Advanced Object Oriented Programming

Pointer

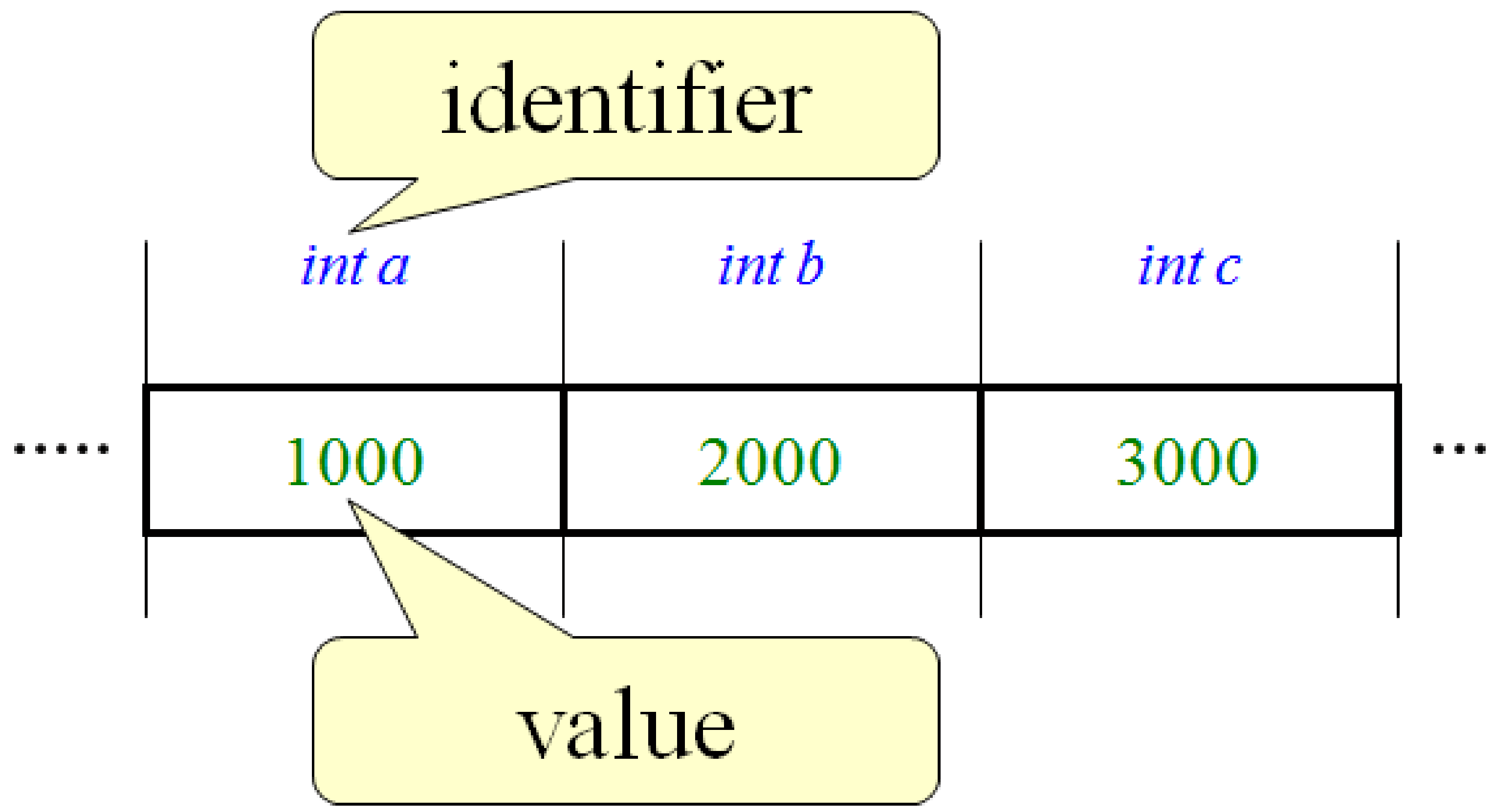
Department of Computer Engineering
Kyung Hee University
drsungwon@khu.ac.kr

Why We Use Pointer in C++?

- Dynamic memory allocation and management
  - you can write programs that can handle unlimited amounts of data on the fly
  - you don't need to know, when write program, how much memory you need

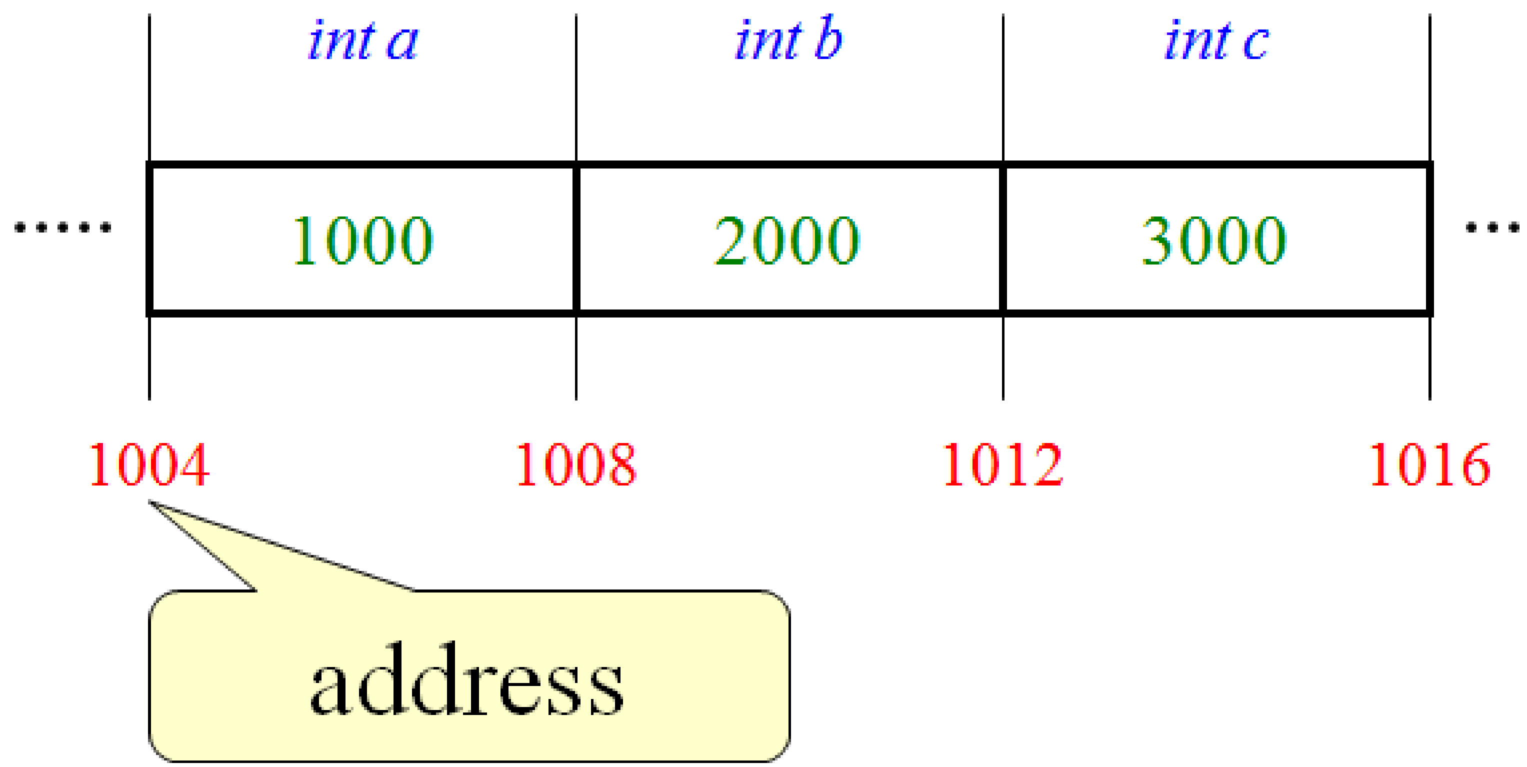
Variable and Memory

```
int main ()
{
    int a;
    int b;
    int c;

    a = 1000;
    b = 2000;
    c = 3000;
}
```



Variable and Memory

```
int main ()
{
    int a;
    int b;
    int c;

    a = 1000;
    b = 2000;
    c = 3000;
}
```



‘&’(Address) Operator

‘&’ is ‘address of’

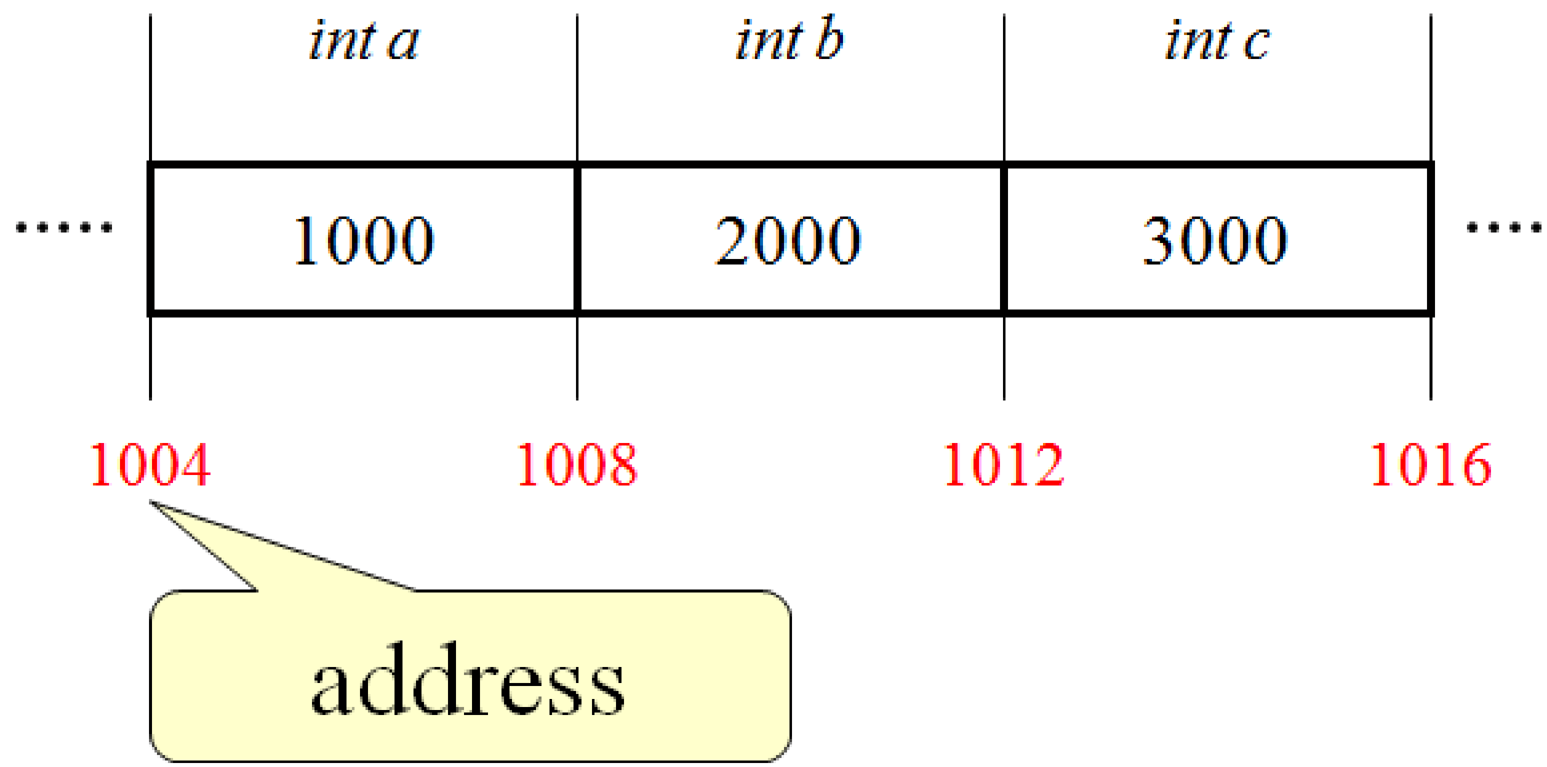
‘&’ Operator

cout << a;

→ 1000

cout << &a;

→ 1004



Pointer Variable

Note:

The address of a variable is the address of the first byte occupied by that variable.

