C语言指针与内存操作原理探秘

荣文戈

w.rong@buaa.edu.cn

北京航空航天大学计算机学院

2021.09.15

先提几个问题

1、int a;
- a的值和&a的值一样吗？
- sizeof(a) == sizeof(a+1-1)？

2、int a[10];
- a的值和&a的值一样吗？
- sizeof(a) == sizeof(a+1-1)？

3、void fun(int a[][20])
数组作为形参的时候，第一维大小为什么没了？

4、const int* p、int const* p和int* const p有区别吗?

交流内容

- 变量类型的再思考
- 分配给变量的内存如何描述
- 一维/高维数组区别在哪
- 形参的取值（理解Pass by Value）
- 动态内存的逻辑含义
- const的使用

内存基本概念简单回顾

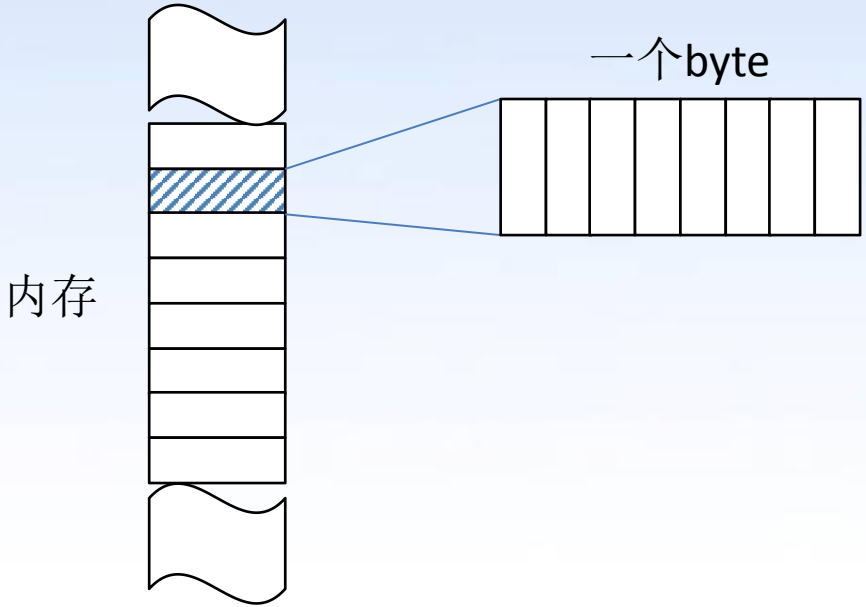
bit（比特）
状态0/1
二进制

8个bit拼在一起，组成一个byte（字节）
存储范围00000000~11111111

二进制转十进制

0~255

内存长什么样？



内存就是一系列byte顺序排列在一起组成的存储结构

我们机器里的内存有多少个这样的byte呢？

我们说8/16/32/64位机到底是什么意思？

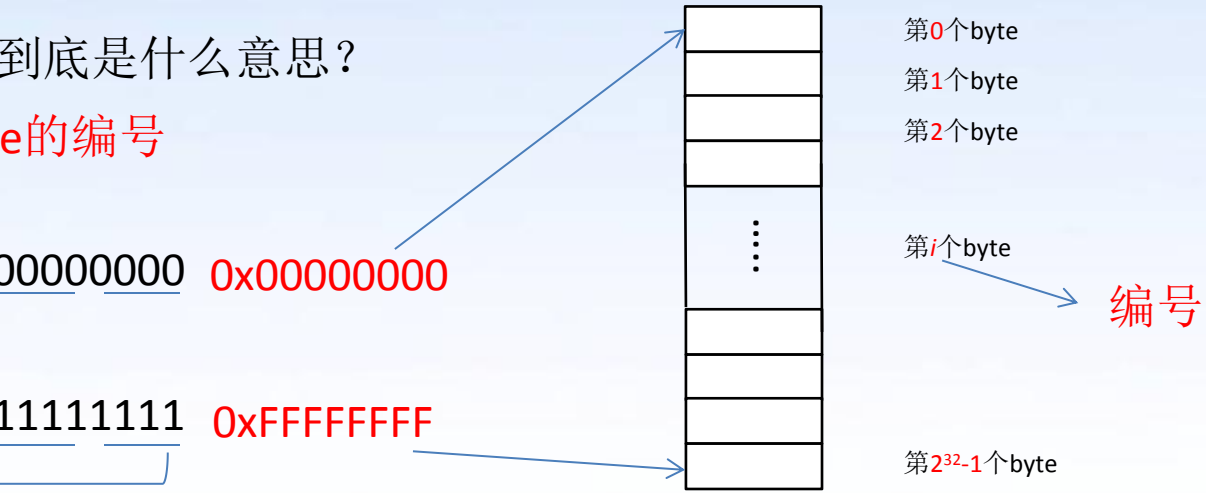
32位bit来表示byte的编号

00000000000000000000000000000000 0x00000000

……

11111111111111111111111111111111 0xFFFFFFFF

32位



可以编号的byte个数一共可以有2^{32}个，$2^{32}=2^2*2^{10}*2^{10}*2^{10}$

4G内存，其中1K=1024Byte，1M=1024K, 1G=1024M

测试题

1、32位机最多能访问4G内存的原因是什么？

2、对64位机器来说，最多能访问的byte数量有多少？

测试题

1、32位机最多能访问4G内存的原因是什么？

2、对64位机器来说，最多能访问的byte数量有多少？

答案：

1、32位机最多能访问的内存byte总数为2^{32}个，因此:

$$2^{32} \div 1024 = 2^{22}\text{K}，\ \text{1K=1024个字节}$$

$$2^{22}\text{K} \div 1024 = 2^{12}\text{M}，\ \text{1M=1024K字节}$$

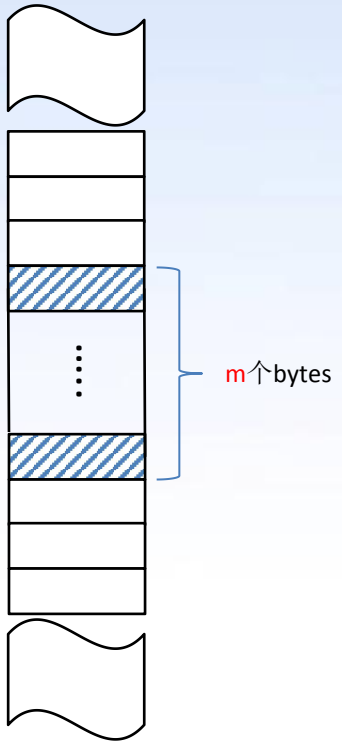
$$2^{12}\text{M} \div 1024 = 2^{2}\text{G} = 4\text{G}，\ \text{1G=1024M字节}$$

2、64位机最多能访问的byte总数为2^{64}个（理论上）

内存的相关操作

分配内存

内存赋值



释放内存

读取内存相关信息

分配内存

在C语言中，分配一段内存的方法主要有两种：

1、变量声明 （例如：局部变量、全局变量等）

2、使用malloc分配

无论那种方法，内存分配的过程都是一样的，
区别只是释放内存的不同需求

我们先来看变量声明的方法

基本概念：变量声明

在C语言中，声明一个变量具有两个基本要素：
1、变量类型
2、变量名称

变量类型约定了这段内存的类型和大小
变量名称则是这块内存的别名，用于识别定位这块内存

变量类型进一步分为：
1、非数组变量类型
2、数组变量类型

基本概念：非数组变量类型及声明

包括int, short, float, double, char, 结构体/联合体等等

声明方式如下：

变量类型为int, 变量名称为a	int a;
变量类型为float, 变量名称为b	float b;
变量类型为double, 变量名称为c	double c;

任何变量类型分配的内存大小可以通过sizeof这个操作符来获取，例如：

sizeof(int), sizeof(float), sizeof(double)

基本概念：数组变量类型

数组变量类型是多个同样的变量类型组成的一种一维构造类型
包含两要素：1、元素个数，2、元素的变量类型

1、数组变量类型的元素可以是非数组变量类型

int[2]包括2个元素，每个元素变量类型是int
char[3]包括3个元素， 每个元素变量类型是char

任何数组变量类型分配的内存也可以通过sizeof这个操作符来获取

sizeof(int[2]), sizeof(char[3])

基本概念：数组变量类型（续）

数组变量类型是多个同样的变量类型组成的一种一维构造类型
包含两要素：1、元素个数，2、元素的变量类型

2、数组变量类型的元素也可以是其他数组变量类型

float[2][3]包括2个元素，每个元素变量类型是float[3]
double[2][3][4]包括2个元素，每个元素变量类型是double[3][4]

仍然可以通过sizeof这个操作符来获取变量类型的大小

sizeof(float[2][3]), sizeof(double[2][3][4])

注意：C语言数组变量类型其实都是一维的

基本概念：数组变量类型的声明

数组变量类型申明方式如下：

变量类型为int[2], 变量名称为a	int a[2];
变量类型为float[2][3], 变量名称为b	float b[2][3];
变量类型为double[2][3][4], 变量名称为c	double c[2][3][4];

需要能正确的识别出数组变量声明中的变量类型和变量名称

思考题

1、char[3][5]数据类型包括几个元素，每个元素是什么数据类型？

2、sizeof(int[7][8])=？

思考题

1、char[3][5]数据类型包括几个元素，每个元素是什么数据类型？

2、sizeof(int[7][8])=？

答案：
1、char[3][5]数据类型是一个一维数组类型，包括3个元素，其中每个元素的数据类型是char[5]

2、sizeof(int[7][[8])=7*sizeof(int[8])=7*8*sizeof(int)=7*8*4=224字节

基本概念：指针变量类型

1、指针变量类型是一个非数组变量类型

2、任何一种变量类型，都有指向其的一个指针变量类型，反之亦然

int、float、double

int[2]、float[2][3]、double[2][3][4]

int*、float*、double*

int(*)[2]、float(*)[2][3]、double(*)[2][3][4]

指向int变量类型的指针变量类型是int*

指向int[2]变量类型的指针变量类型是int(*)[2]

这是C语言语法的规范

基本概念：指针变量类型（续）

指针变量类型也有相应的指向其的另一个指针变量类型

int*、float*、double*

int(*)[2]、float(*)[2][3]、double(*)[2][3][4]

int**、float**、double**

int(**)[2]、float(**)[2][3]、double(**)[2][3][4]

指向int*变量类型的指针变量类型是int**

指向int(*)[2]变量类型的指针变量类型是int(**)[2]

基本概念：指针变量类型

指针变量类型也可以用来构造数组类型，作为数组元素的变量类型

int*[2]、float*[2][3]、double*[2][3][4]

int*[2]包括2个元素，每个元素数据类型是int*
float*[2][3]包括2个元素，每个元素变量类型是float*[3]

指针变量类型构成的数组变量也有其对应的指针变量

int*[2]、float*[2][3]、double*[2][3][4] ⟺ int*(*)[2]、float*(*)[2][3]、double*(*)[2][3][4]

提示：把int*当作一个整体来看待

基本概念：指针变量申明

变量类型是int*，变量名是a	int* a
变量类型是float*，变量名是b	float* b
变量类型是double*，变量名是c	double* c
变量类型是int(*)[2]，变量名是d	int (*d)[2]
变量类型是float(*)[2][3]，变量名是e	float (*e)[2][3]
变量类型是double(*)[2][3][4]，变量名是f	double (*f)[2][3][4]
变量类型是int*[2]，变量名是g	int* g[2]
变量类型是float*[2][3]，变量名是h	float* h[2][3]
变量类型是double*[2][3][4]，变量名是i	double* i[2][3][4]

理解sizeof(int*), sizeof(int(*)[2]), sizeof(int*[2])

int* p vs. int *p：空格放在哪更好？

```
int *p;
int* p;

int *p, q;
int* p, q;

int *p, *q;

int* p;
int* q;
```

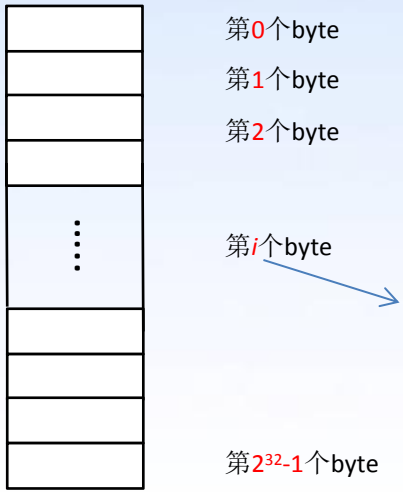
1、int *p和int* p是一样的
都定义了一个int*类型的变量，变量名为p

2、int *p, q和int* p, q是一样的
定义了一个int*类型的变量，变量名为p，同时定义了一个int类型的变量，变量名为q

3、int *p, *q才是定义了两个int*类型的变量，变量名分别为p和q

4、建议int*，因为int*是一个数据类型，因此多个int*类型变量申明，建议一行一个

指针变量类型的取值范围



32位机内存

任何指针变量的值就是内存中一个字节的编号

取值范围就是[0 ~ 最大可寻址的字节编号]

