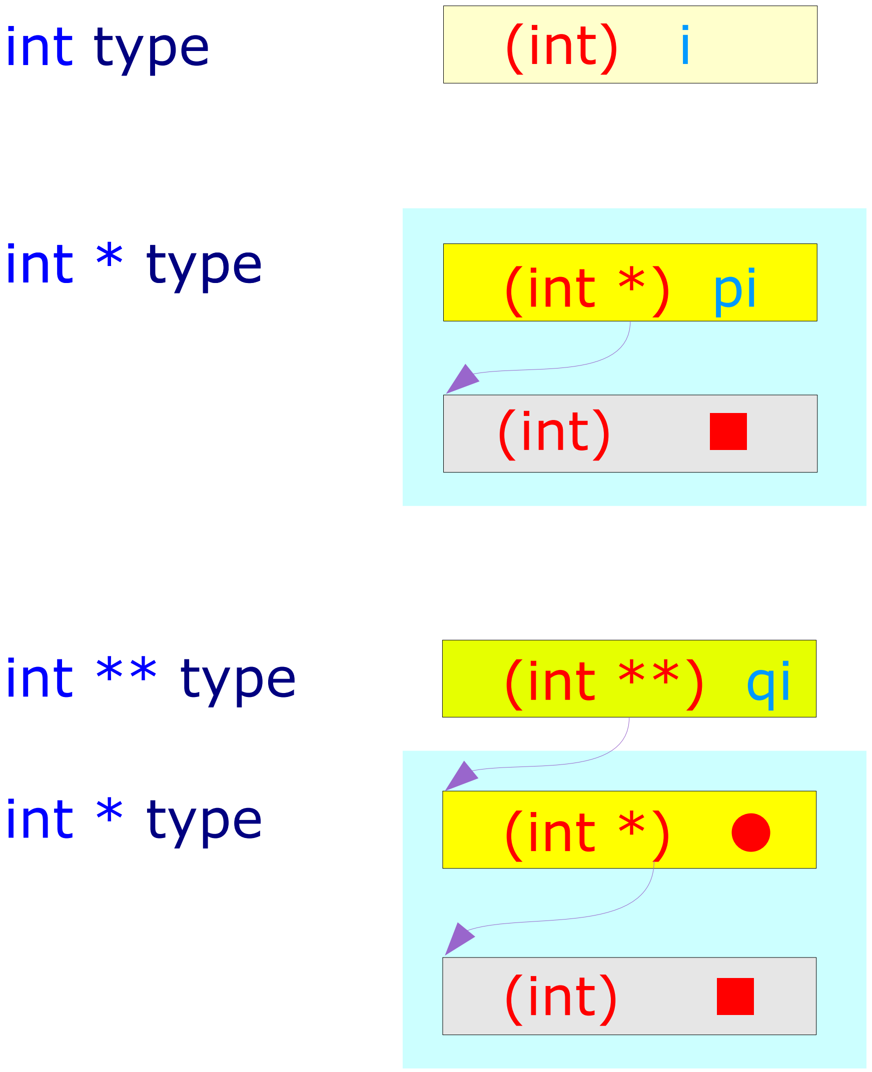
Integer Pointer Examples (1)

```
int      i;
int *    pi;
int **   qi;
```