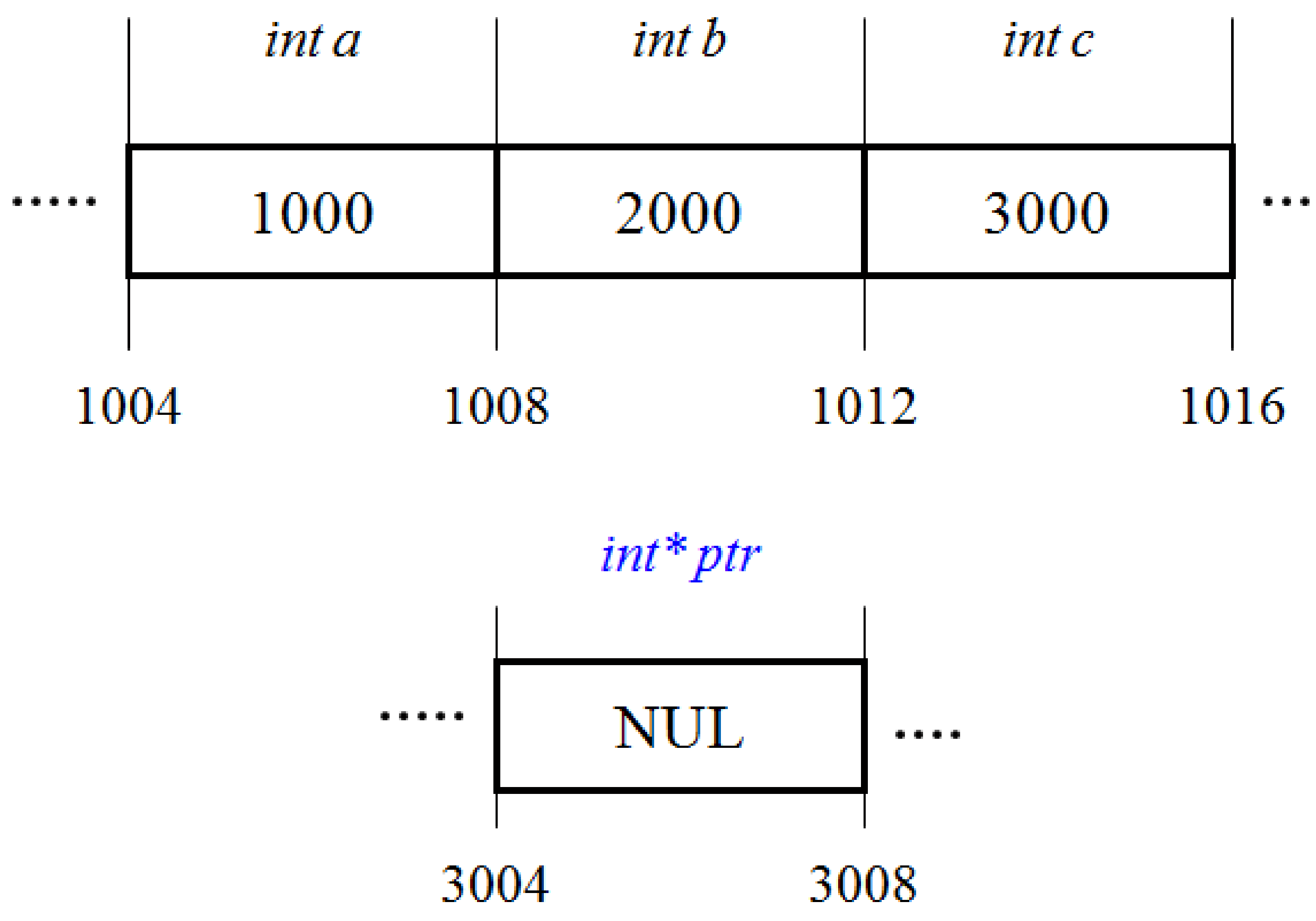
Pointer Variable

'Pointer value' is

- *Strange variable*
- *Does not contain value for int, char, float*
- *Has memory address value*
- *Used to **POINT** other value*

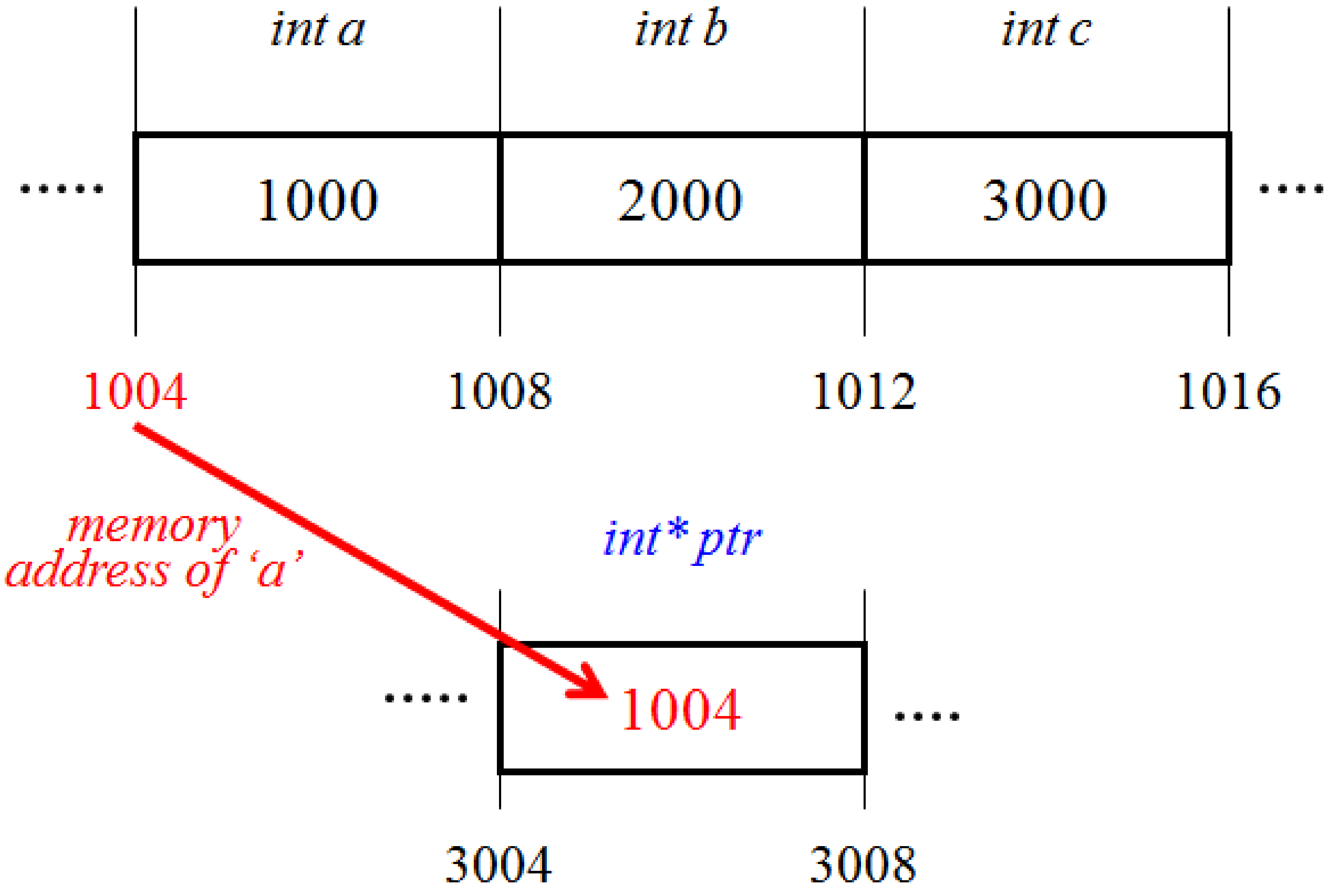
Pointer Variable

```
int main ()
{
    int a;
    int b;
    int c;
    ...
    int* ptr;

    a = 1000;
    b = 2000;
    c = 3000;
}
```

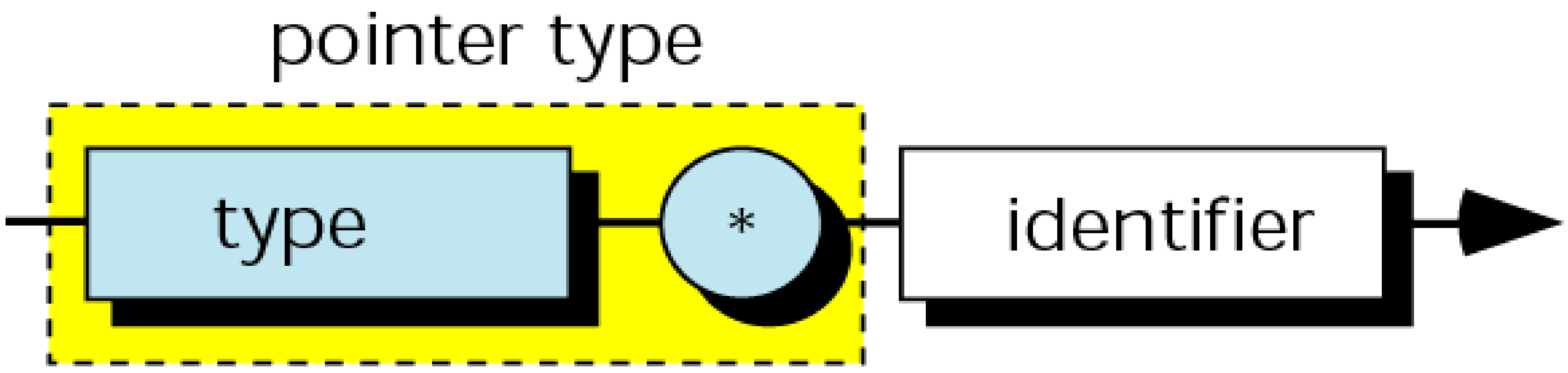


Pointer Variable

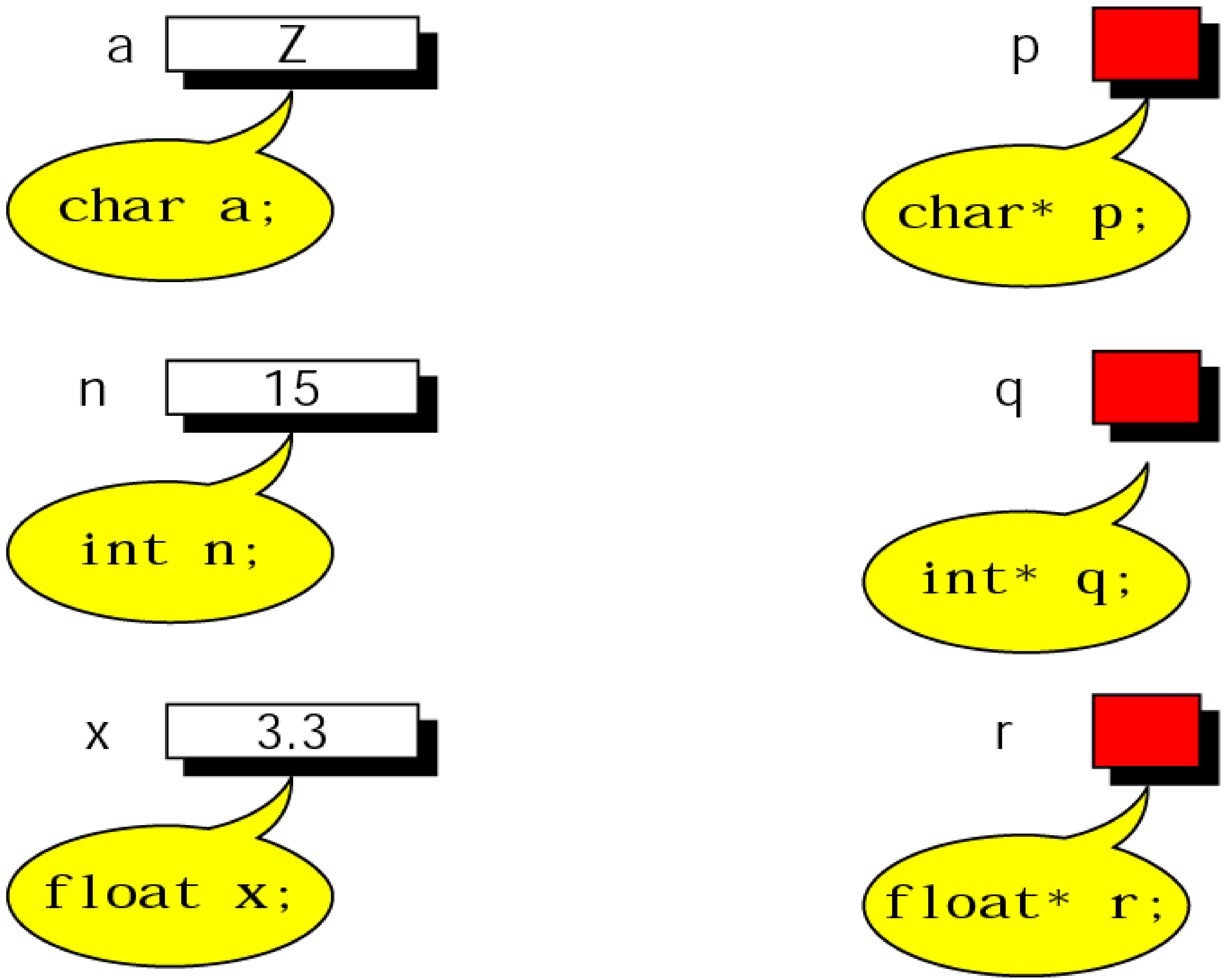
```
int main ()
{
    int a;
    int b;
    int c;
    int* ptr;

    a = 1000;
    b = 2000;
    c = 3000;
    ptr = &a;
}
```