对32位机器而言，可以寻址的字节编号范围为
[0 ~ 2^{32}-1]

转换成16进制表示
[0x00000000 ~ 0xFFFFFFFF]

0x00000000=NULL

基本概念：变量类型总结

假设变量类型Variable_Type，变量名Name

1、变量申明可以形式化定义为Variable_Type Name;

2、变量类型Variable_Type对应的指针类型可以形式化记为Variable_Type*

注1：具体变量申明语法和变量类型有关，例如：
int a; vs. int b[10];

注2： Variable_Type*里面*的位置跟语法有关，例如：
int*; vs. int(*)[10];

基本概念：变量类型总结

Variable Category	Variable Type	Variable Name	Variable Declaration	Element Type	Corresponding Pointer Type	Size
Non-Array Variables	int	a	int a;	N/A	int*	4
	char	b	char b;	N/A	char*	1
	float	c	float c;	N/A	float*	4
	int*	d	int* d;	N/A	int**	4
	char*	e	char* e;	N/A	char**	4
	float*	f	float* f;	N/A	float**	4
Array Variables	int[2]	g	int g[2];	int	int(*)[2]	8
	char[2][3]	h	char h[2][3];	char[3]	char(*)[2][3]	6
	float[2][3][4]	i	float i[2][3][4];	float[3][4]	float(*)[2][3][4]	96
	int*[2]	j	int* j[2];	int*	int*(*)[2]	8
	char*[2][3]	k	char* k[2][3];	char*[3]	char*(*)[2][3]	24
	float*[2][3][4]	l	float* l[2][3][4];	float*[3][4]	float*(*)[2][3][4]	96

思考题

1、int**[3] 变量类型包括几个元素，每个元素是什么数据类型？

2、指向int**[3]变量类型的指针变量类型是什么？

3、如何声明一个变量，变量名为a，变量类型为int**[3]

思考题

1、int**[3] 变量类型包括几个元素，每个元素是什么数据类型？

2、指向int**[3]变量类型的指针变量类型是什么？

3、如何声明一个变量，变量名为a，变量类型为int**[3]

答案（提示：int**也是一个数据类型，将int**当作一个整体来看）：
1、int**[3]数据类型是一个一维数组类型，包括3个元素，其中每个元素的数据类型是int**

2、int**(*)[3]

3、int** a[3]（建议）或int **a[3]

思考题

1、32位机里面，int*数据类型的取值需要几个字节来存储？

2、64位机里面，int*的取值范围是多大？需要几个字节来存储？

思考题

1、32位机里面，int*数据类型的取值需要几个字节来存储？

2、64位机里面，int*的取值范围是多大？需要几个字节来存储？

答案

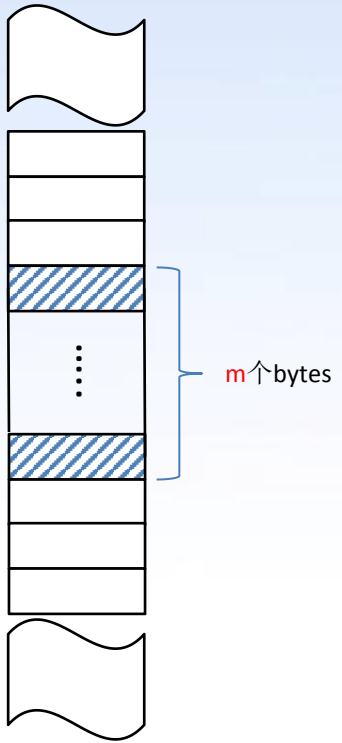
1、32位机里面，int*的取值范围是[0x00000000, 0xFFFFFFFF]，因此4个字节来存储就够了，所以sizeof(int*)=4

2、64位机里面，理论上可以访问的字节有2^{64}个，因此字节编号范围为[0x0000000000000000, 0xFFFFFFFFFFFFFFFF]，需要8个字节来存储，因此sizeof(int*)=8

sizeof(int*)可以判断你的编译器是8/16/32/64位

内存的相关操作

分配内存
1、通过变量声明
2、通过malloc



内存赋值

释放内存

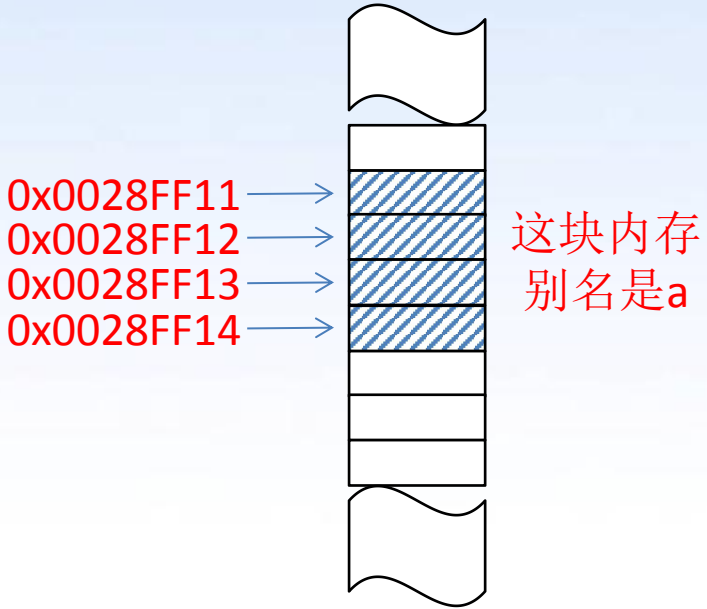
读取内存相关信息

声明变量时的内存分配

对于这样一个变量声明语句：int a;

1、分配4个连续byte（为什么是4？）

2、这4个byte组成的块标记为a



思考：0x0028FF11~0x0028FF14怎么来的？

如何描述变量a对应的内存

int a；

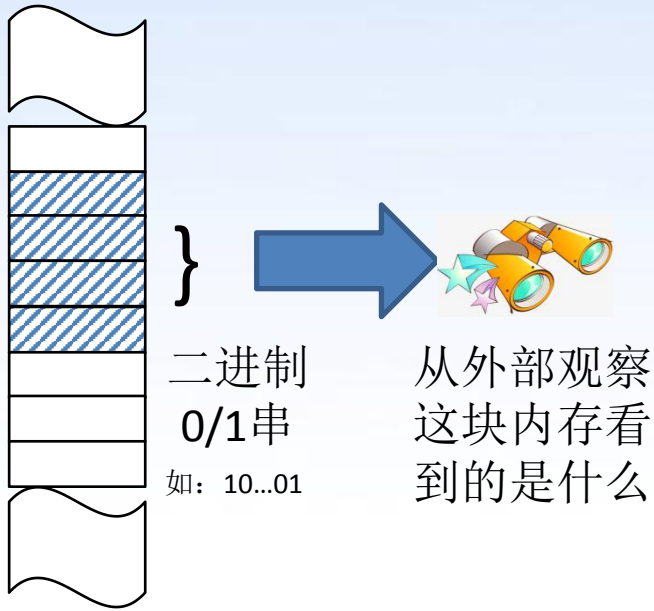
1、这块连续内存第一个byte编号是0x00FF2811

2、这块内存的变量类型是int

3、这块内存的别名是a

4、这块内存大小是4个字节（sizeof(int)）

5、内存的表示值



值（Value）是什么？

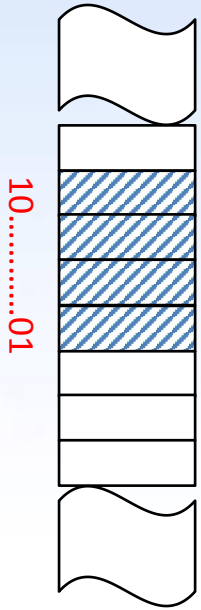
在C语言中，值包含了两个层面的语义：1）值；2）值的类型

记为<Value, Value_Type>

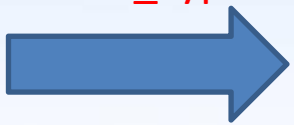
任何一个表达式都是有值的，例如：
10是一个表达式，值为<10, int>
'b'是一个表达式，值为<'b', char>
10>20也是一个表达式，值为<0, int>（c99标准才有bool类型）

变量a所对应的那块内存也有表示值，也记为<Value, Value_Type>

进一步理解内存的表示值

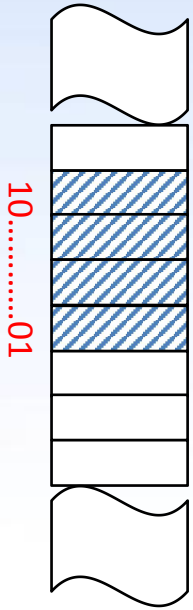


Value_Type

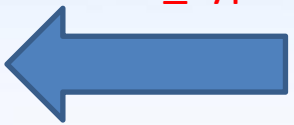


Value

按照表示值类型去观察内存得到我们能够理解的值



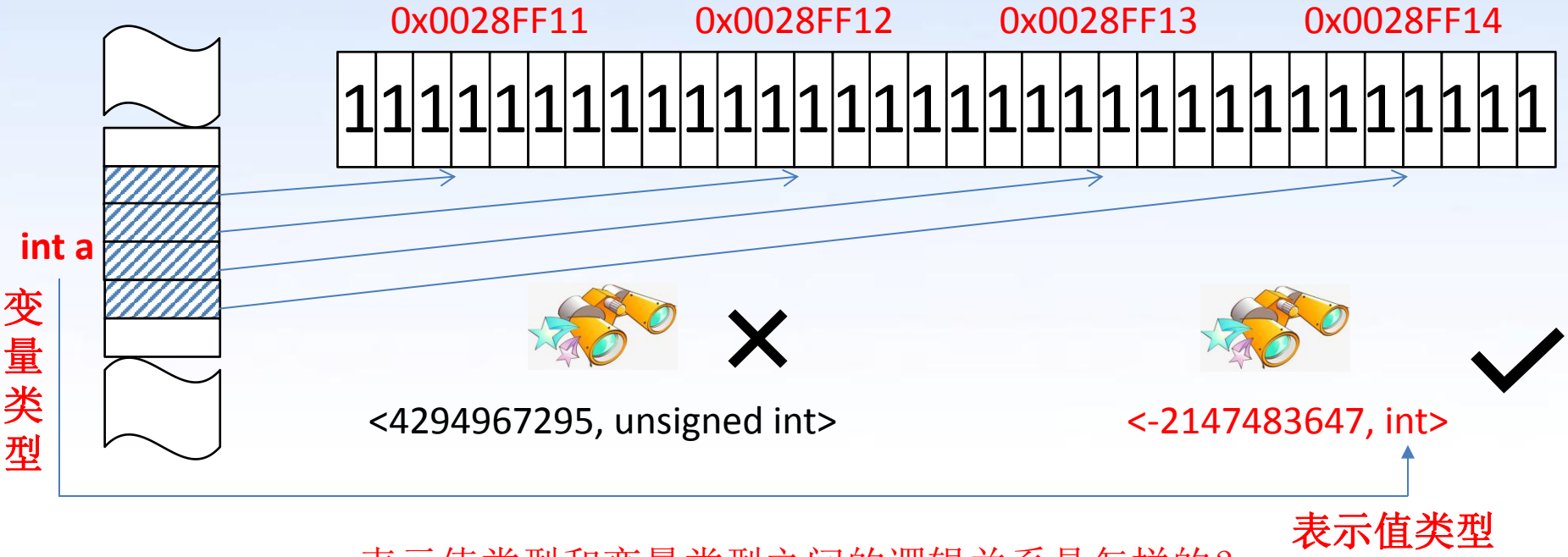
Value_Type



Value

将一个值按照表示值类型写入内存

进一步理解内存的表示值

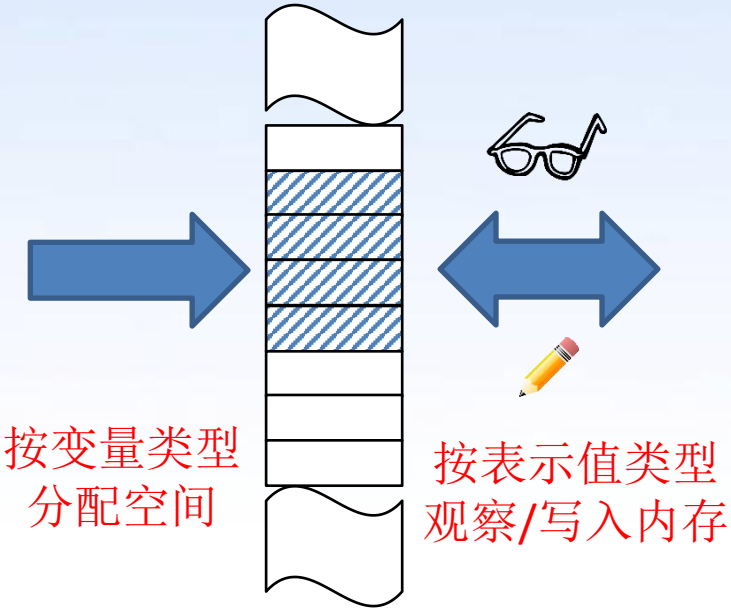


表示值类型和变量类型之间的逻辑关系是怎样的？

内存对应的变量类型 vs. 表示值类型

1、内存的变量类型是Physical View，是在内存分配时按什么类型去申请内存

2、内存的表示值类型是Logical View，是这块内存从外部观察能看到的值的类型



表示值类型 vs. 变量类型

内存表示值与内存对应的变量类型关系

假设内存表示值记为<Value, Value_Type>，内存对应的变量类型记为Variable_Type

1、如果Variable_Type是非数组类型

Value_Type=Variable_Type，Value则是通过Value_Type去观察这段内存获得的值

例如：int a，Variable_Type是int，Value_Type也是int

2、如果Variable_Type是数组类型

Value_Type是该数组类型中元素变量类型对应的指针类型，Value是数组第一个元素所处内存的第一个字节编号

例如：int a[10]，Variable_Type是int[10]，元素类型是int，Value_Type是int*

思考题

1、char c; double d; float f[3][4]; int* p[3];
这四个变量对应的内存表示值的类型是什么？

思考题

1、char c; double d; float f[3][4]; int* p[3];
这四个变量对应的内存表示值的类型是什么？

答案

1、char c; 变量类型Variable_Type为char，非数组类型，
Value_Type=char

2、double d; 变量类型Variable_Type为double，非数组类型，
Value_Type=double

3、float f[3][4]; 变量类型Variable_Type为float[3][4]，数组类型，元素类型为float[4]，
Value_Type=float(*)[4]