i holds an ***integers***

pi holds an ***address***
of Int type

qi holds an ***address***
of Pointer to Int type



int type

(int) i

int * type

(int *) pi

(int) ■

int ** type

(int **) qi

int * type

(int *) ●

(int) ■

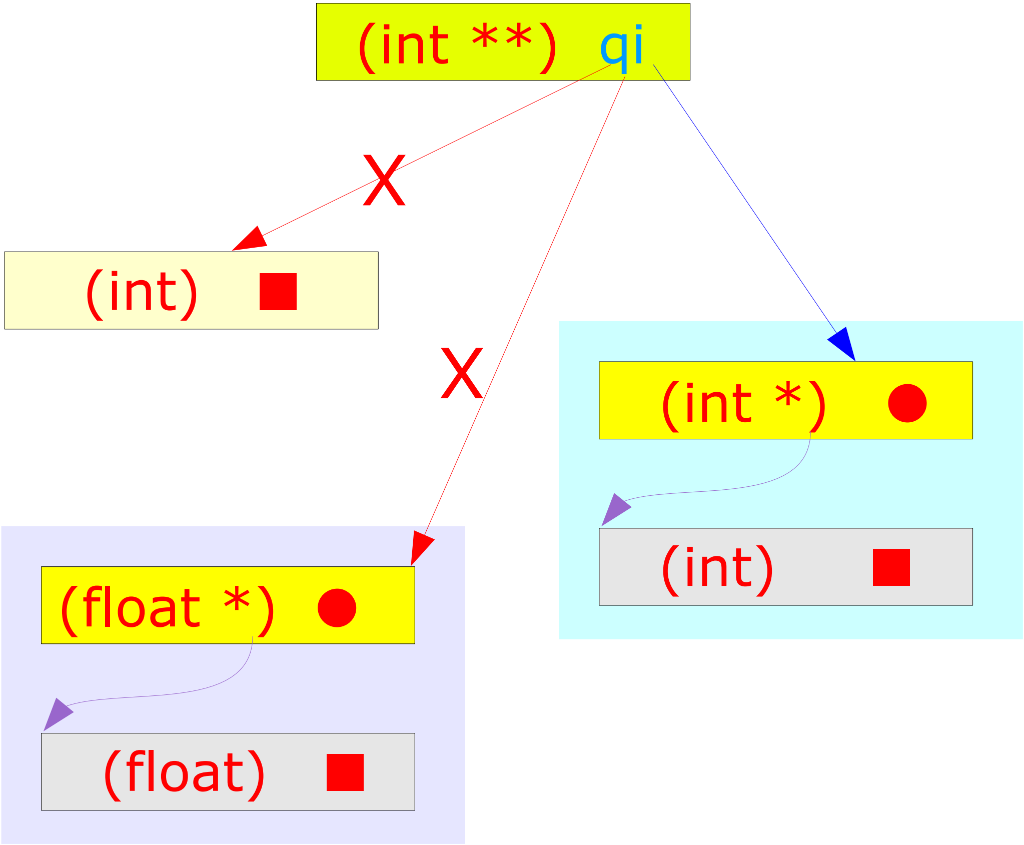
Integer Pointer Examples (2)

```
int      i;
int *    pi;
int **   qi;
```

i holds an *integers*

pi holds an *address*
of Int type

qi holds an *address*
of Pointer to Int type



Integer Pointer Examples (3)

```
int       i = 200;
int *     pi = &i;
int **    qi = &pi;
```