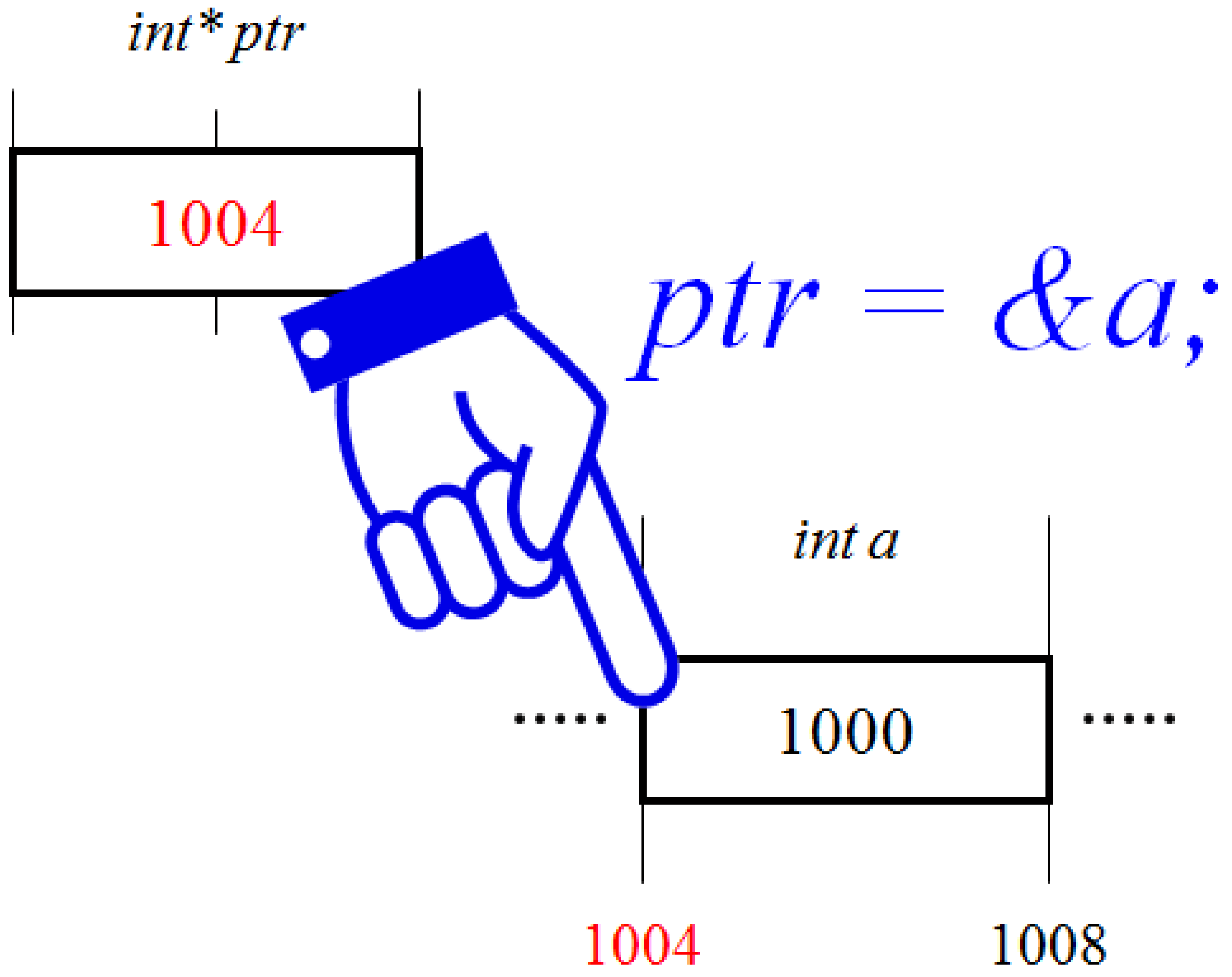
Pointer Variable Declaration



Declaring Pointer Variables



Pointer Variable Initialization



‘*’(Indirection) Operator

‘’ is ‘value of’*

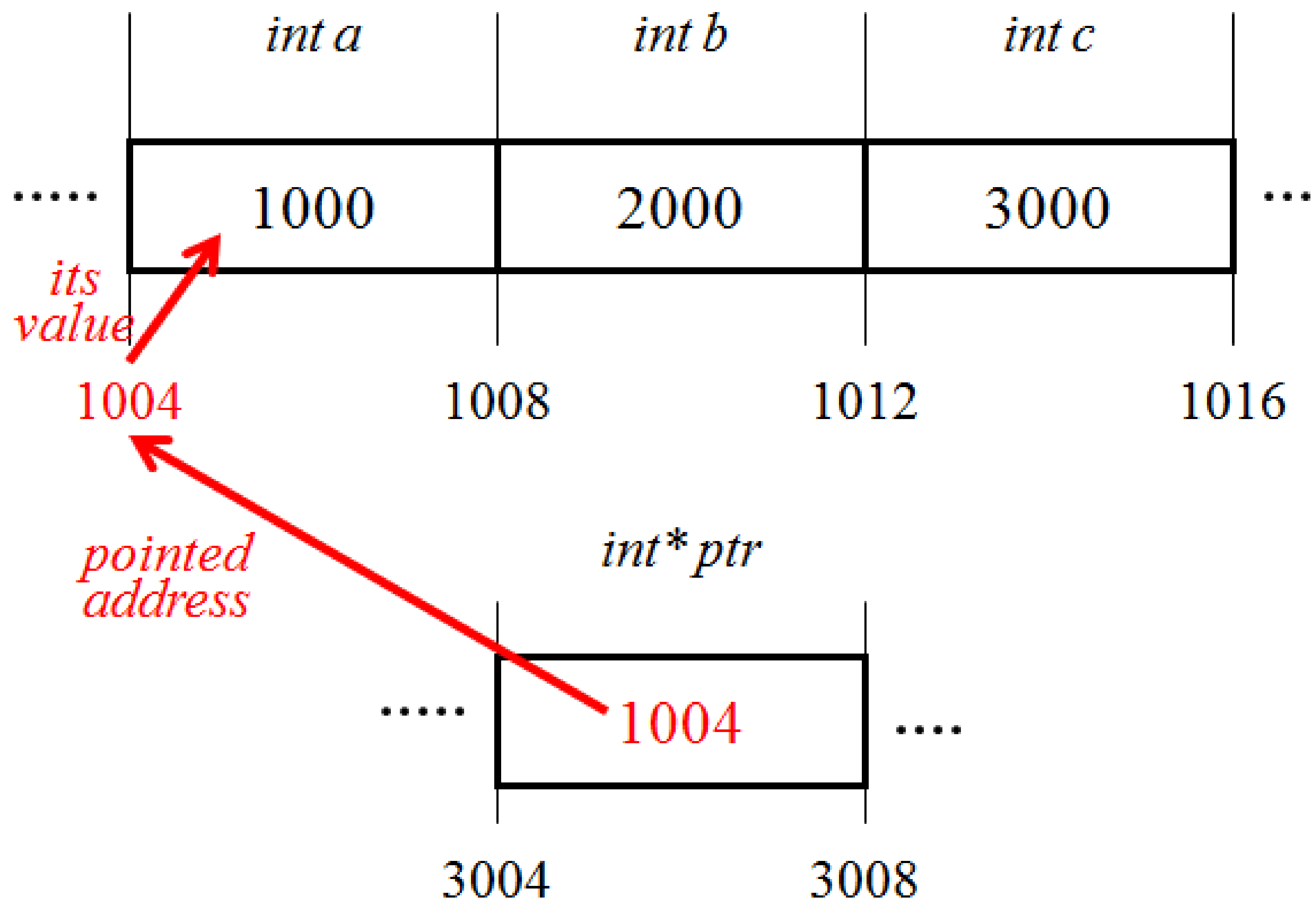
‘*’ Operator

cout << ptr;

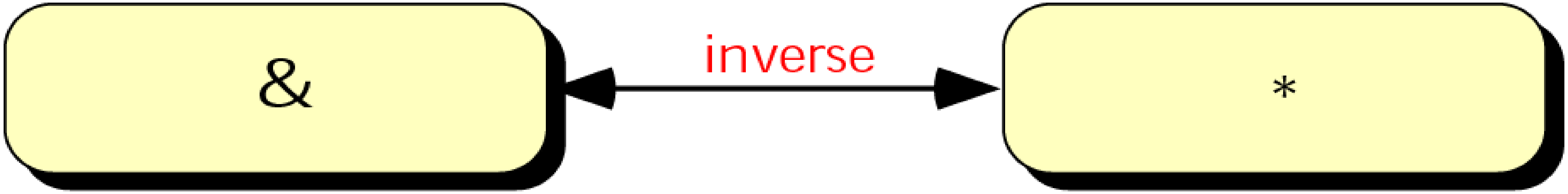
→ 1004

cout << *ptr;

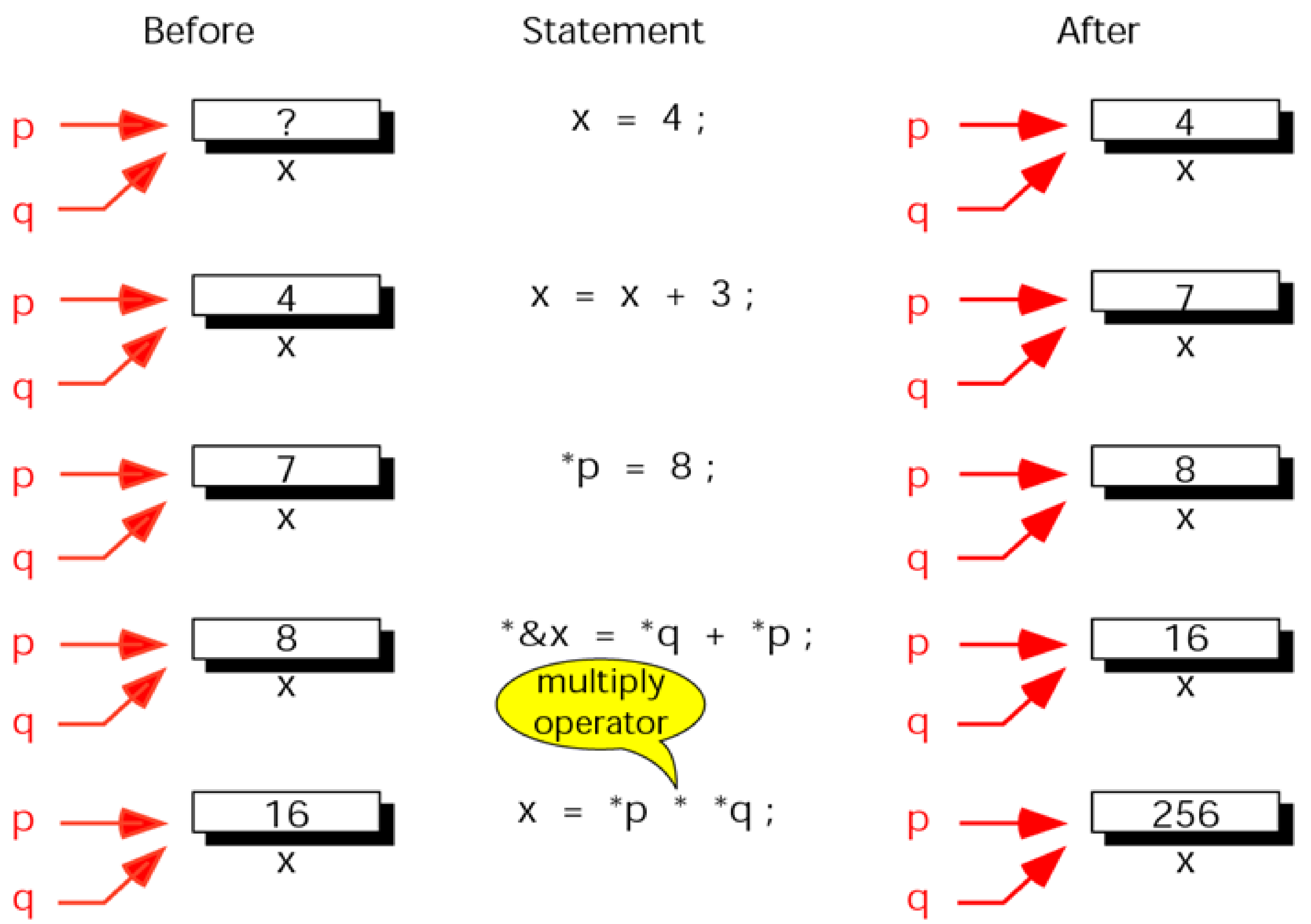
→ 1000



Address and indirection operators



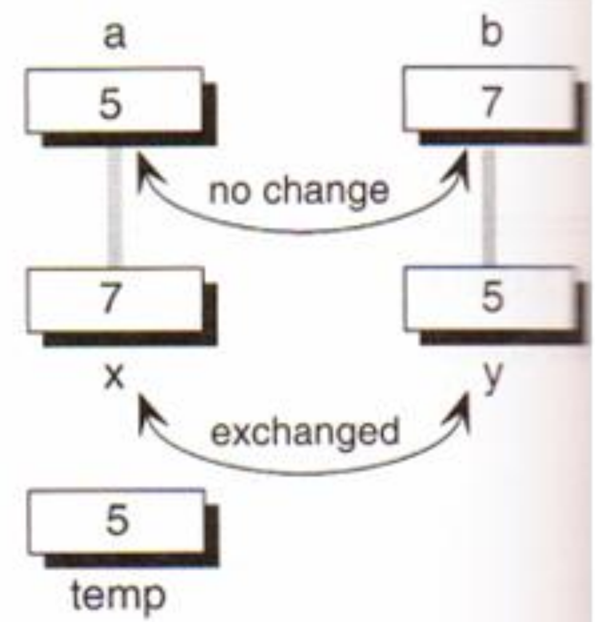
Pointer Examples



Function Call using Pointer

```
int a = 5;
int b = 7;

// Pass by value
exchange (a, b);
```

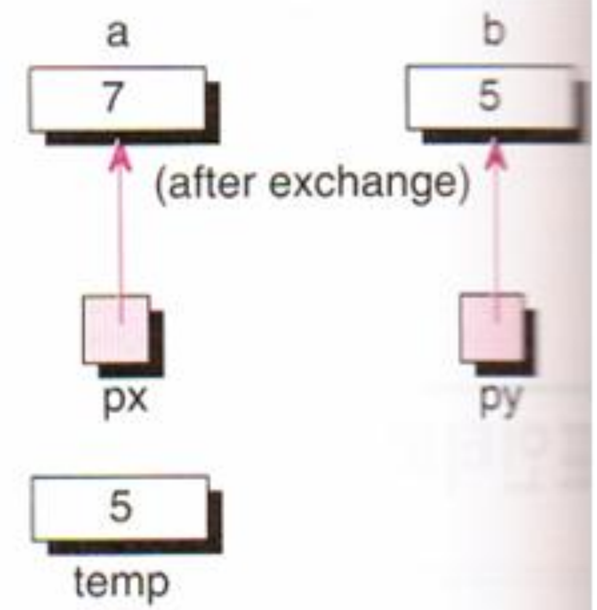
```
void exchange (int x, int y)
{
    int temp = x;
    x        = y;
    y        = temp;
    return;
} // exchange
```