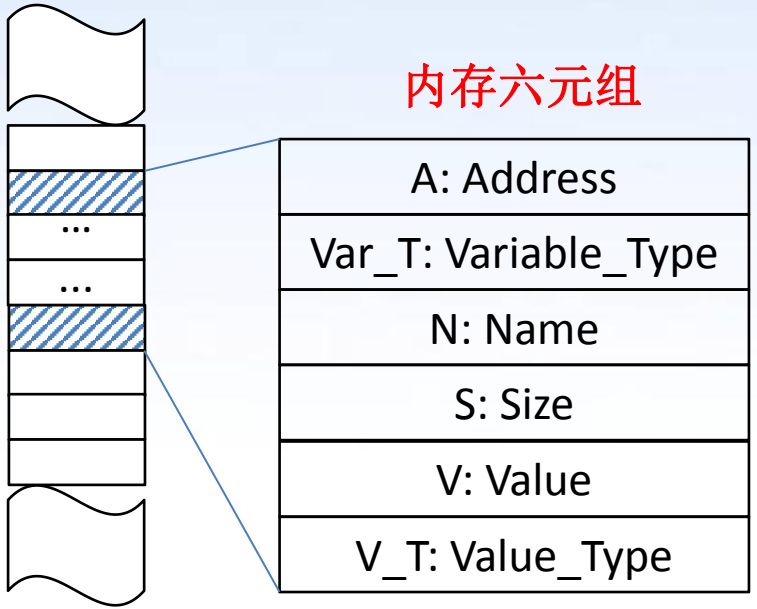
4、int* p[3]; 变量类型Variable_Type为int*[3]，数组类型，元素类型为int*，
Value_Type=int**

内存六元组模型

M = {Address, Variable_Type, Name, Size, Value, Value_Type}

M：声明变量系统给分配的一段内存

- Address：这一段内存第一个字节的编号
- Variable_Type: 变量类型
- Name：变量名称
- Size：内存大小（字节数量）
- Value：这段内存的表示值
- Value_Type：表示值的类型



内存六元组取值规则

M = {Address, Variable_Type, Name, Size, Value, Value_Type}

1、Address由系统分配，一旦确定无法修改

2、Variable_Type和Name是变量声明对应的变量类型和变量名

3、Size是这块内存的大小（字节数）

4、Value和Value_Type的取值根据Variable_Type来确定

Variable_Type是非数组变量类型 vs. 数组变量类型

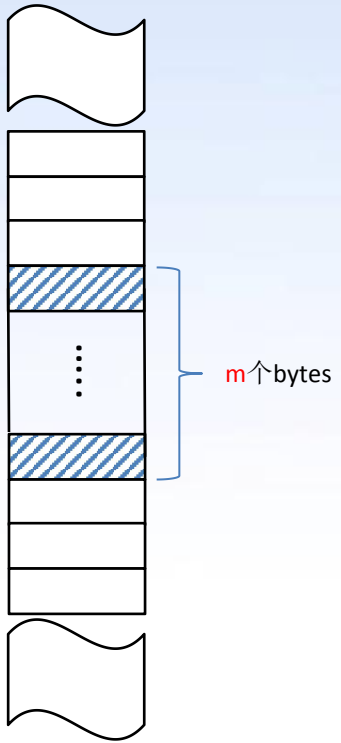
内存六元组是为了让大家更好的了解分配的内存而提出的一个逻辑模型

内存的相关操作

分配内存

1、通过变量声明

2、通过malloc

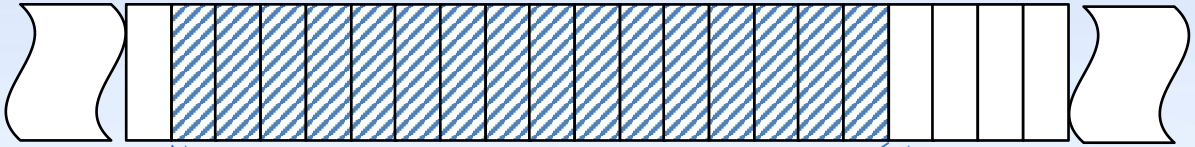


内存赋值

释放内存

读取内存相关信息

观察：malloc(16)



0x00351728　　0x00351738

A: 0x00351728
Var_T: N/A
N: N/A
S: 16
V: N/A
V_T: N/A

```
malloc(16);
```

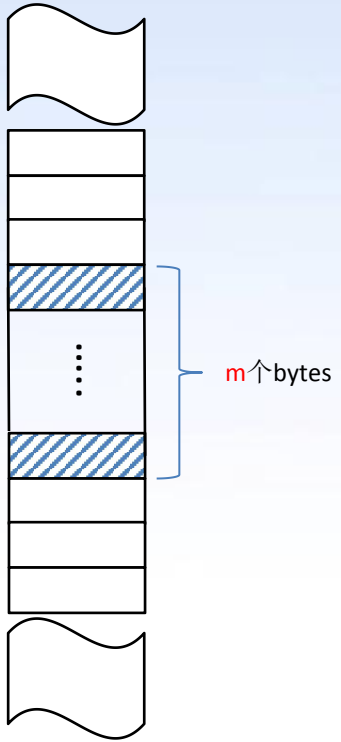
思考1：为什么有Address和Size

思考2：这块内存为什么没有Variable_Type和Name

思考3：为什么这块内存没有Value和Value_Type

内存的相关操作

分配内存
1、通过变量声明
2、通过malloc



内存赋值

释放内存

读取内存相关信息

内存赋值：简单示例int a=10;

int a=10; vs. int a; a=10;

```
int a = 10;
```

声明变量的同时为这块内存赋值（变量初始化）

```
int a;
a = 10;
```

先声明变量，然后再为这块内存赋值（变量赋值）

对非数组类型变量来说，这两种方式没有区别
对数组类型变量来说，只能使用变量初始化的方式为内存赋值

我们先来看变量赋值

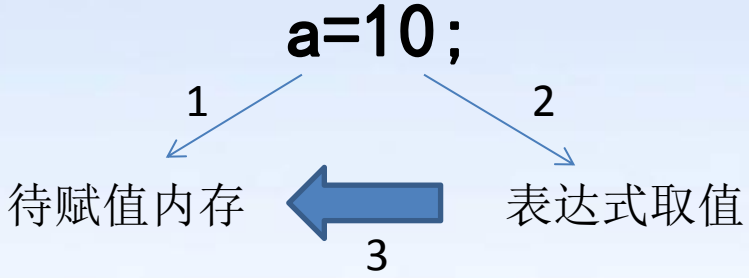
变量赋值示例：a=10；

int a; a = 10;

左值　　右值

在C语言中，只能对一块内存进行赋值，因此
1、左值：要求必须是一块有效内存
2、右值：要求必须是一个有效表达式

变量赋值的过程



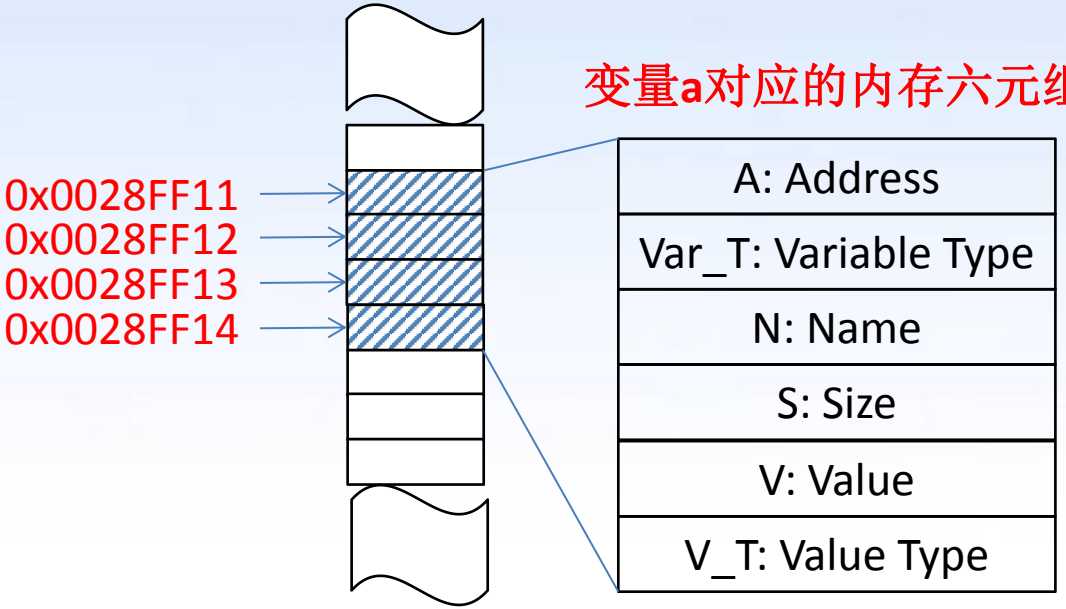
这个变量赋值过程：对10这个表达式取值，然后将值赋给变量a所对应的内存

1、内存如何识别？如何描述？

2、表达式怎么取值？

3、取值如何往内存里存？

通过int a变量声明出来的内存如何描述



变量a对应的内存六元组

A: Address
Var_T: Variable Type
N: Name
S: Size
V: Value
V_T: Value Type

Address: 0x0028FF11
Variable Type: int
Name: a
Size: 4
Value: ?
Value Type: int

Variable_Type是非数组变量类型
Value_Type = Variable_Type

思考：这段内存有Value吗？

内存如何定位

1、通过变量名定位内存：即通过变量名来定位这段内存，例如：

int a; float b; double c;

通过a，b，c就可以定位到对应的内存

2、通过*运算符来定位内存：假设一个表达式expression的取值为<Value, Value_Type>，如果Value_Type是一个指针类型，则可以用*expression的方式来定位到Value对应字节编号开头的一段内存

我们先来看通过变量名定位内存

int类型变量a赋值为10过程

a=10;