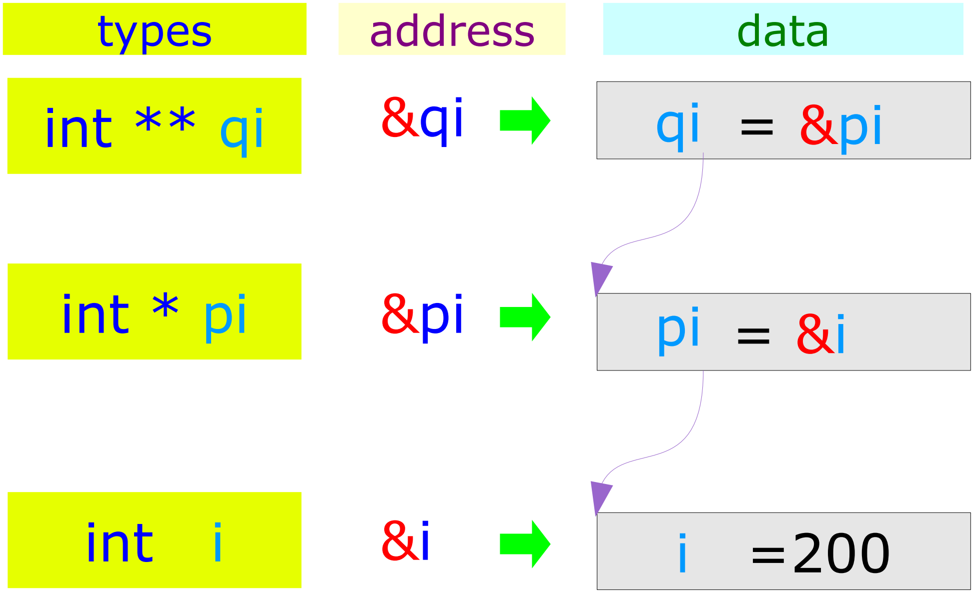
i holds an *integers*

pi holds an *address*

of Int type

qi holds an *address*

of Pointer to Int type



types	address	data
int ** qi	&qi	qi = &pi
int * pi	&pi	pi = &i
int i	&i	i =200

*qi = pi

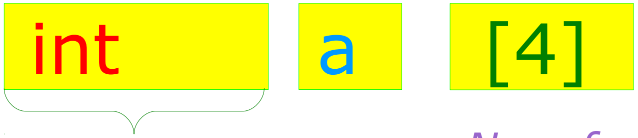
*pi = i

**qi = *pi = i

Array of Pointers (1)

```
int       a [4];

int *     b [4];
```

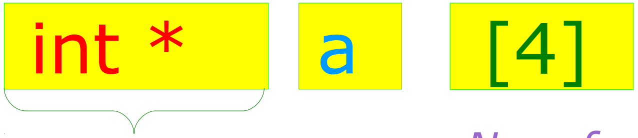
Array name a holds the starting *address*