(a) 원본 값들이 바뀌지 않음

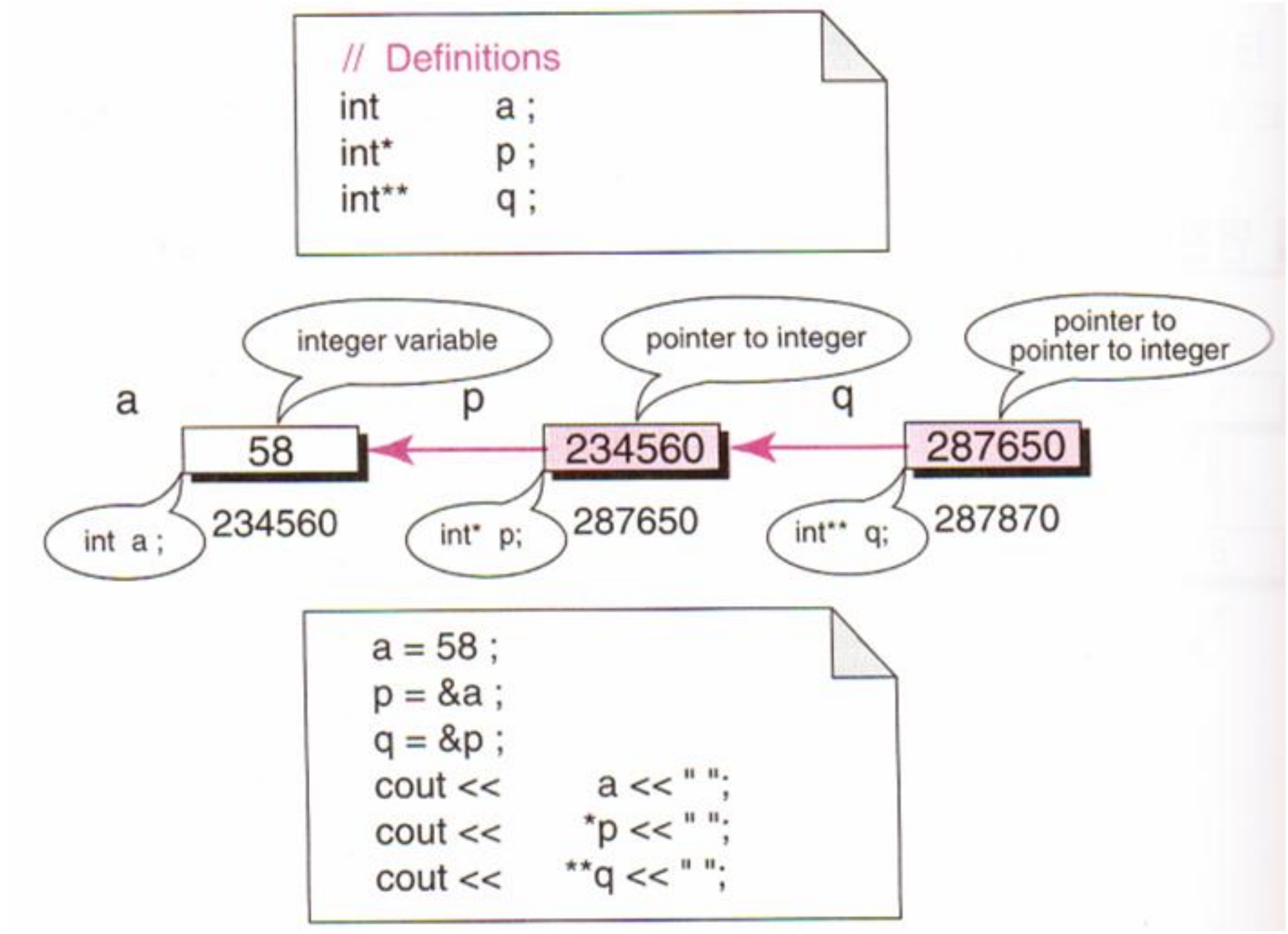
```
int a = 5;
int b = 7;

// Passing pointers
exchange (&a, &b);
```

```
void exchange (int* px, int* py)
{
    int temp   = *px;
    *px        = *py;
    *py        = temp;
    return;
} // exchange
```

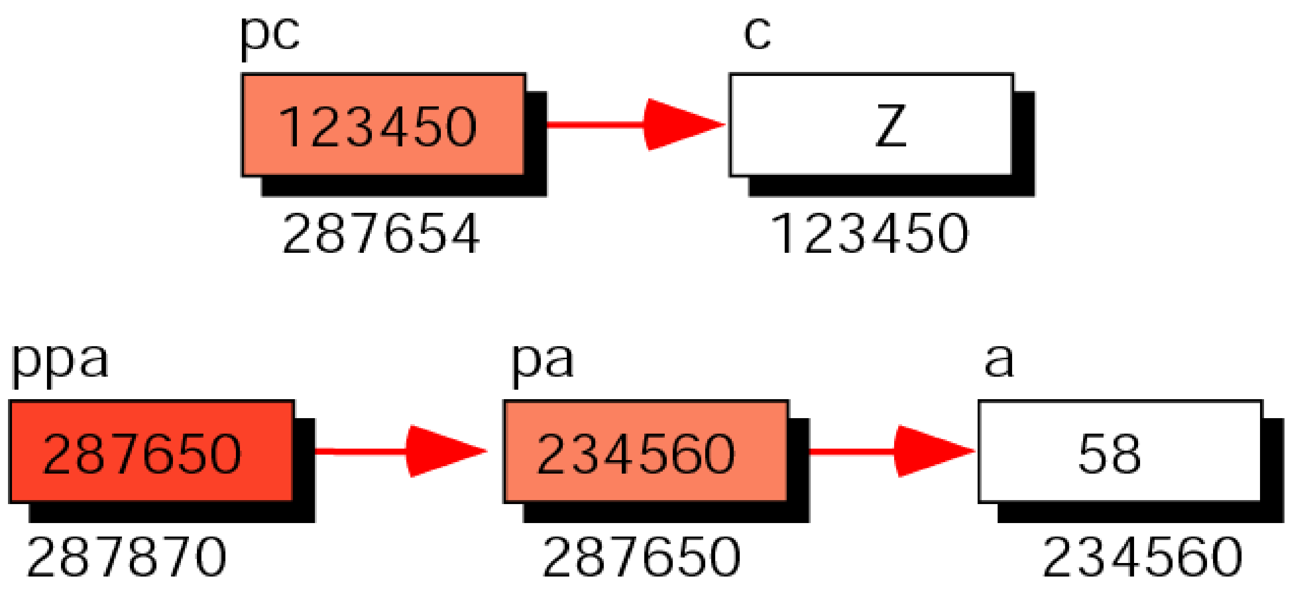


(c) 원본 값이 바뀜

Pointer to Pointer

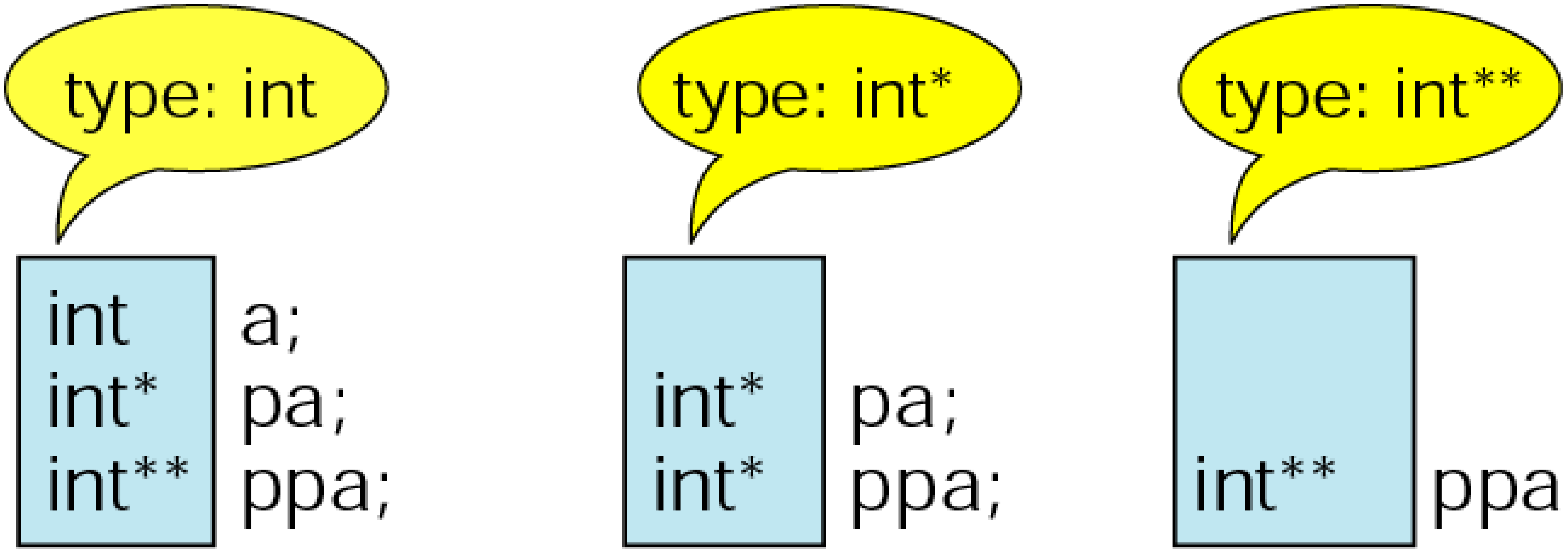


Pointer Compatibility




```
char      c = 'z' ;
char*     pc ;

int       a = 58 ;
int*      pa ;
int**     ppa ;

pc    = &c ;              //  Good and valid
pa    = &a ;              //  Good and valid
ppa   = &pa ;             //  Good and valid

//  The following are invalid and will generate errors
pc    = &a ;              //  Different types
ppa   = &a ;              //  Different levels
```

Pointer Types Must Match




```
a      = 4;
*pa    = 4;
**ppa  = 4;
```

```
pa    = &a;
*ppa  = &a;
```