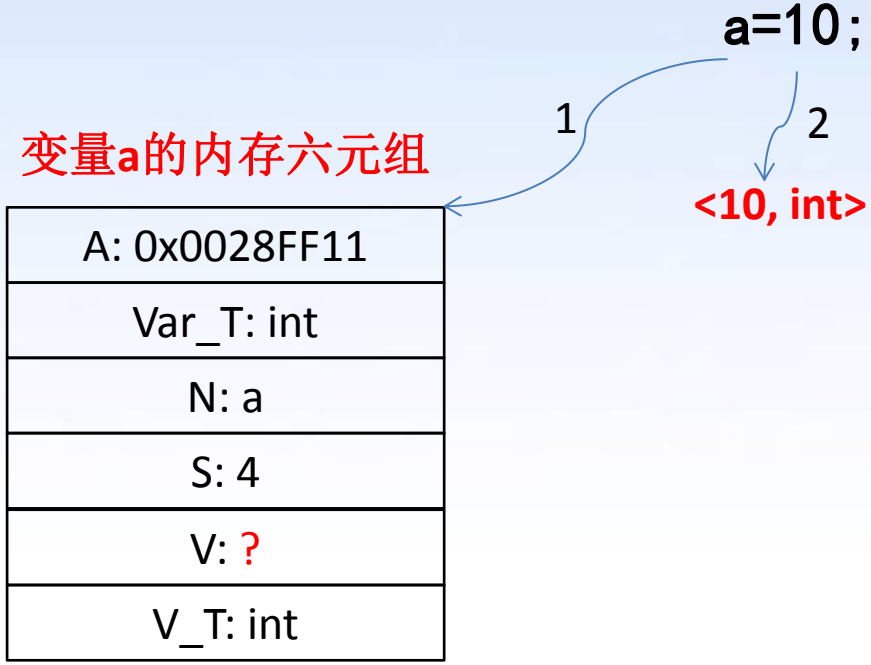
1

1、根据变量名a定位内存（用六元组描述）

变量a的内存六元组

A: 0x0028FF11
Var_T: int
N: a
S: 4
V: ?
V_T: int

int类型变量a赋值为10过程

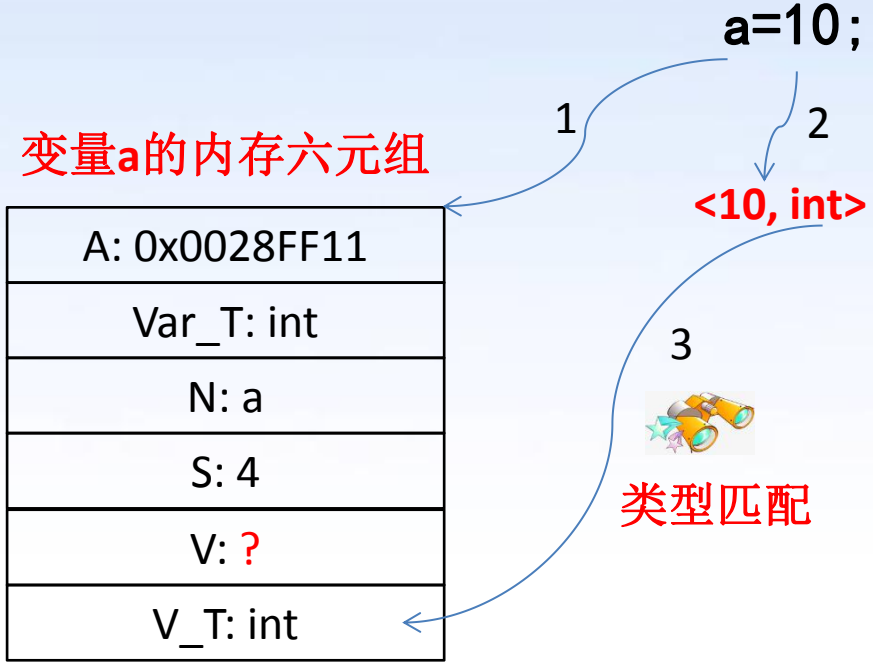


1、根据变量名a定位内存（用六元组描述）

2、获得右值表达式的值

<Value, Value_Type>（<10, int>）

int类型变量a赋值为10过程



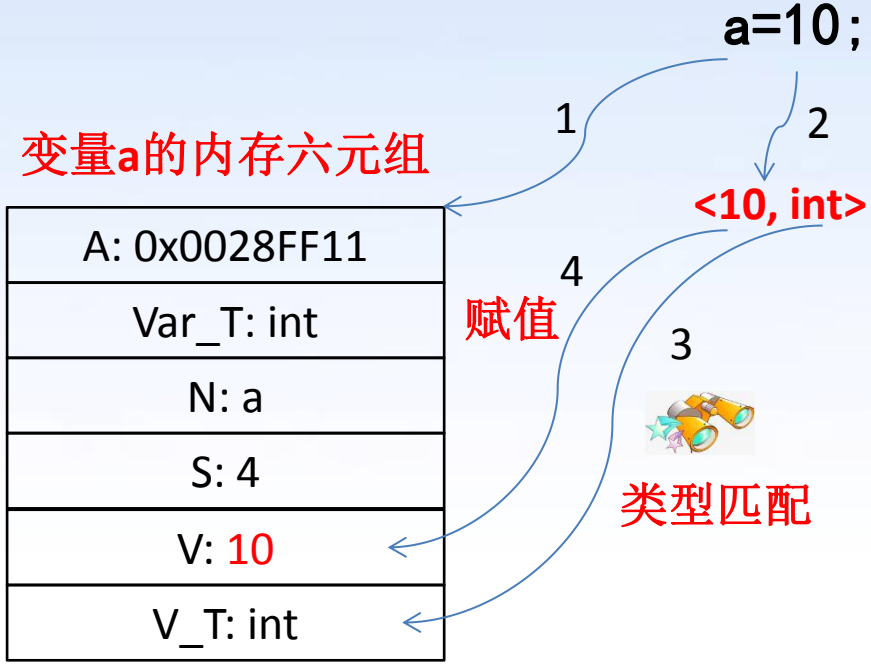
1、根据变量名a定位内存（用六元组描述）

2、获得右值表达式的值

<Value, Value_Type>（<10, int>）

3、检查右值Value_Type是否与内存表示值的Value_Type匹配

int类型变量a赋值为10过程



1、根据变量名a定位内存（用六元组描述）

2、获得右值表达式的值

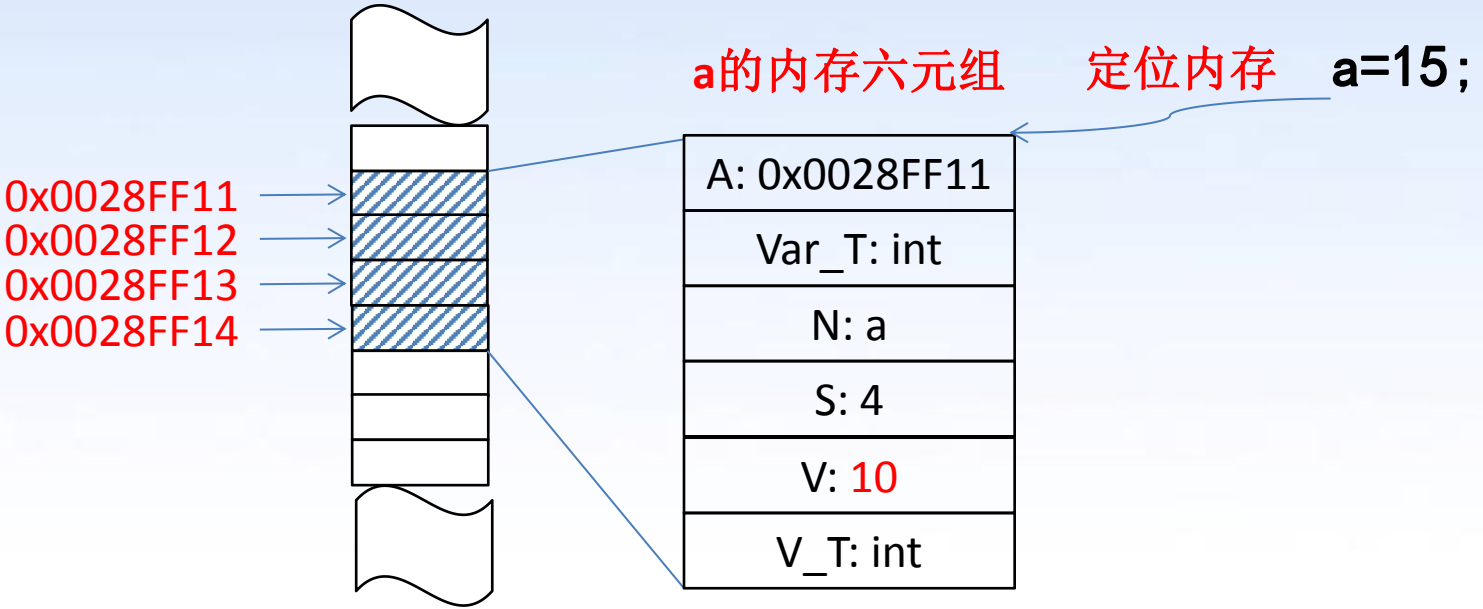
<Value, Value_Type>（<10, int>）

3、检查右值Value_Type是否与内存表示值的Value_Type匹配

4、将右值Value赋值到内存表示值的Value

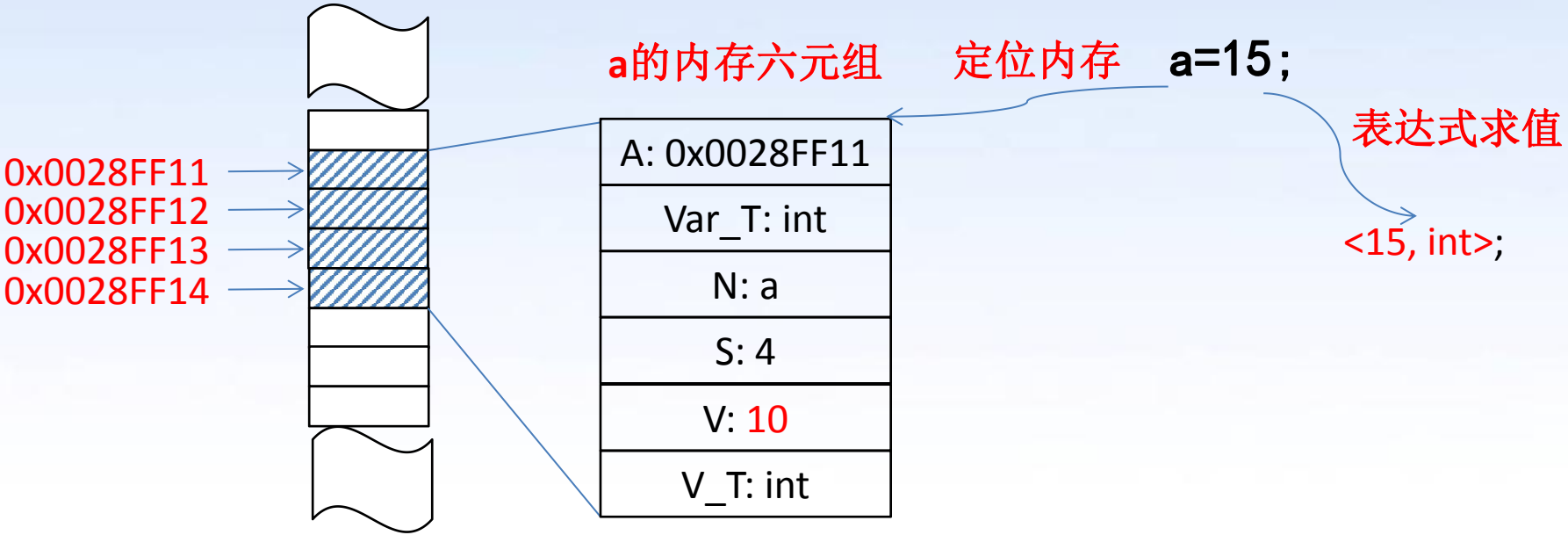
系统将这个Value(10)按Value_Type(int)类型转化成32位0/1值存入到a所在的4个字节中