No. of elements = 4

Type of each element

Array name b holds the starting *address*



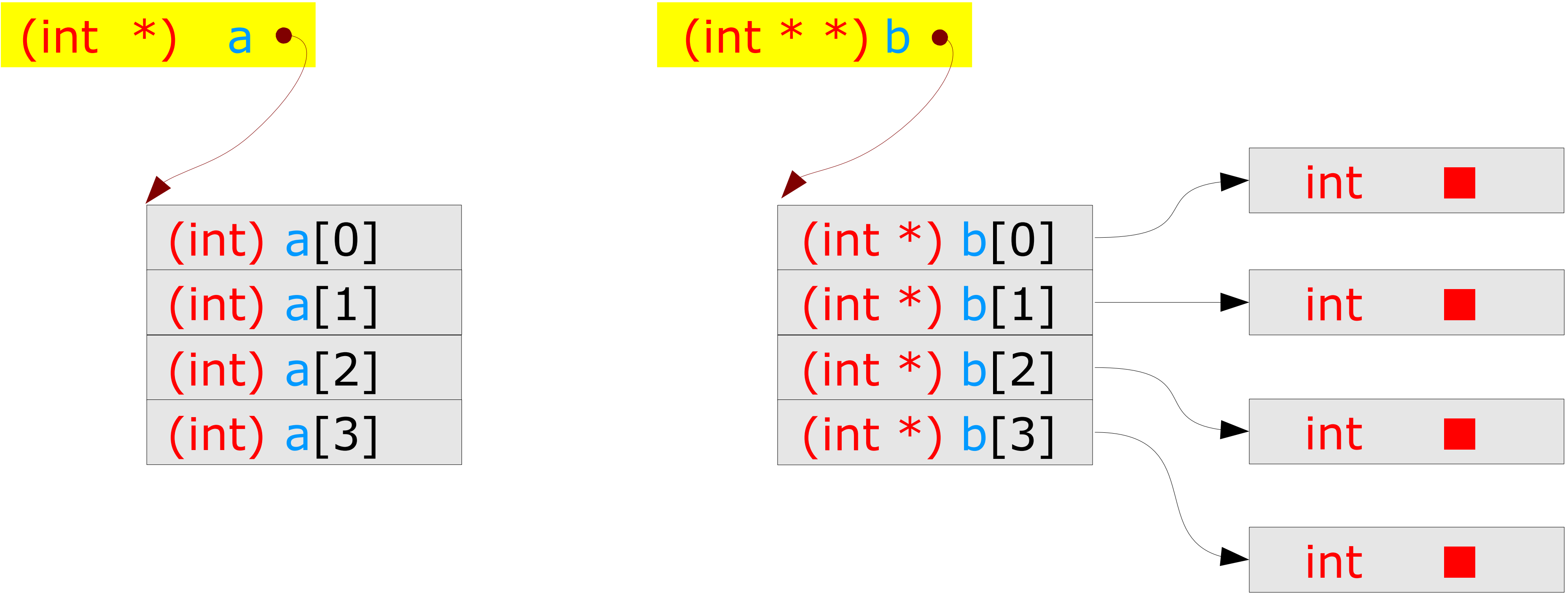
No. of elements = 4

Type of each element

Array of Pointers (2)

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

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



(int *) a

(int) a[0]
(int) a[1]
(int) a[2]
(int) a[3]

(int * *) b

(int *) b[0]
(int *) b[1]
(int *) b[2]
(int *) b[3]