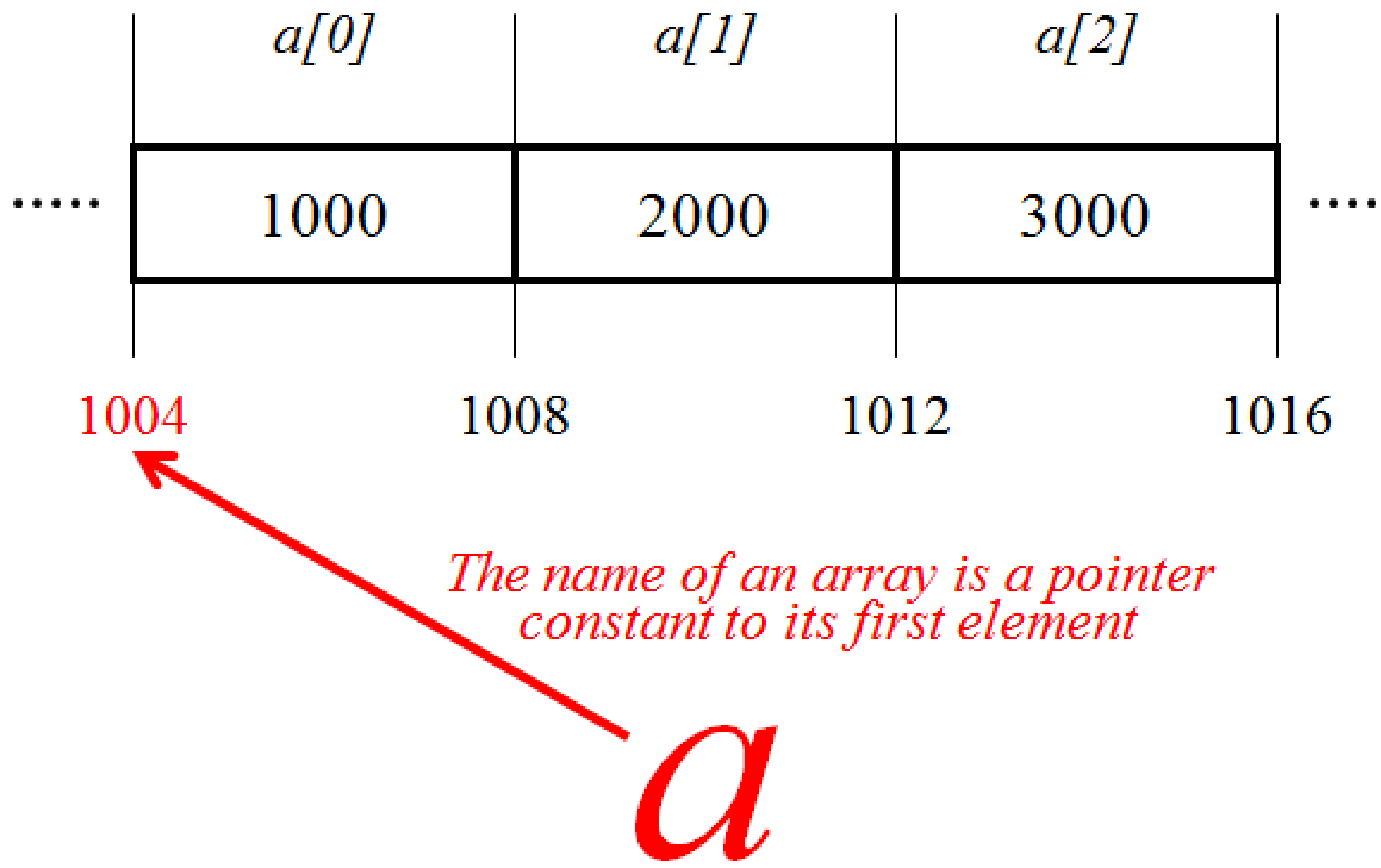
```
ppa  = &pa;
```

Array and Pointer

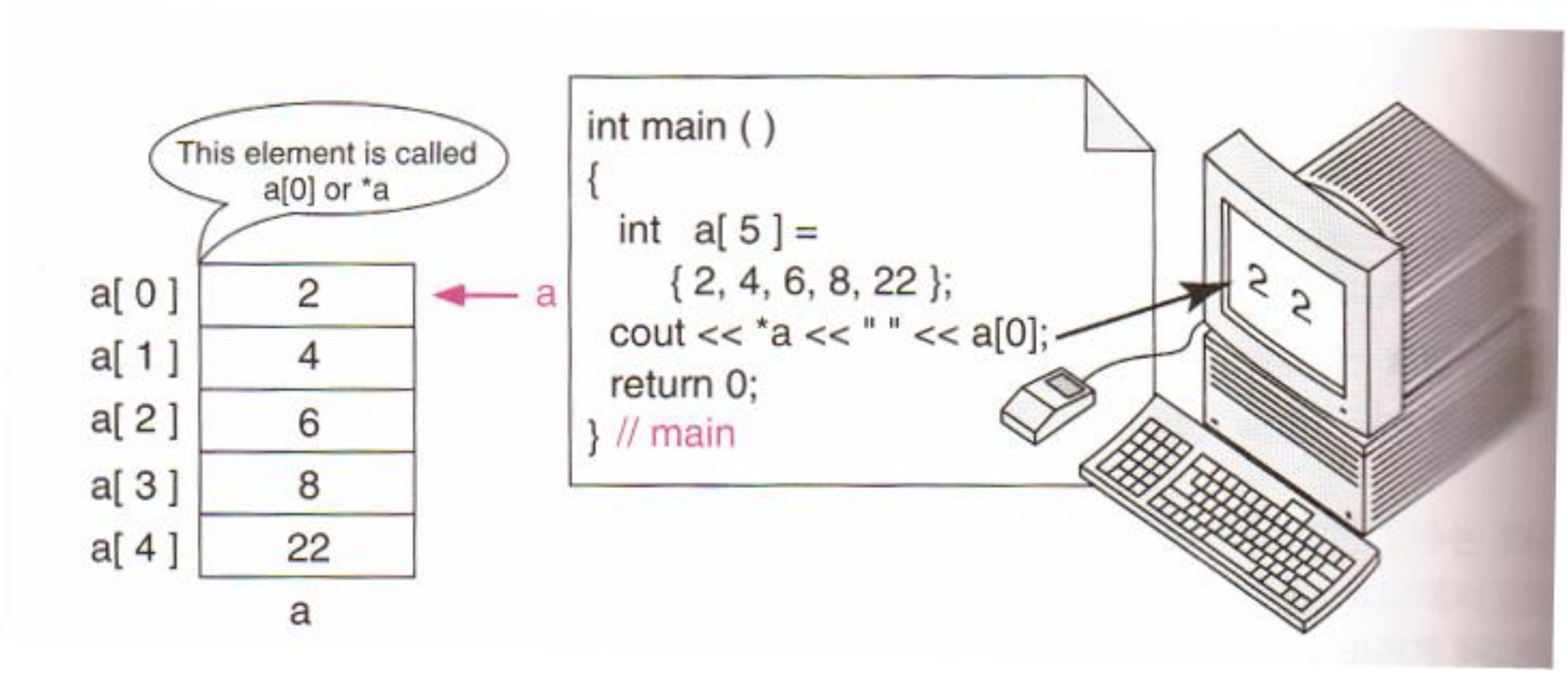
The name of an array is a pointer constant to its first element

Array and Pointer

```
int main ()
{
    int a[3];

    cout << &a[0];
//  → 1004

    cout << a;
//  → 1004
}
```



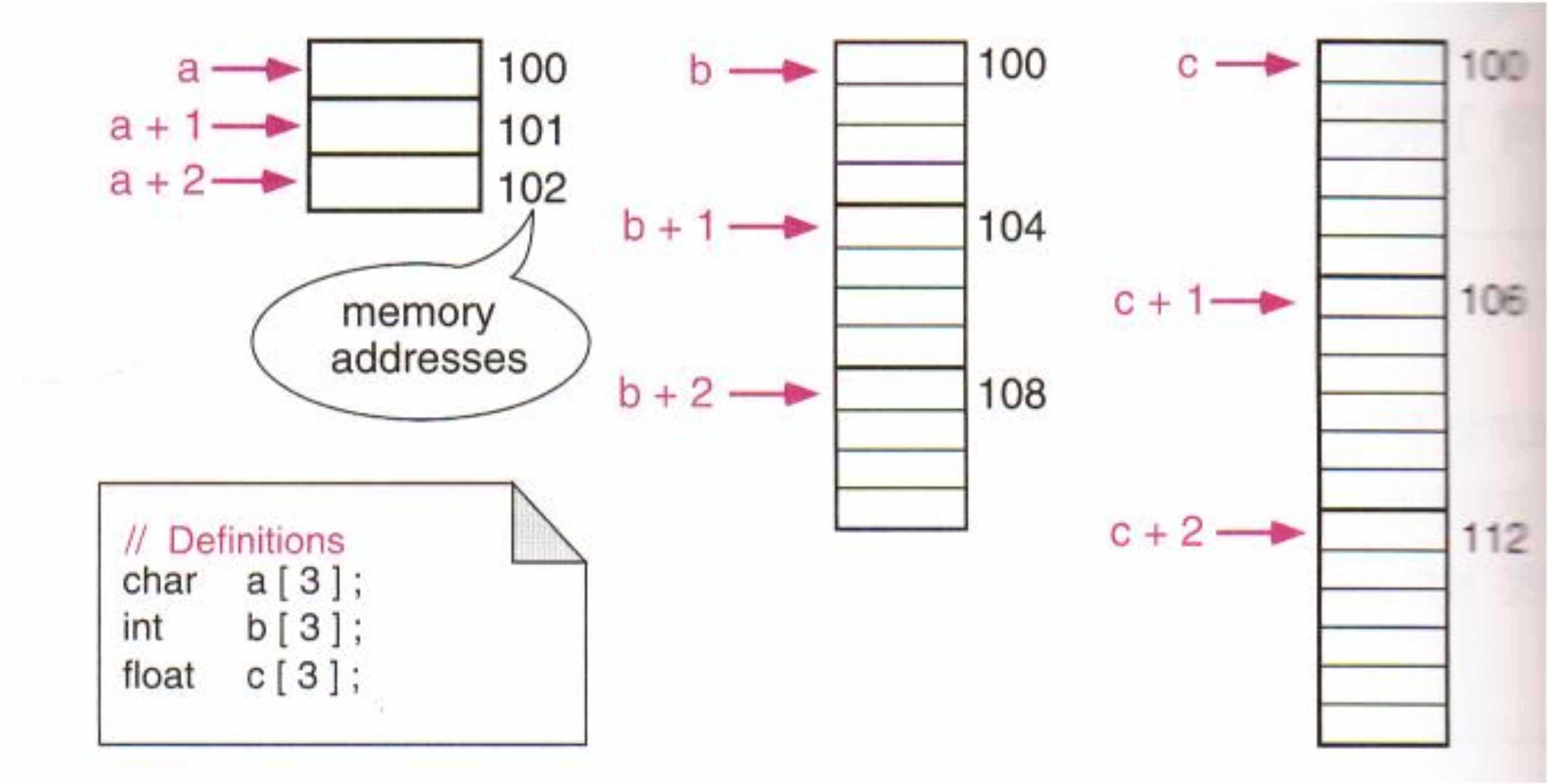
Array and Pointer



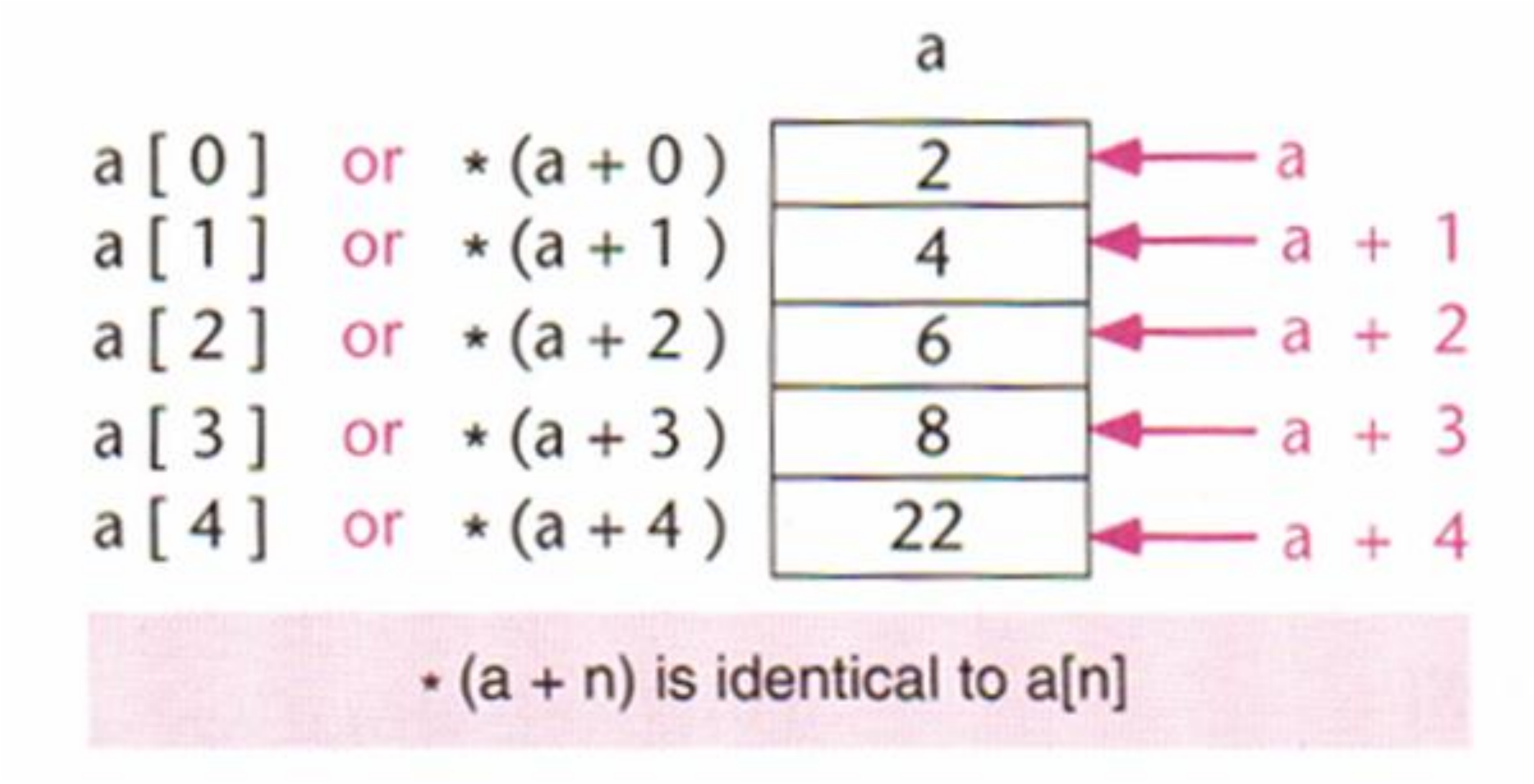
Pointer Arithmetic

Pointer +1/-1 means
"increase/decrease pointer value according to 'sizeof (pointer value type declaration)'