进一步a=15发生了什么？

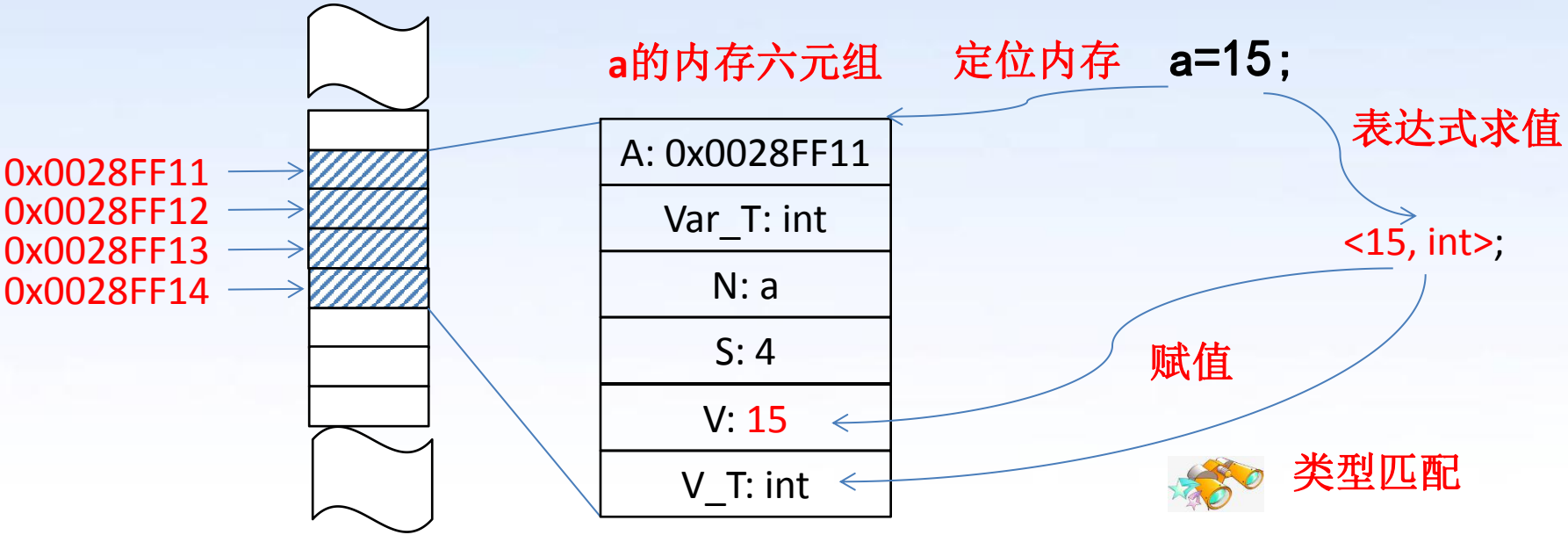


思考：现在Value为什么是10

进一步a=15发生了什么？

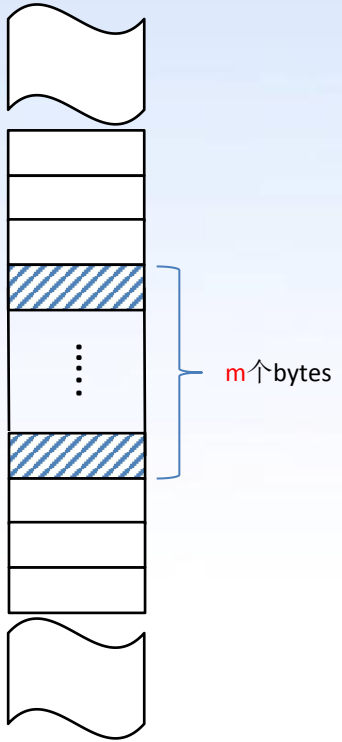


进一步a=15发生了什么？



其他变量类型声明示例

分配内存
1、通过变量声明
2、通过malloc



内存赋值

释放内存

读取内存相关信息

其他变量类型声明示例

```
int a = 10;

float b = 1.0;
double c = 2.0;
char d = 'a';

int* p;

int e[2];
char f[8];

int g[2][3];

struct m_struct {
    int a;
    float b;
}h;

union m_union {
    int a;
    float b;
}i;
```

基本数据类型，例如：int, float, double, char

指针类型，例如：int*

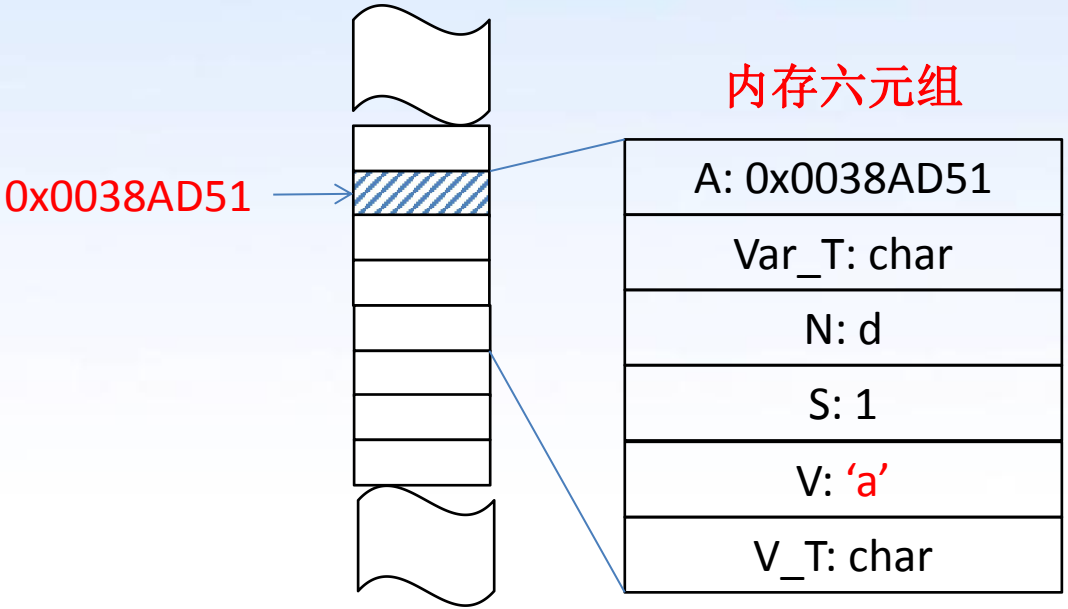
一维数组，例如：int[2], char[8]

“二维数组”，例如：int[2][3] （C语言数组都是一维的）

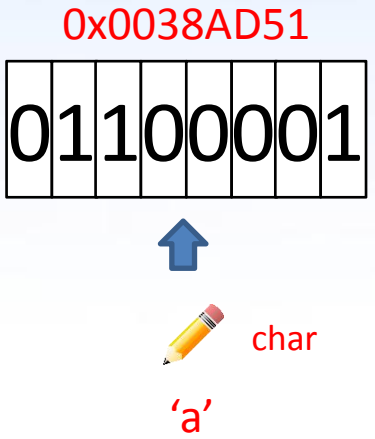
结构体/联合体，例如：struct m_struct/union m_union

观察：char d='a';

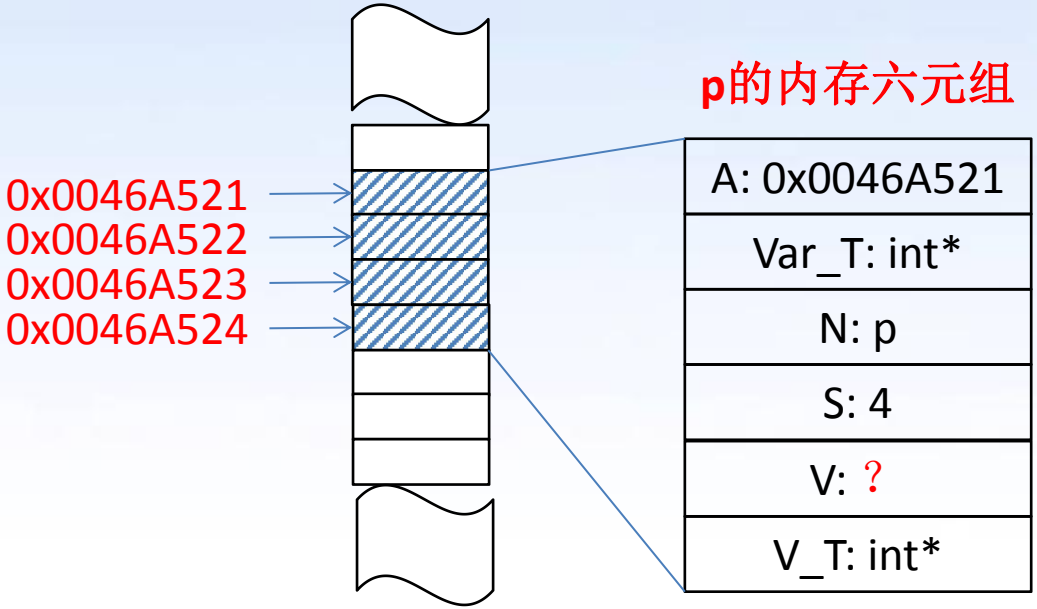
这是变量初始化，分配内存的同时对内存赋值



将'a'按char类型翻译成0/1值



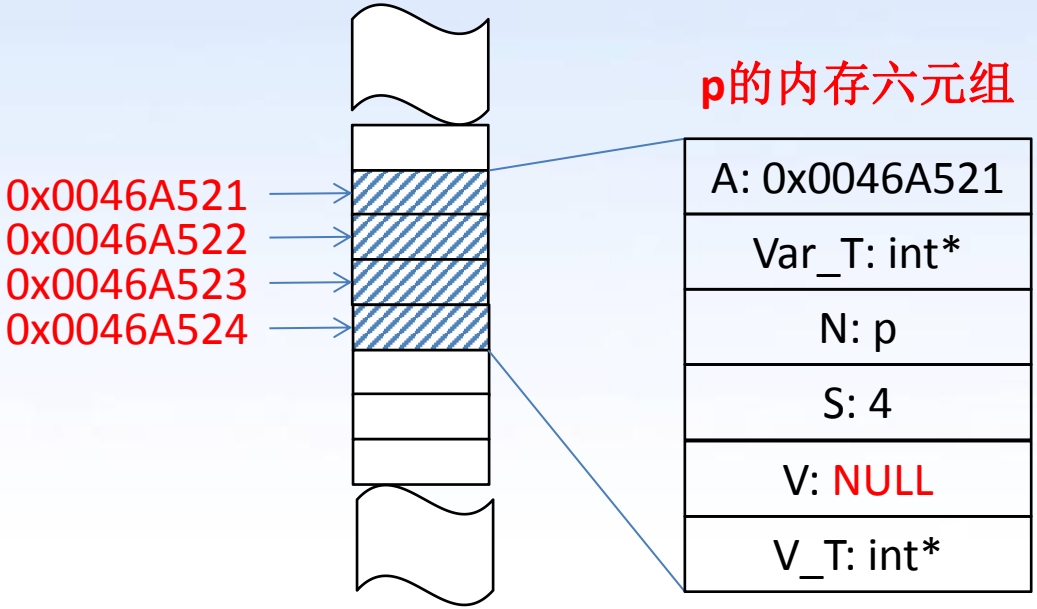
观察：int* p;



Address: 0x0046A521
Variable Type: int*
Name: p
Size: 4
Value: ? (Uninitialized)
Value Type: int*

思考1：为什么size是4
思考2：这个时候value有值吗？

改进一下：int* p=NULL;



Address: 0x0046A521
Variable Type : int*
Name: p
Size: 4
Value: NULL
Value Type: int*

思考1：初始化为NULL的好处
思考2：int*的取值范围是什么？

复习一下：数组变量类型的表示值

假设内存表示值记为<Value, Value_Type>，内存对应的变量类型记为Variable_Type

1、如果Variable_Type是**非数组类型**

Value_Type=Variable_Type，Value则是通过Value_Type去观察这段内存获得的值

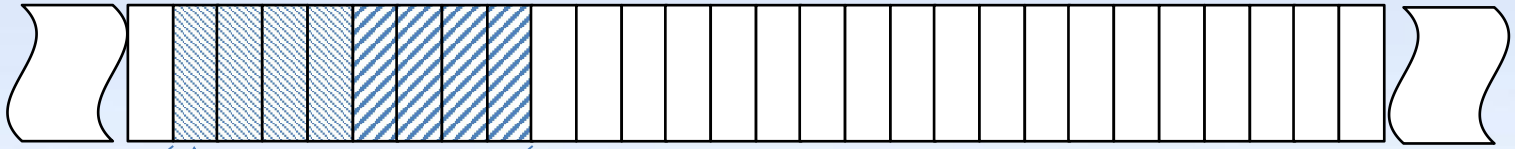
例如：int a，Variable_Type是int，Value_Type也是int

2、如果Variable_Type是**数组类型**

Value_Type是该数组类型中元素变量类型对应的指针类型，Value是数组第一个元素所处内存的第一个字节编号

例如：int a[10]，Variable_Type是int[10]，元素类型是int，Value_Type是int*

观察：int e[2];



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

1、数组类型：Variable_Type是int[2]

2、数组元素类型是int，其对应的指针类型是int*
Value_Type = int*

3、第一个元素变量类型是int，所占4个字节编号
[0x0076AB11, 0x0076AB14]