int ■
int ■
int ■
int ■

2-D Array (1)

```
int       a [4];

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

Array name a holds the starting *address*

int   a   [4]

No. of elements = 4

Type of each element

c[0], c[1], c[2], c[3] holds the starting *address*

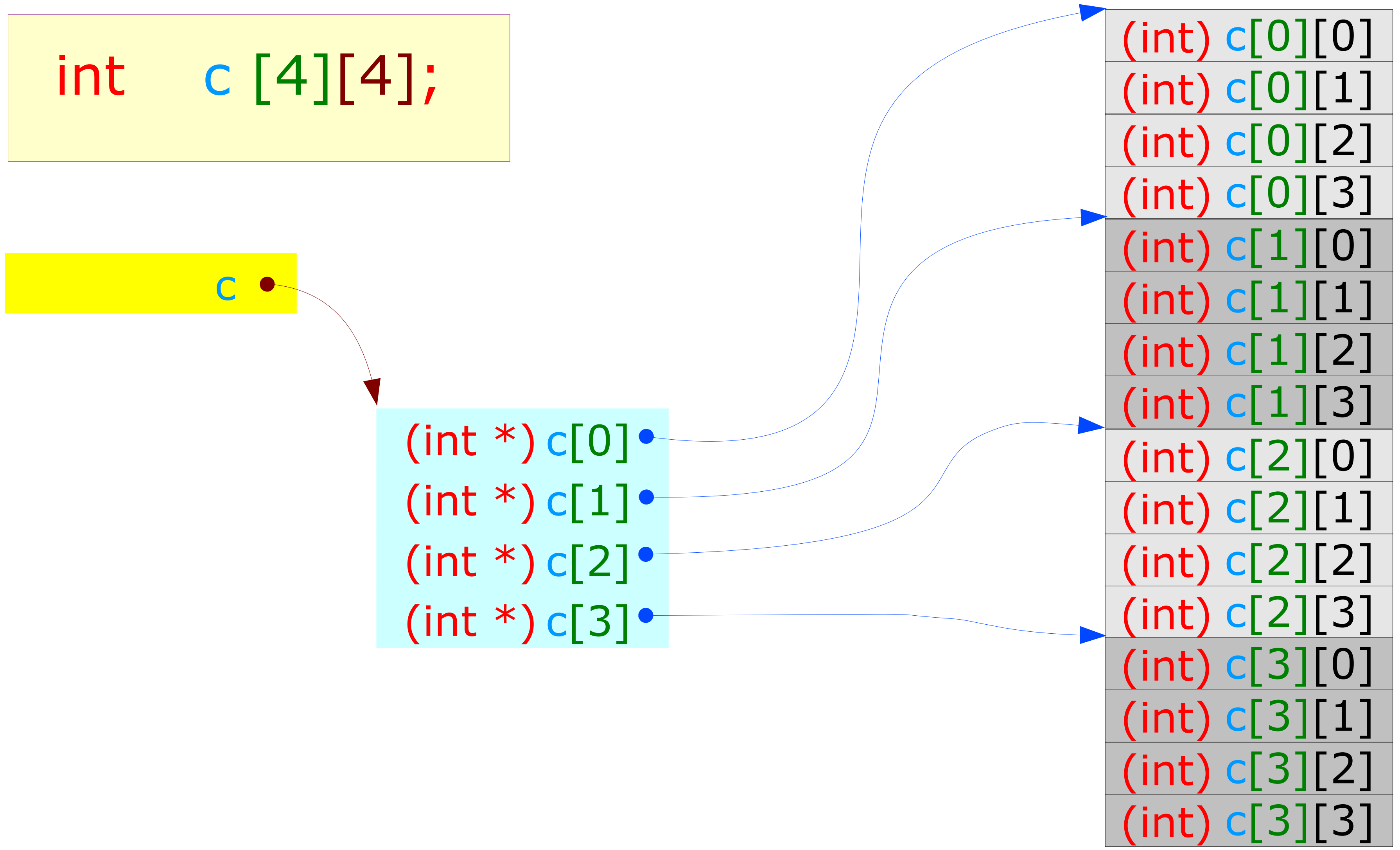
int   c[4]   [4]

No. of elements = 4

Type of each element

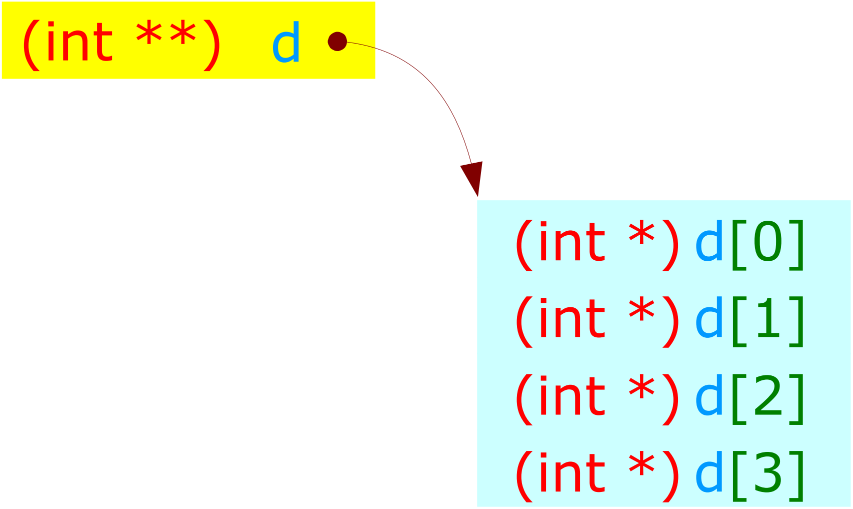
2-D Array (2)

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



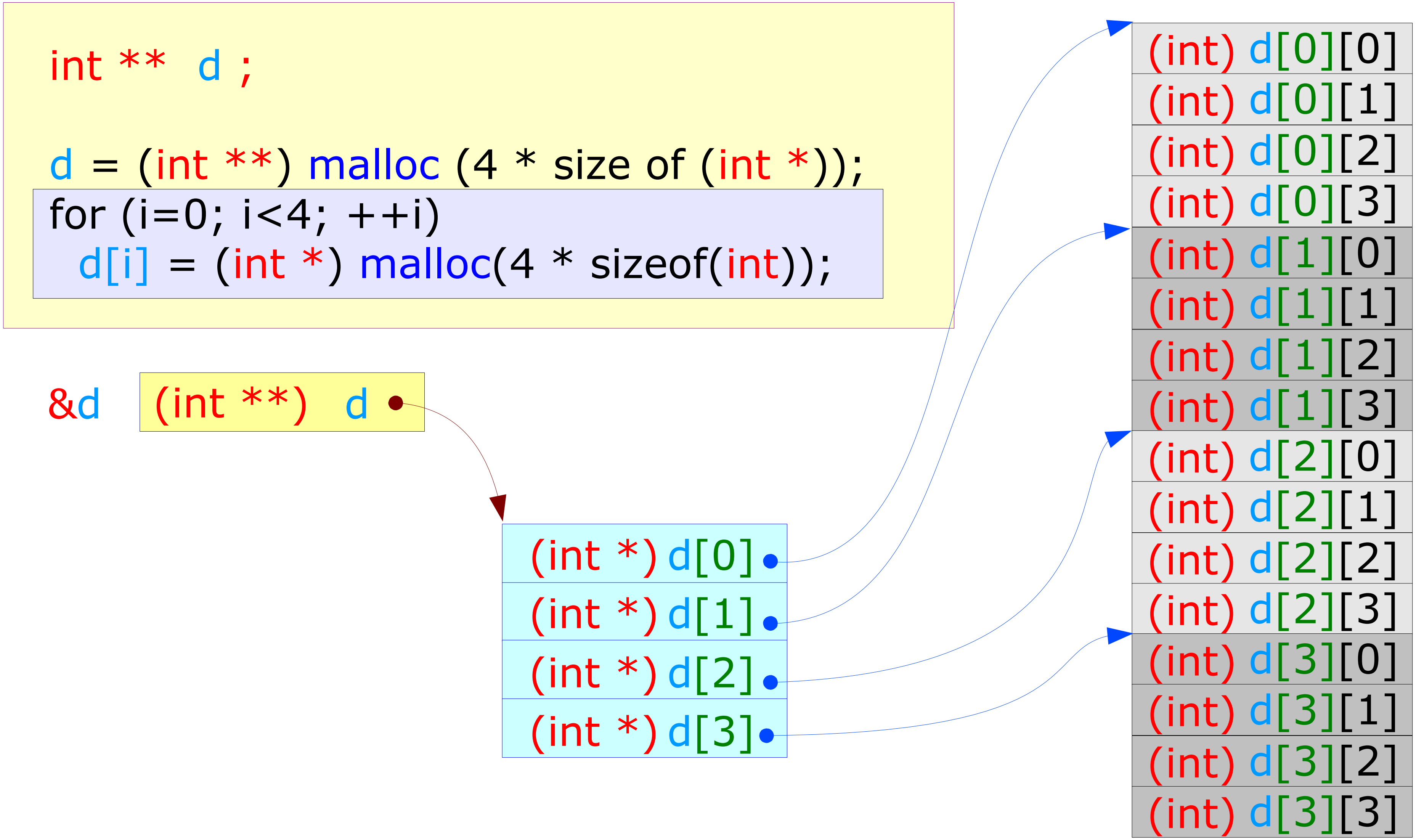
2-D Array Dynamic Memory Allocation (1)

```
int **  d ;

d = (int **) malloc (4 * size of (int *));
for (i=0; i<4; ++i)
  d[i] = (int *) malloc(4 * sizeof(int));
```



2-D Array Dynamic Memory Allocation (2)

```
int **  d ;

d = (int **) malloc (4 * size of (int *));
for (i=0; i<4; ++i)
  d[i] = (int *) malloc(4 * sizeof(int));
```



&d

(int **) d

(int *) d[0]
(int *) d[1]
(int *) d[2]
(int *) d[3]

(int) d[0][0]
(int) d[0][1]
(int) d[0][2]
(int) d[0][3]
(int) d[1][0]
(int) d[1][1]
(int) d[1][2]
(int) d[1][3]
(int) d[2][0]
(int) d[2][1]
(int) d[2][2]
(int) d[2][3]
(int) d[3][0]
(int) d[3][1]
(int) d[3][2]
(int) d[3][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