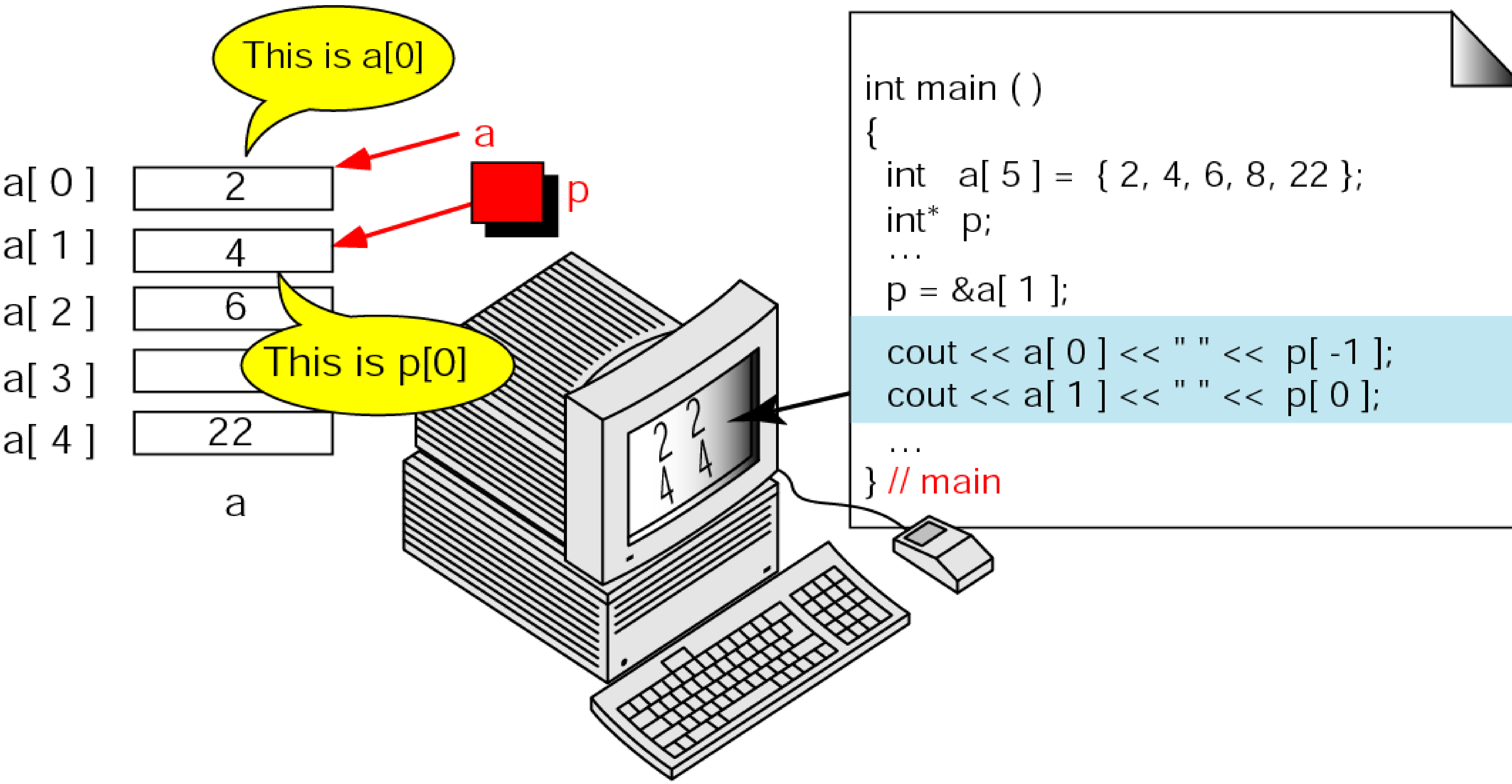
Pointer Arithmetic



Pointer Arithmetic and Array

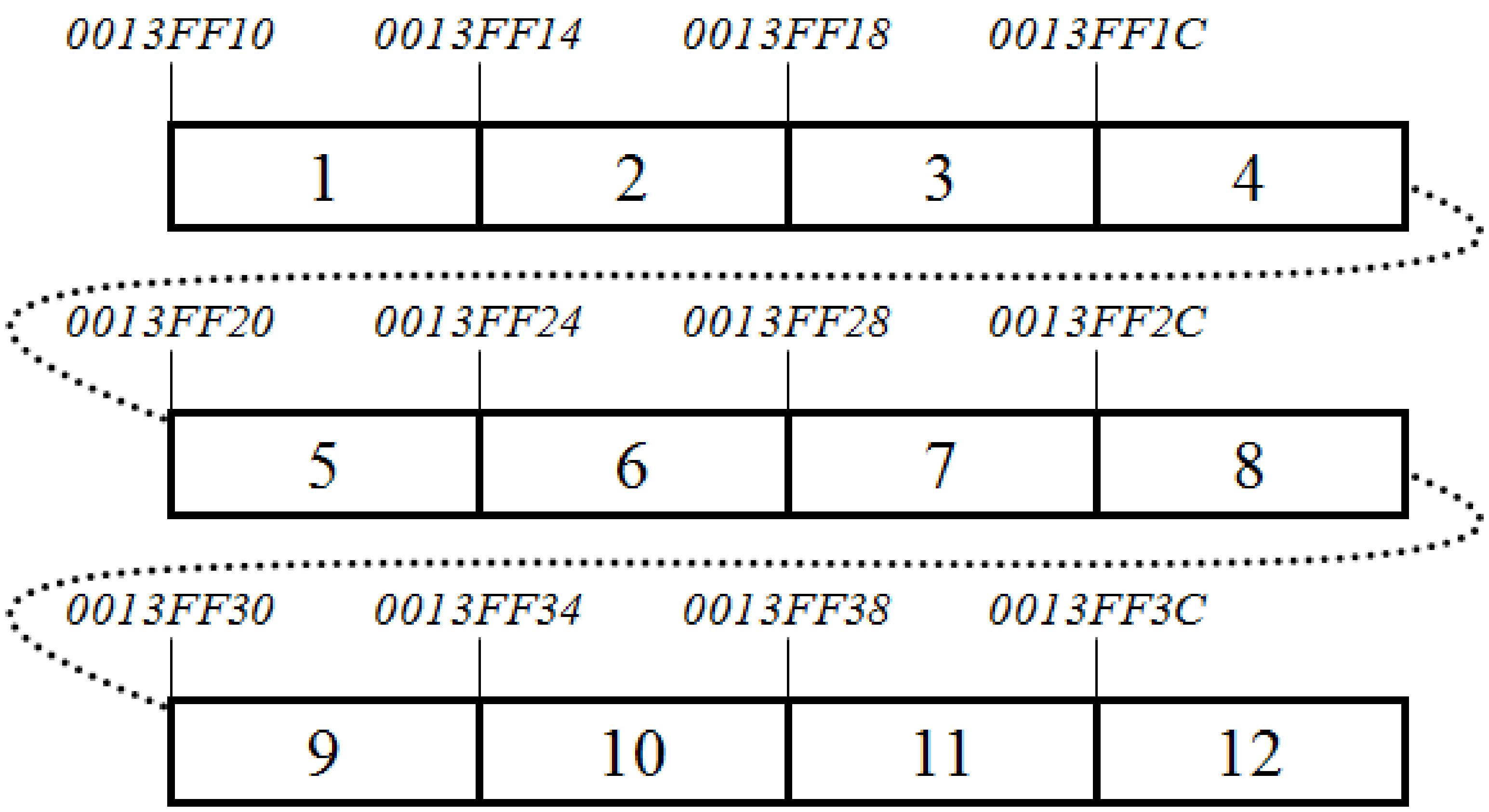


Pointer Arithmetic and Array – Negative Index Value



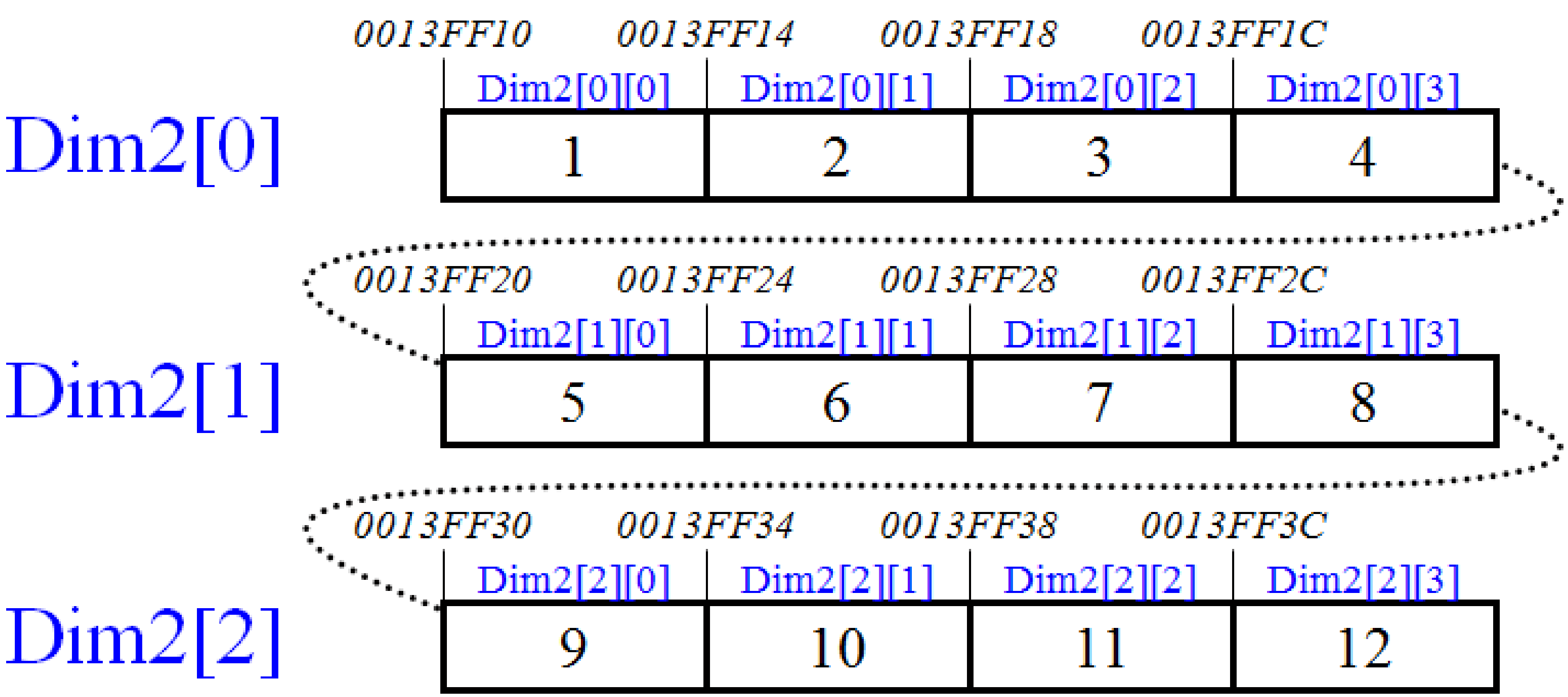
Case Study

int Dim2[3][4]

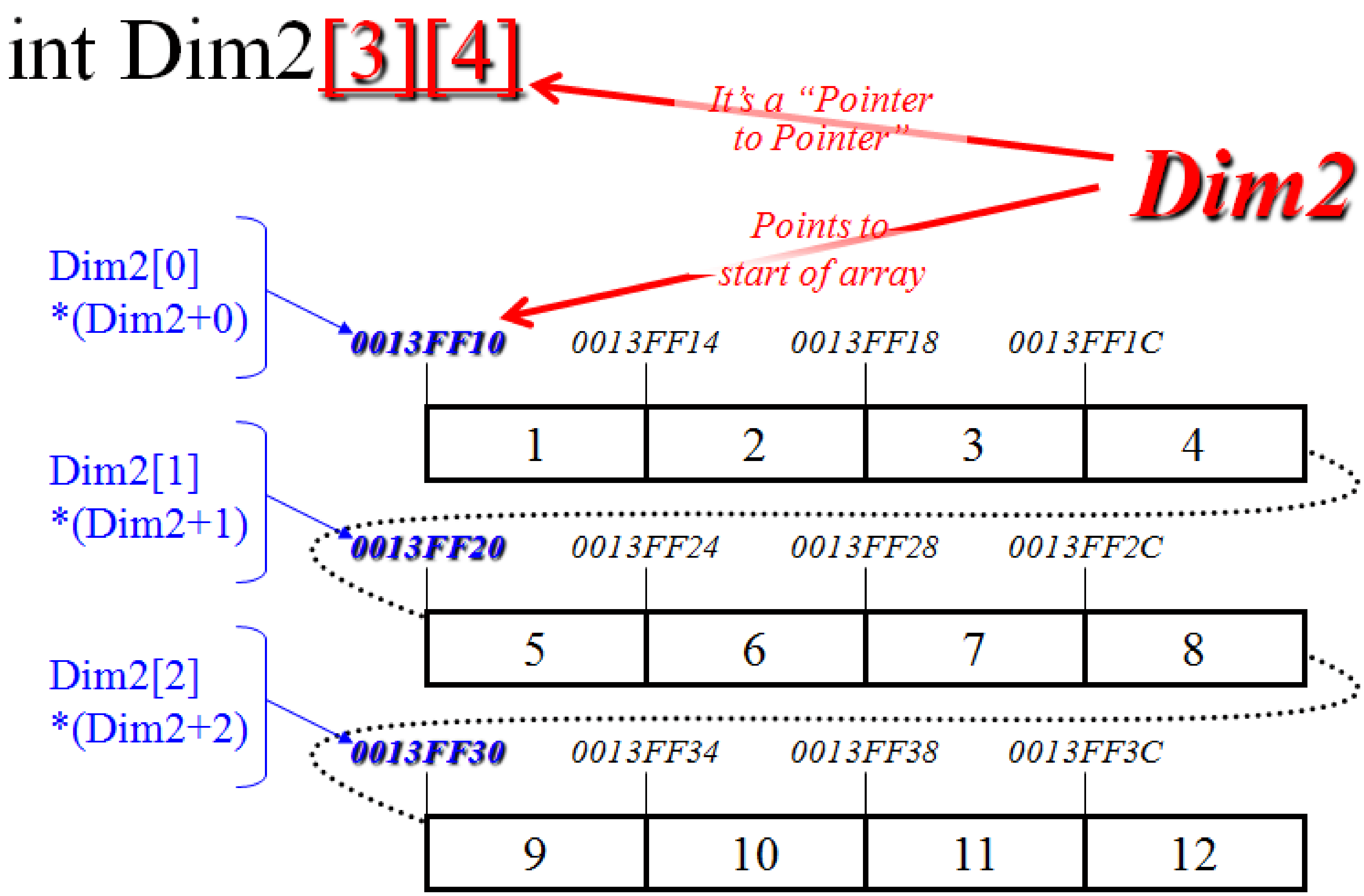


Case Study

int Dim2[3][4]