观察：int e[2];e=NULL;

```
int e[2];

e=NULL;
```

```
=== Build file: "no target" in "no project" (compiler: unknown) ===
In function 'main':
error: assignment to expression with array type
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

✗ 赋值

<NULL, int*>;

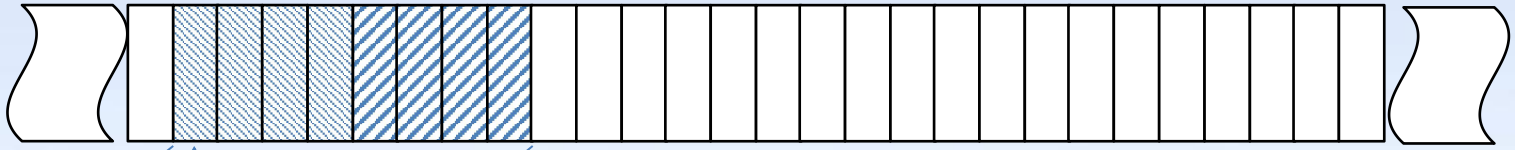
类型匹配 ✓

数组变量内存的Value一定是指向数组第一个元素的地址编号

V的值必须和A相等

变量e并不是一个常量，它的值不能更改是语法限制的

观察：int e[2]={1, 2};



0x0076AB11

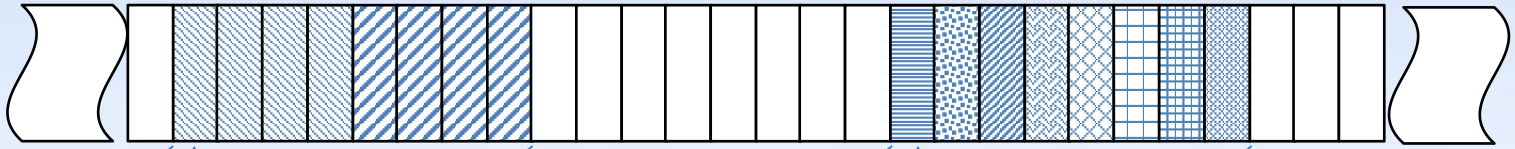
A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

<1, int> <2, int>

1、数组对应的内存赋值只能通过变量初始化

2、数组对应的内存表示值不会发生任何变化

观察：int e[2];和char f[8];



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

0x0076AB21

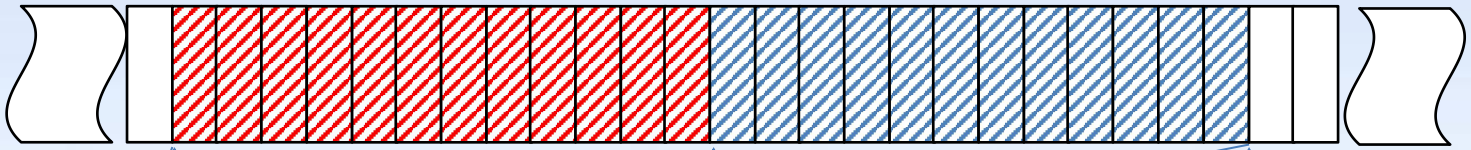
A: 0x0076AB21
Var_T: char[8]
N: f
S: 8
V: 0x0076AB21
V_T: char*

思考：为什么大小都是8

sizeof(int[2]) = 8
2*sizeof(int)

sizeof(char[8]) = 8
8*sizeof(char)

观察：int g[2][3];



A: 0x0098B11D
Var_T: int[2][3]
N: f
S: 24
V: 0x0098B11D
V_T: int(*)[3]

1、数组类型：Variable_Type是int[2][3]

2、数组元素类型是int[3]，其对应的指针类型是int(*)[3]
Value_Type = int(*)[3]

3、第一个元素变量类型是int[3]，所占12个字节编号
[0x0098B11D, 0x0098B128]

思考题

1、声明一个变量int m[2][3][4]，变量m对应这个内存的Variable_Type=?
假设m变量第一个字节编号为0x0037DC11

2、这个数组变量的元素变量类型是什么？

3、变量m对应的内存的表示值类型Value_Type是什么？

思考题

1、声明一个变量int m[2][3][4]，变量m对应这个内存的Variable_Type=?
假设m变量第一个字节编号为0x0037DC11

2、这个数组变量的元素变量类型是什么？

3、变量m对应的内存的表示值类型Value_Type是什么？

A: 0x0037DC11
Var_T: int[2][3][4]
N: m
S: 96
V: 0x0037DC11
V_T: int(*)[3][4]

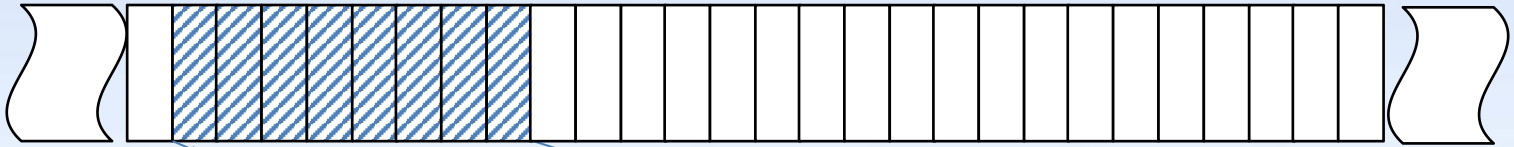
答案：

1、Variable_Type=int[2][3][4]

2、该数组变量的元素类型为int[3][4]

3、Value_Type=int(*)[3][4]

观察：结构体变量h



0x0085D611

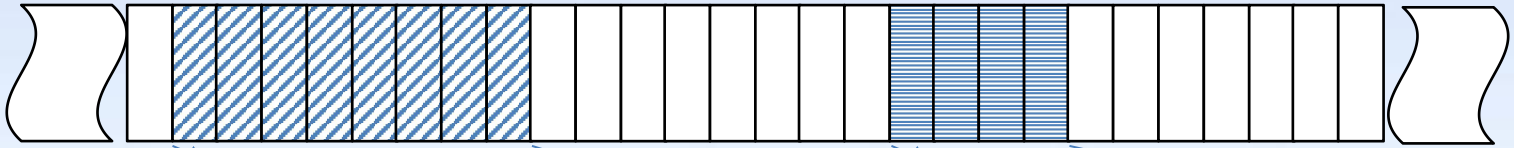
```
struct m_struct {
    int a;
    float b;
}h;
```

A: 0x0085D611
Var_T: struct m_struct
N: h
S: 8
V: Invisible
V_T: struct m_struct

struct m_struct是非数组变量类型
sizeof(struct m_struct)

结构体变量h对应的内存也有Value，但是我们却看不懂

观察：对比结构体变量h和联合体变量i



0x0085D611

0x0085D621

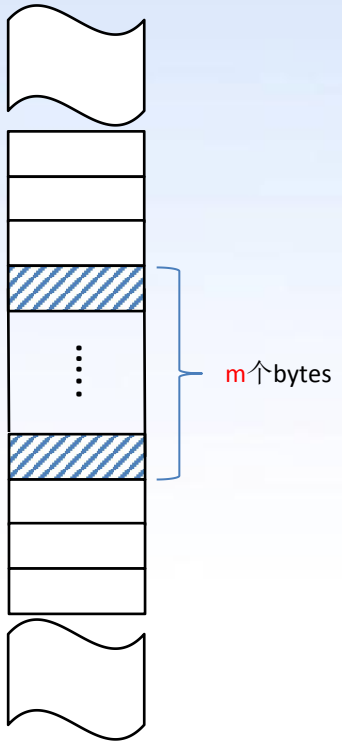
```
struct m_struct {
    int a;
    float b;
}h;

union m_union {
    int a;
    float b;
}i;
```

A: 0x0085D611
Var_T: struct m_struct
N: h
S: 8
V: Invisible
V_T: struct m_struct

A: 0x0085D621
Var_T: union m_union
N: i
S: 4
V: Invisible
V_T: union m_union

内存的相关操作

分配内存
1、通过变量声明
2、通过malloc

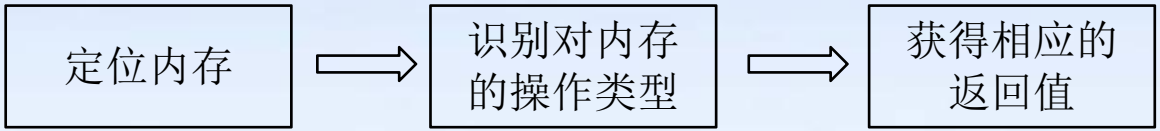


内存赋值

释放内存

读取内存相关信息

内存相关信息读取的三种操作



定位内存的合法表达式：

1、变量名表达式

例如： a, b, c

2、*+指针类型表达式

例如: *p, *(p+1)

每一块内存都有相关有三种操作类型，每一种都获得一个返回值：

1、获得内存的首地址

&(表达式)

2、获得内存的大小

sizeof(表达式)

3、获得内存的表示值

表达式本身

返回值包括：

1、返回值的类型

2、返回值的值

<Value, Value_Type>

复习一下：内存如何定位

1、通过变量名定位内存：即通过变量名来定位这段内存，例如：

int a; float b; double c;

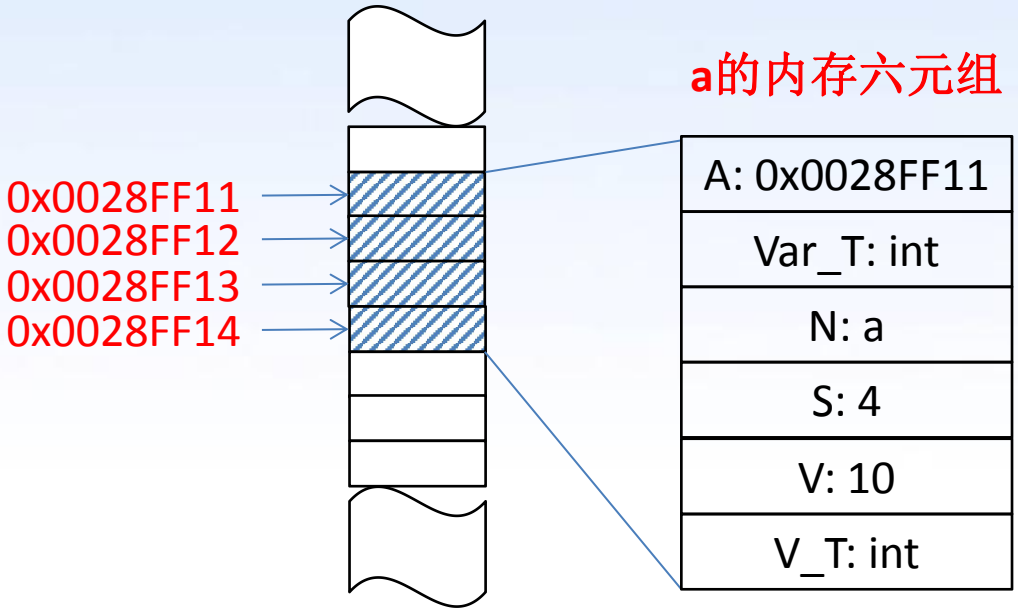
通过a，b，c就可以定位到对应的内存

2、通过*运算符来定位内存：假设一个表达式expression的取值为<Value, Value_Type>，如果Value_Type是一个指针类型，则可以用*expression的方式来定位到Value对应字节编号开头的一段内存

我们先来看对通过变量名定位内存的取值相关操作

对内存的三种取值操作

给定int a; a=10;



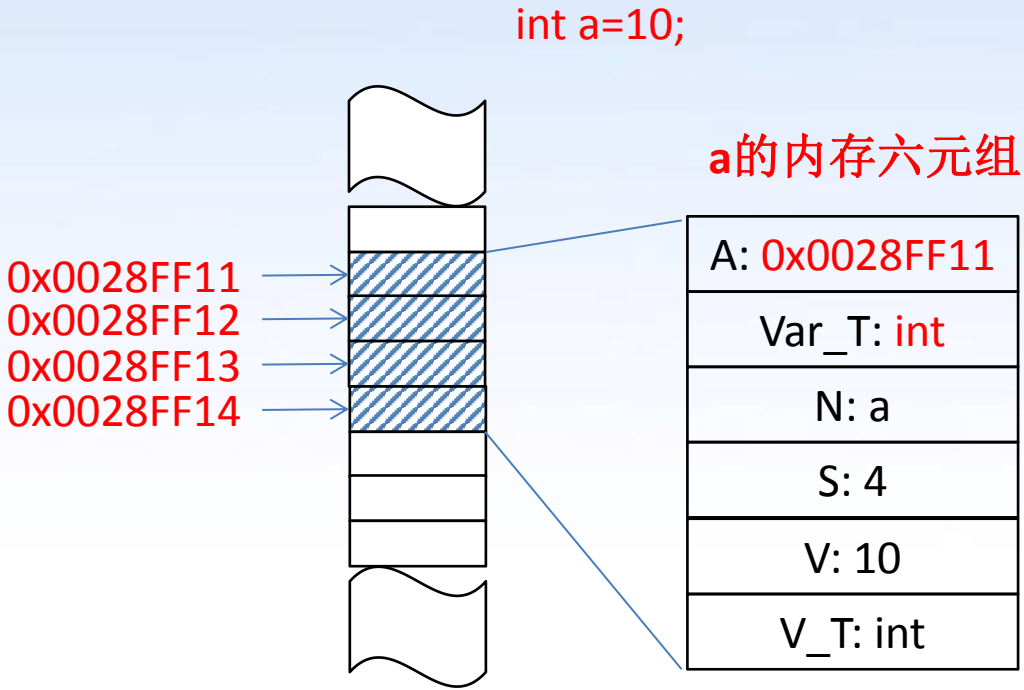
观察下面三个语句

&a;	内存首地址
sizeof(a);	内存大小
a;	内存表示值

首先通过变量名定位a所在内存，然后三个内存相关信息获取操作按优先级：