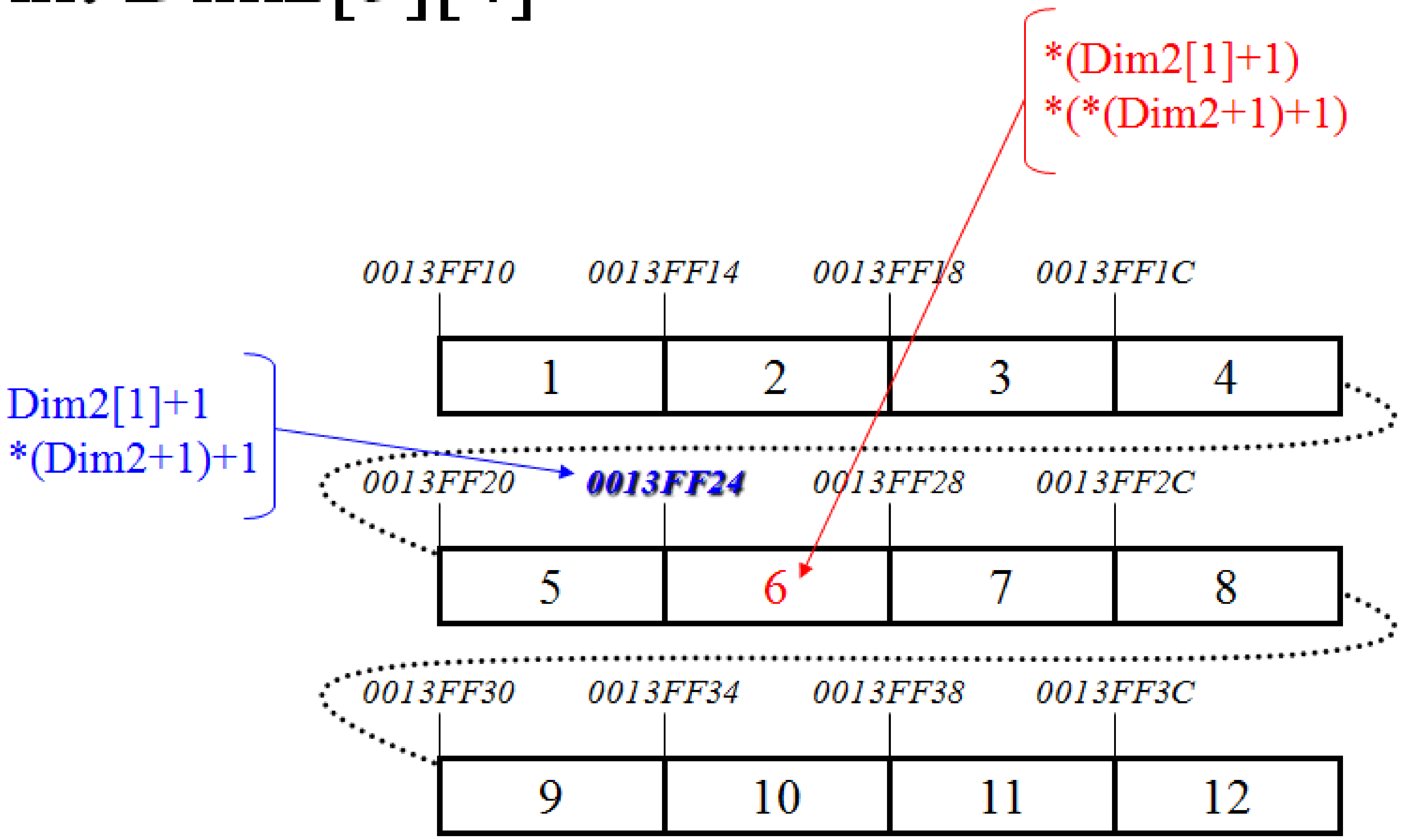
Case Study



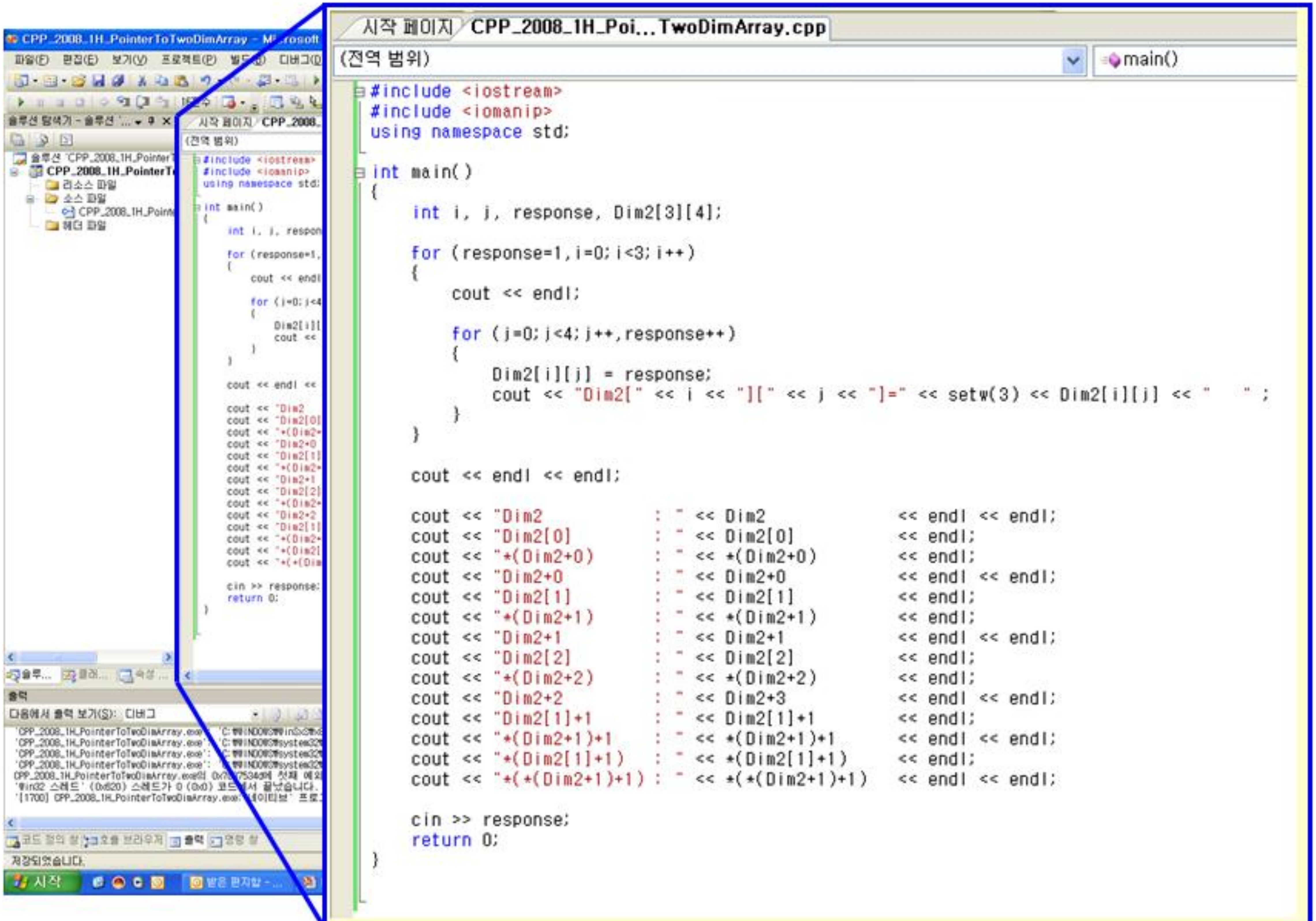
int Dim2[3][4]
It's a "Pointer
to Pointer"
Dim2
Points to
start of array
Dim2[0]
*(Dim2+0)
Dim2[1]
*(Dim2+1)
Dim2[2]
*(Dim2+2)
0013FF10
0013FF14
0013FF18
0013FF1C
1
2
3
4
0013FF20
0013FF24
0013FF28
0013FF2C
5
6
7
8
0013FF30
0013FF34
0013FF38
0013FF3C
9
10
11
12

Case Study

int Dim2[3][4]

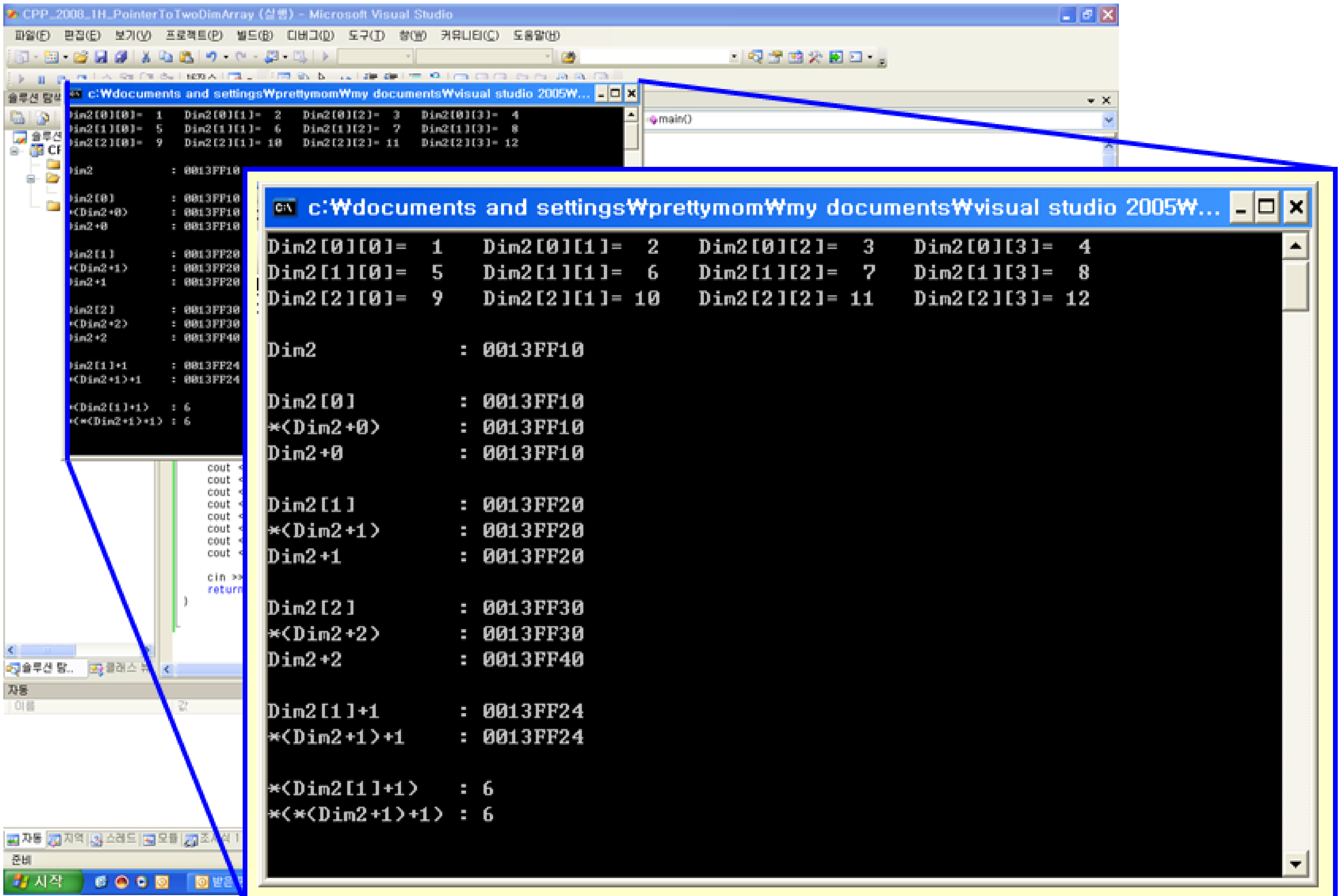


Case Study



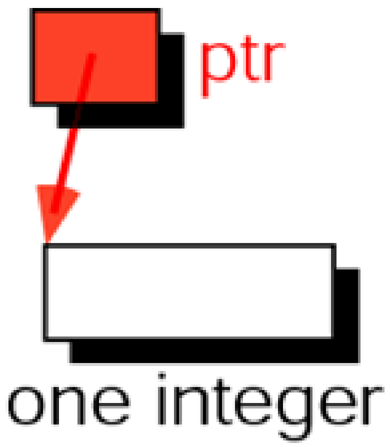
시작 페이지
CPP_2008_1H_Poi...TwoDimArray.cpp
(전역 범위)
main()
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
int i, j, response, Dim2[3][4];
for (response=1,i=0;i<3;i++)
{
cout << endl;
for (j=0;j<4;j++,response++)
{
Dim2[i][j] = response;
cout << "Dim2[" << i << "][" << j << "]=" << setw(3) << Dim2[i][j] << "   " ;
}
}
cout << endl << endl;
cout << "Dim2 : " << Dim2 << endl << endl;
cout << "Dim2[0] : " << Dim2[0] << endl;
cout << "*(Dim2+0) : " << *(Dim2+0) << endl;
cout << "Dim2+0 : " << Dim2+0 << endl << endl;
cout << "Dim2[1] : " << Dim2[1] << endl;
cout << "*(Dim2+1) : " << *(Dim2+1) << endl;
cout << "Dim2+1 : " << Dim2+1 << endl << endl;
cout << "Dim2[2] : " << Dim2[2] << endl;
cout << "*(Dim2+2) : " << *(Dim2+2) << endl;
cout << "Dim2+2 : " << Dim2+3 << endl << endl;
cout << "Dim2[1]+1 : " << Dim2[1]+1 << endl;
cout << "*(Dim2+1)+1 : " << *(Dim2+1)+1 << endl << endl;
cout << "*(Dim2[1]+1) : " << *(Dim2[1]+1) << endl;
cout << "*(*(Dim2+1)+1) : " << *(*(Dim2+1)+1) << endl << endl;
cin >> response;
return 0;
}

Case Study

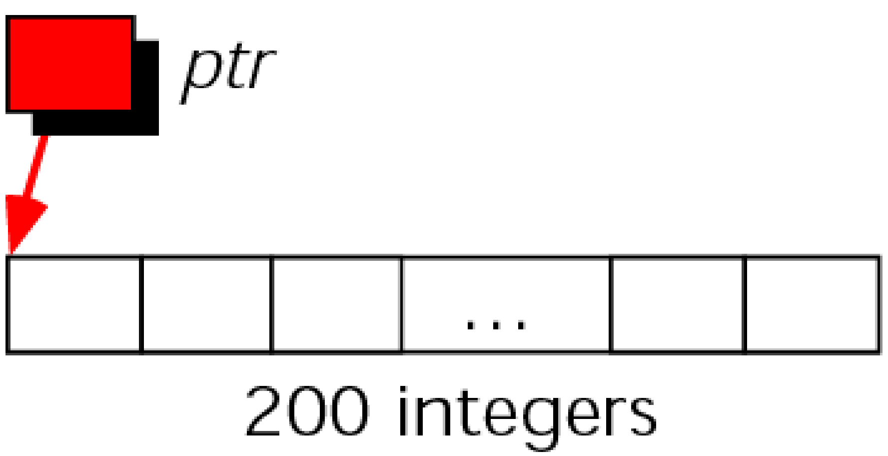


CPP_2008_1H_PointerToTwoDimArray (실행) - Microsoft Visual Studio
c:₩documents and settings₩prettymom₩my documents₩visual studio 2005₩...
Dim2[0][0]= 1 Dim2[0][1]= 2 Dim2[0][2]= 3 Dim2[0][3]= 4
Dim2[1][0]= 5 Dim2[1][1]= 6 Dim2[1][2]= 7 Dim2[1][3]= 8
Dim2[2][0]= 9 Dim2[2][1]= 10 Dim2[2][2]= 11 Dim2[2][3]= 12
Dim2 : 0013FF10
Dim2[0] : 0013FF10
*(Dim2+0) : 0013FF10
Dim2+0 : 0013FF10
Dim2[1] : 0013FF20
*(Dim2+1) : 0013FF20
Dim2+1 : 0013FF20
Dim2[2] : 0013FF30
*(Dim2+2) : 0013FF30
Dim2+2 : 0013FF40
Dim2[1]+1 : 0013FF24
*(Dim2+1)+1 : 0013FF24
*(Dim2[1]+1) : 6
((Dim2+1)+1) : 6

‘new’ operator

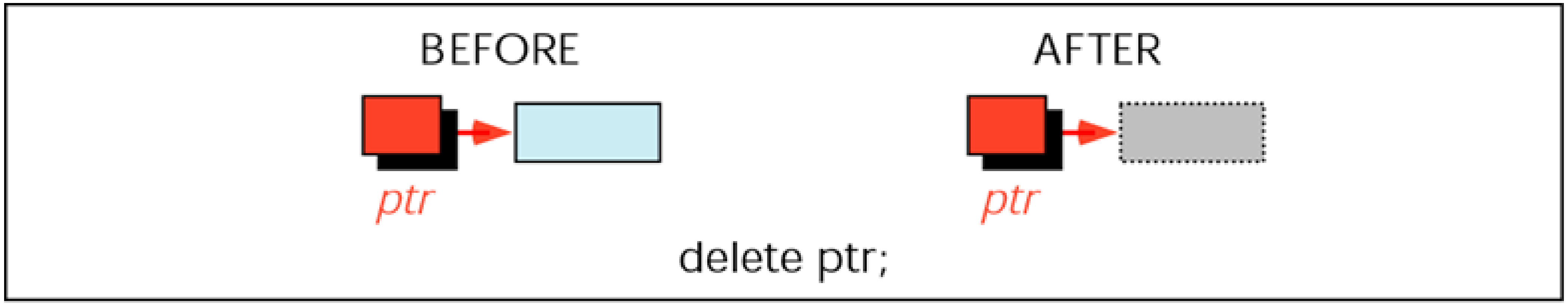


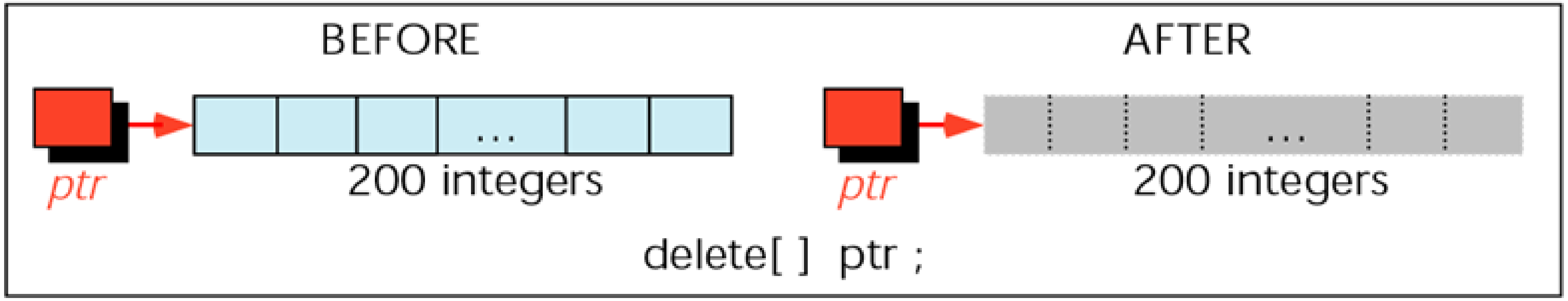

```
int* ptr = new int;
```



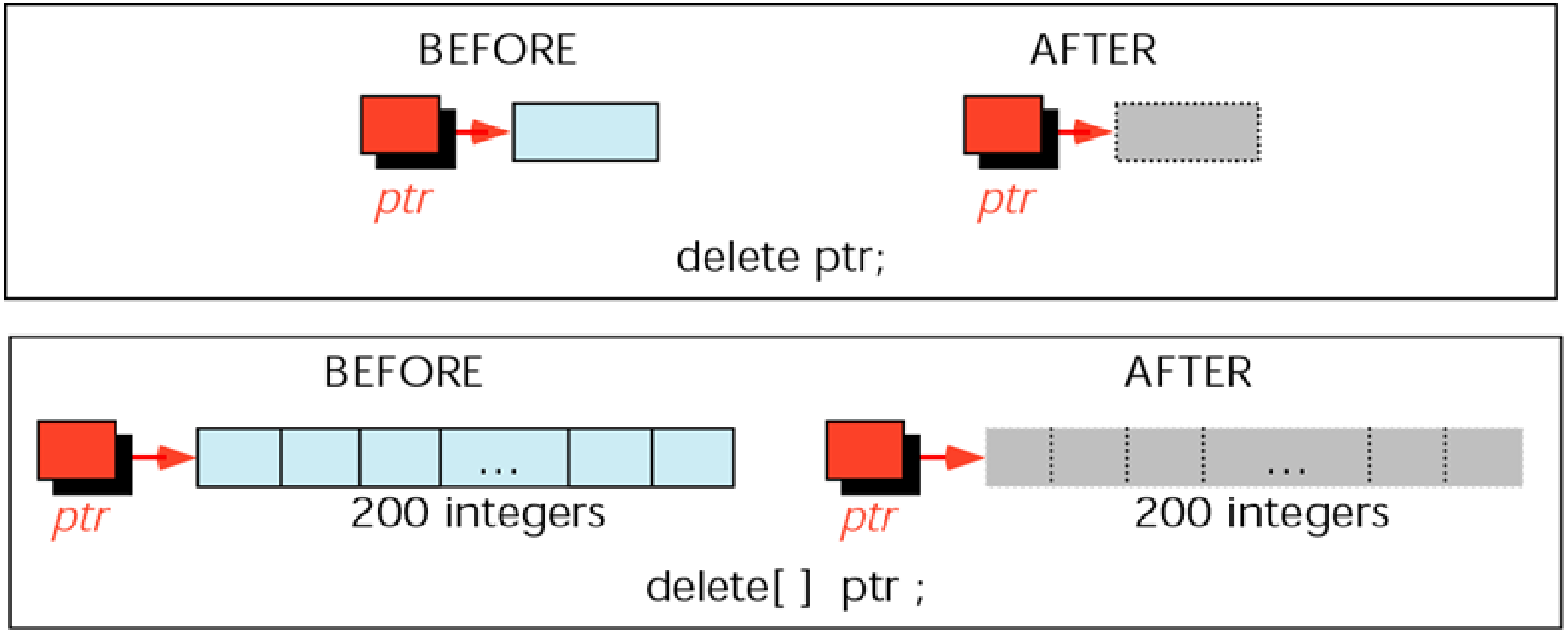

```
int* ptr = new int[200];
```

‘delete’ operator

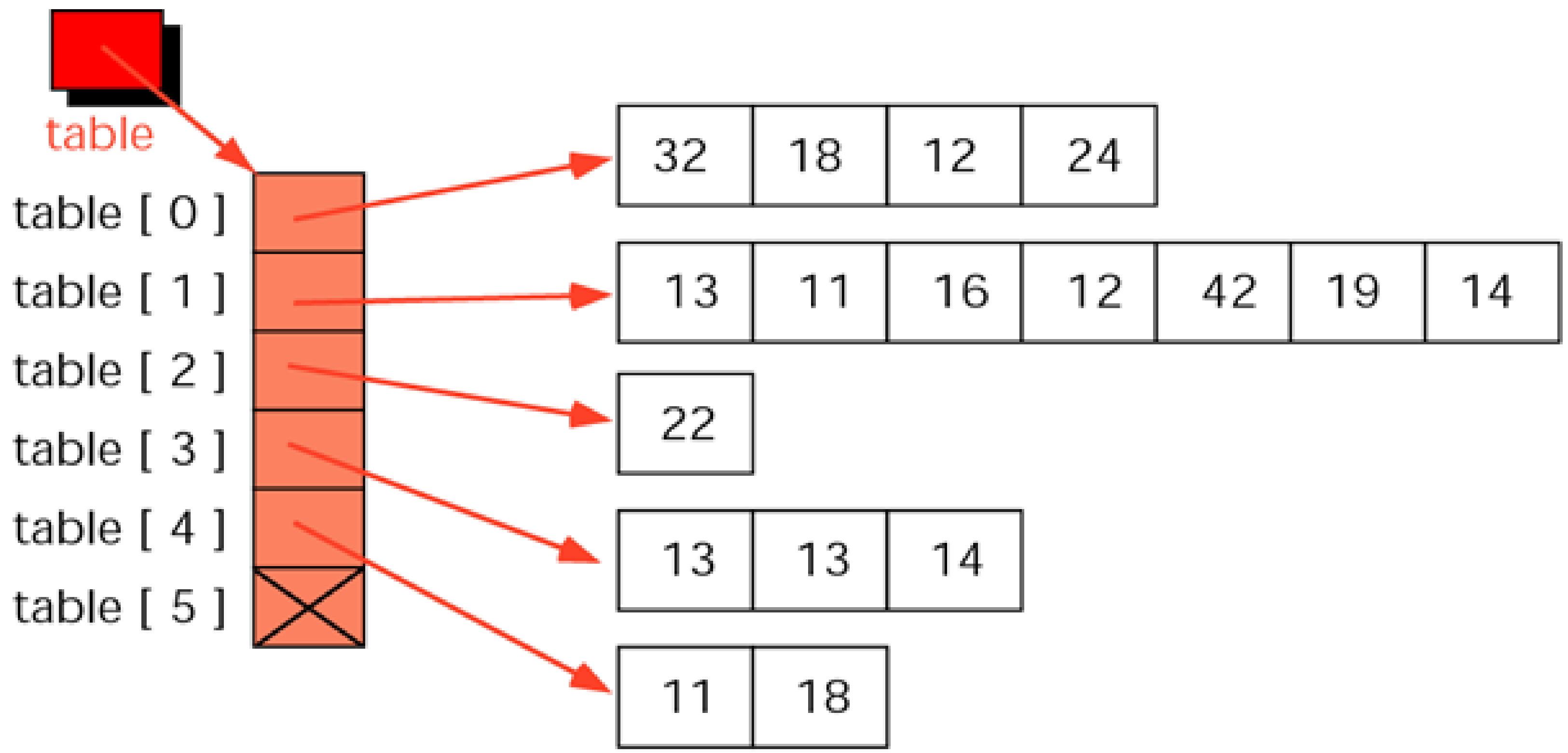




‘delete’ operator



Array Pointer




```
int** table;
 ...
table = new int* [rowNum + 1] ;
 ...
table[0] = new  int[4];
table[1] = new  int[7];
table[2] = new  int[1];
table[3] = new  int[3];
table[4] = new  int[2];
table[5] = NULL ;
```

Array vs Pointer

	Array	Dynamic Memory Allocation using Pointer
Declaration	int arrayVar[50];	int* ptrVar;
Memory Allocation	Not required (automatically at program execution)	ptrVar = new int[50];
Memory Release	Impossible (memory reserved until termination)	delete ptrVar;
Too much data case	Impossible to increase memory size	Easy (allocate more memory)
Too small data case	Impossible to decrease Memory size	Easy (release and maintain small memory)
Pros	Easy to programming using just an INDEX	Optimized memory usage
Cons	Fixed memory space (Big program), Weak for unexpected memory request	Complex to manage pointers (side effect expected)

Questions?