1、&a
2、sizeof(a)
3、a

观察：int a=10；&a返回值是什么？



对于&a：

1、识别a所在内存

2、发现前面有&

3、返回值规则如下：

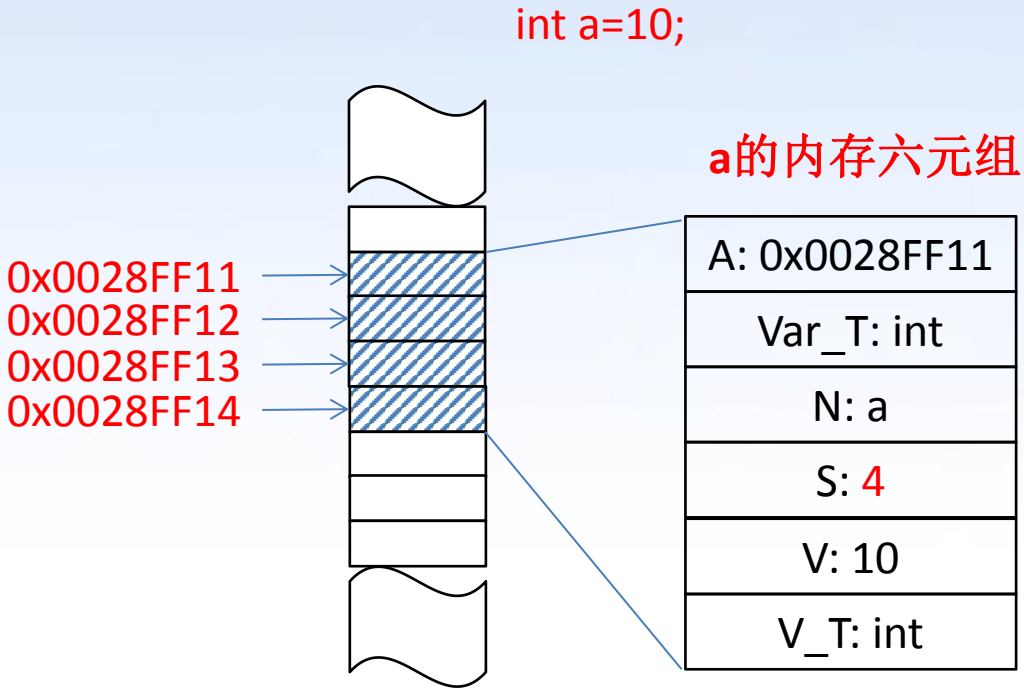
返回值：<Address, Var_T*>

Var_T*是指向Var_T的指针类型

*的具体位置依语法而定

&a：<0x0028FF11, int*>

观察：int a=10; sizeof(a)返回值是什么？

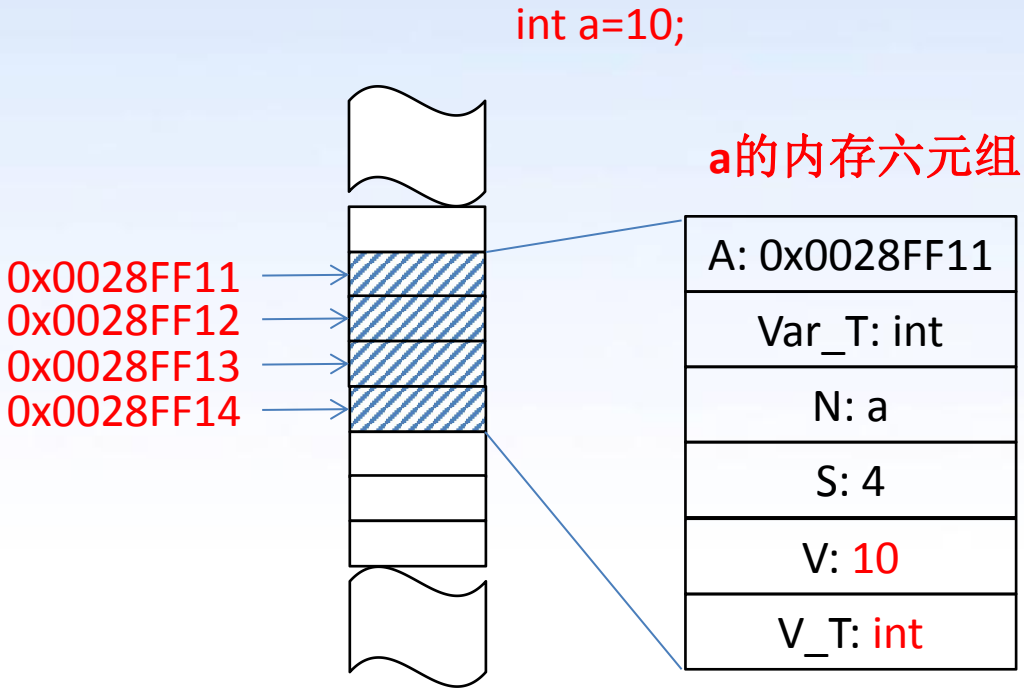


对于sizeof(a)：
1、识别a所在内存
2、发现前面没有&
3、发现和sizeof结合一起
3、返回值规则如下：

返回值：<Size, size_t>
size_t是内存大小的数据类型

sizeof(a)：<4, size_t>

观察：int a=10；a返回值是什么？



对于a：

1、识别a所在内存

2、发现前面没有&

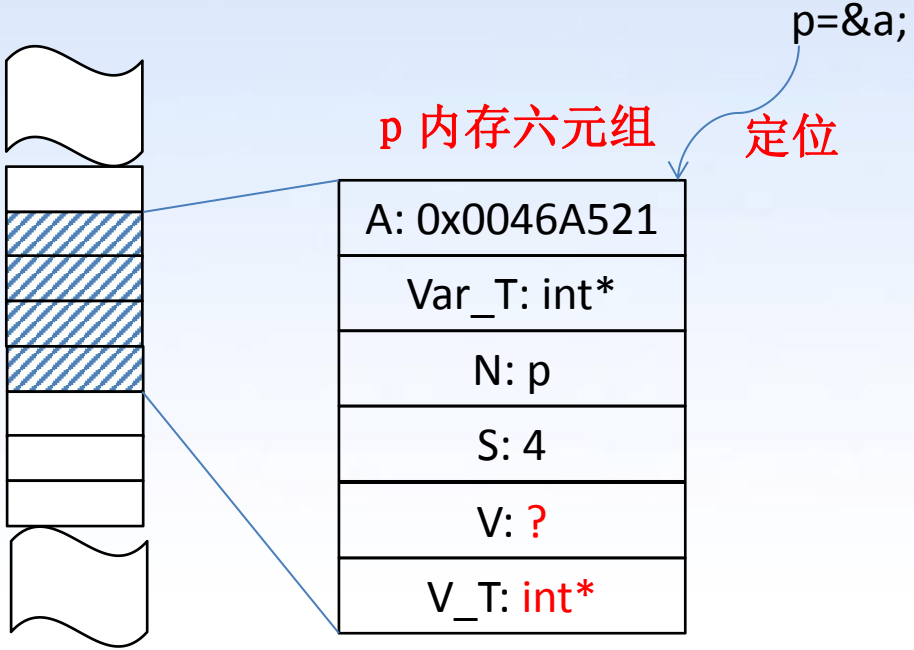
3、发现没有和sizeof结合一起

3、返回值规则如下：

返回值：<Value, Value_Type>

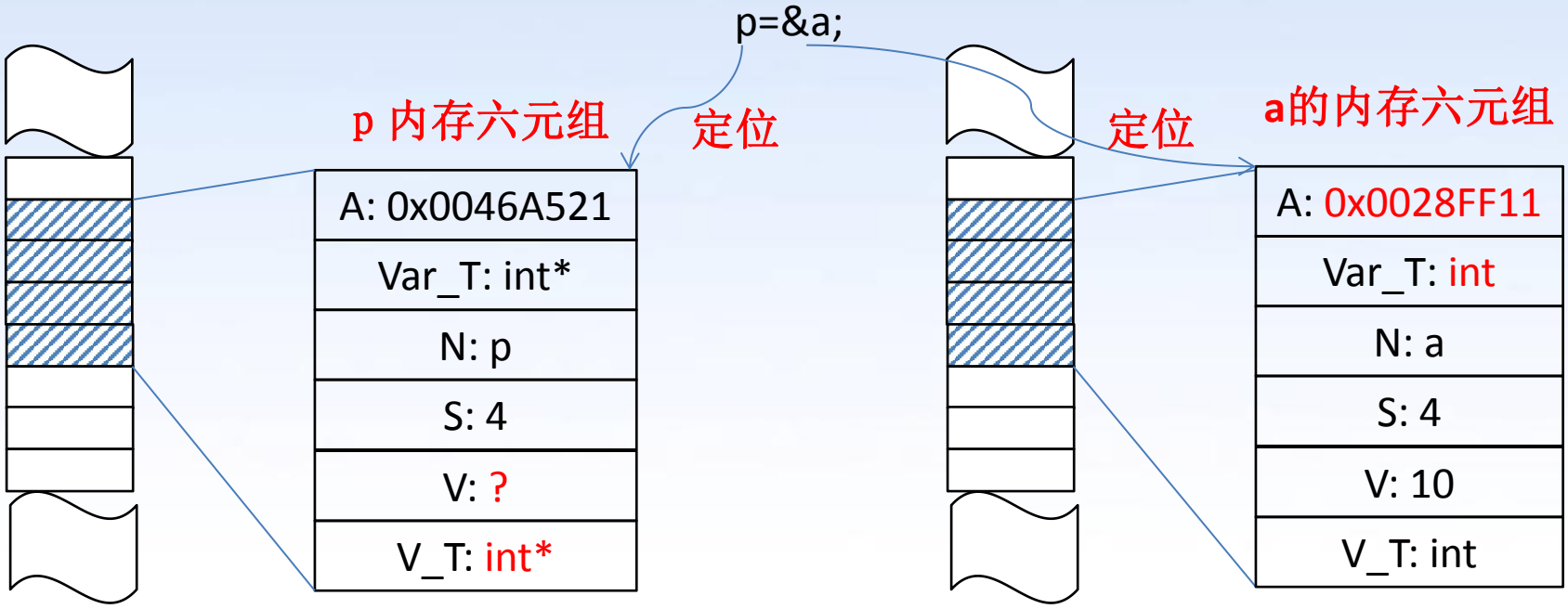
a: <10, int>

观察：int* p=&a发生了什么？

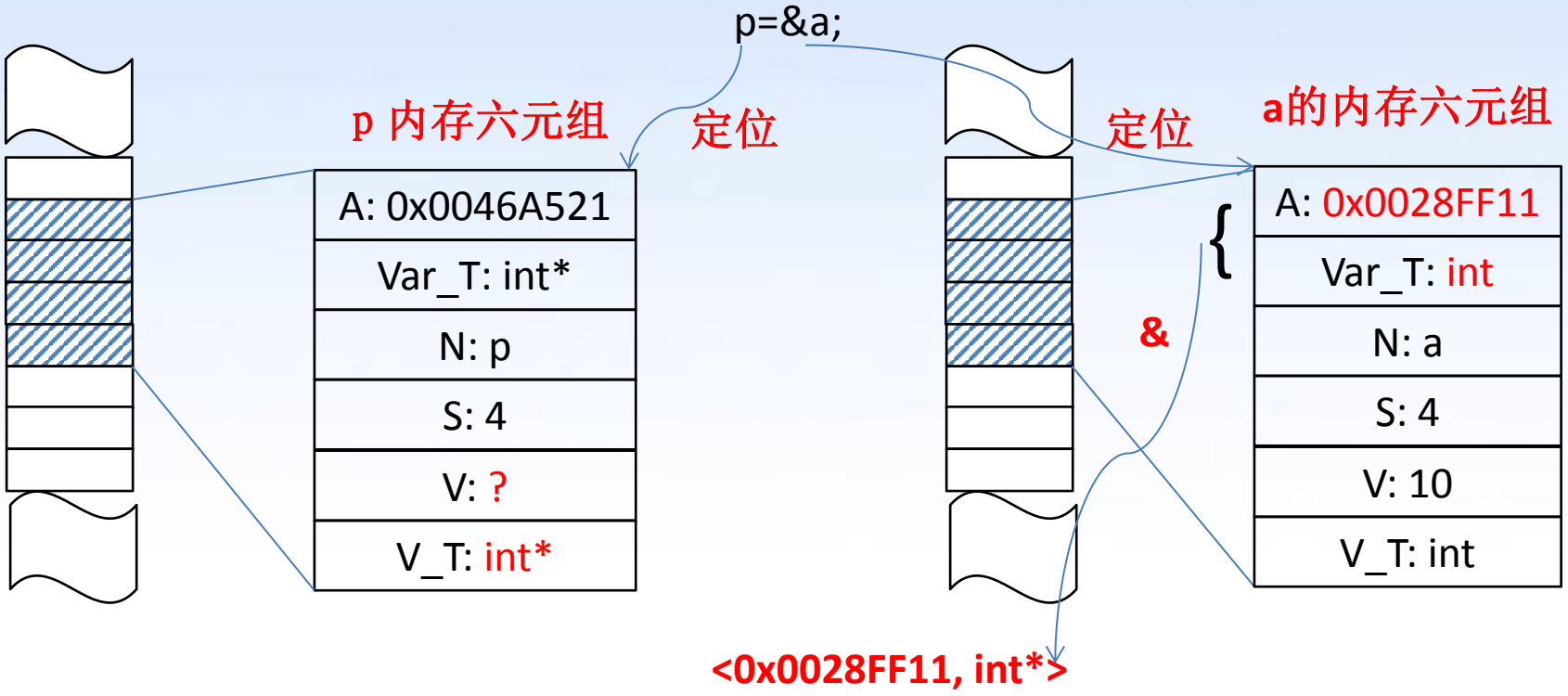


思考：为什么Value=?

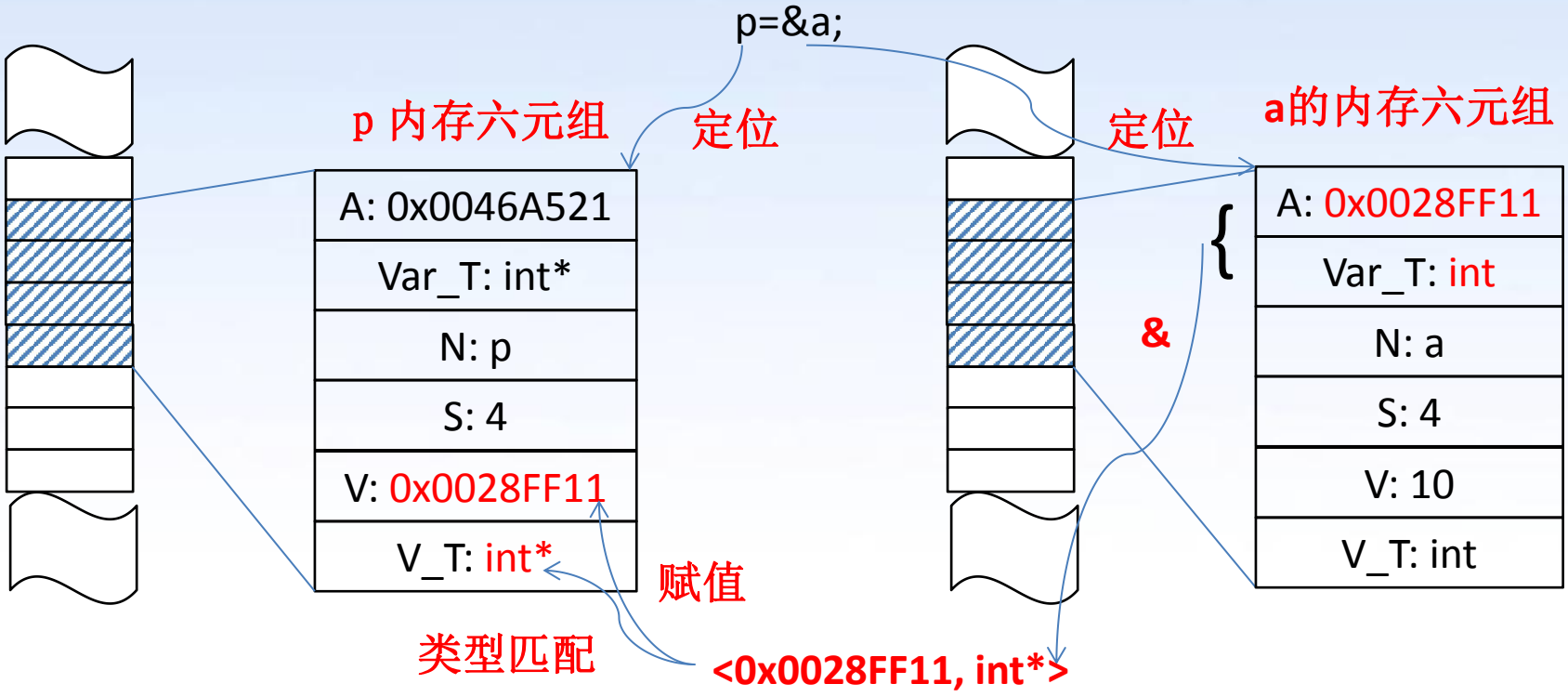
观察：int* p=&a发生了什么？



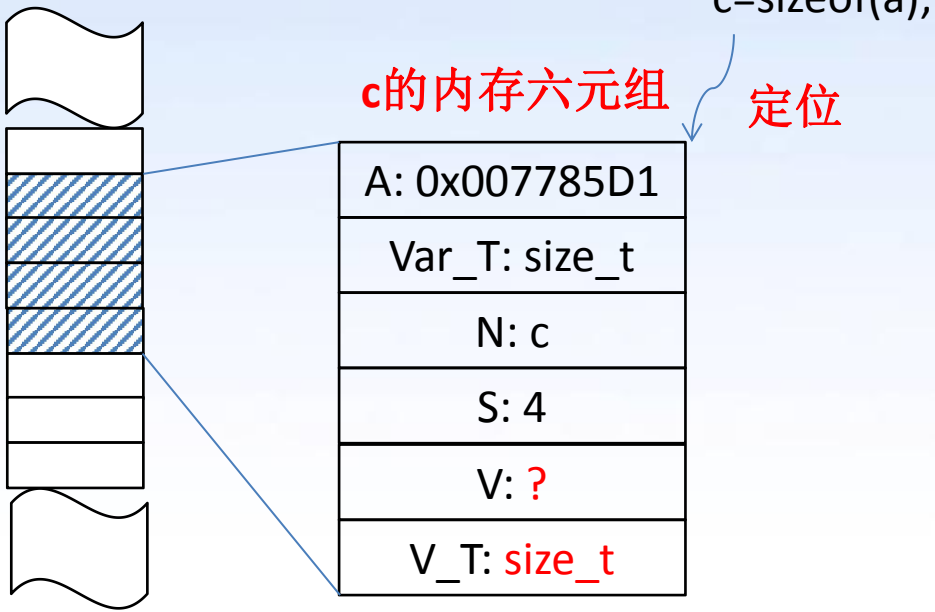
观察：int* p=&a发生了什么？



观察：int* p=&a发生了什么？

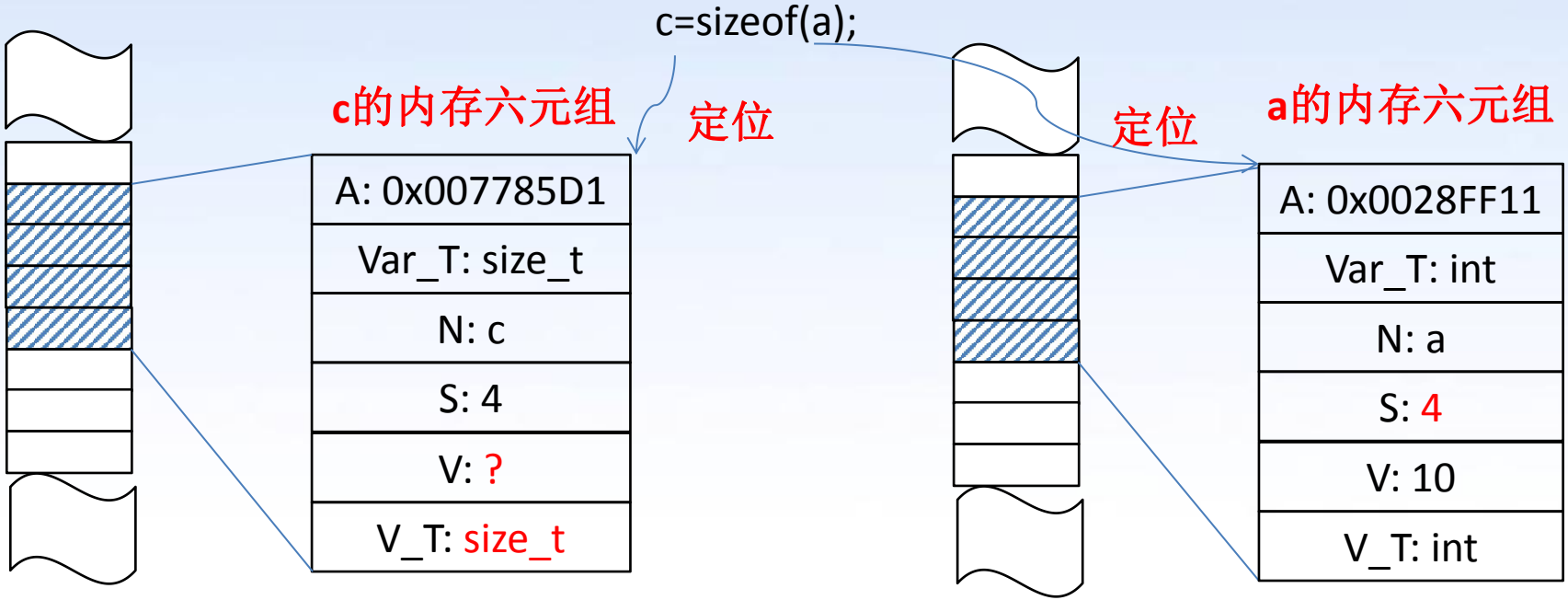


size_t c=sizeof(a)发生了什么?

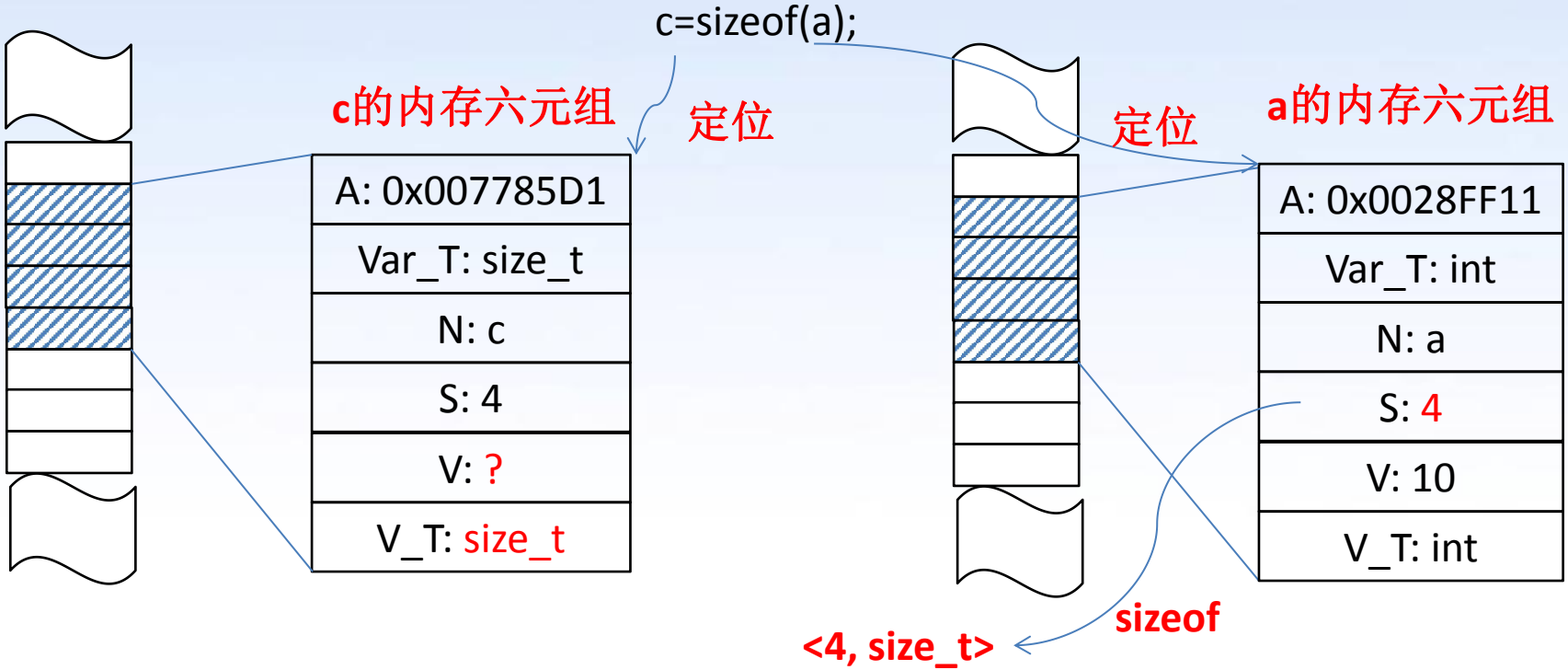


思考：为什么Value=?

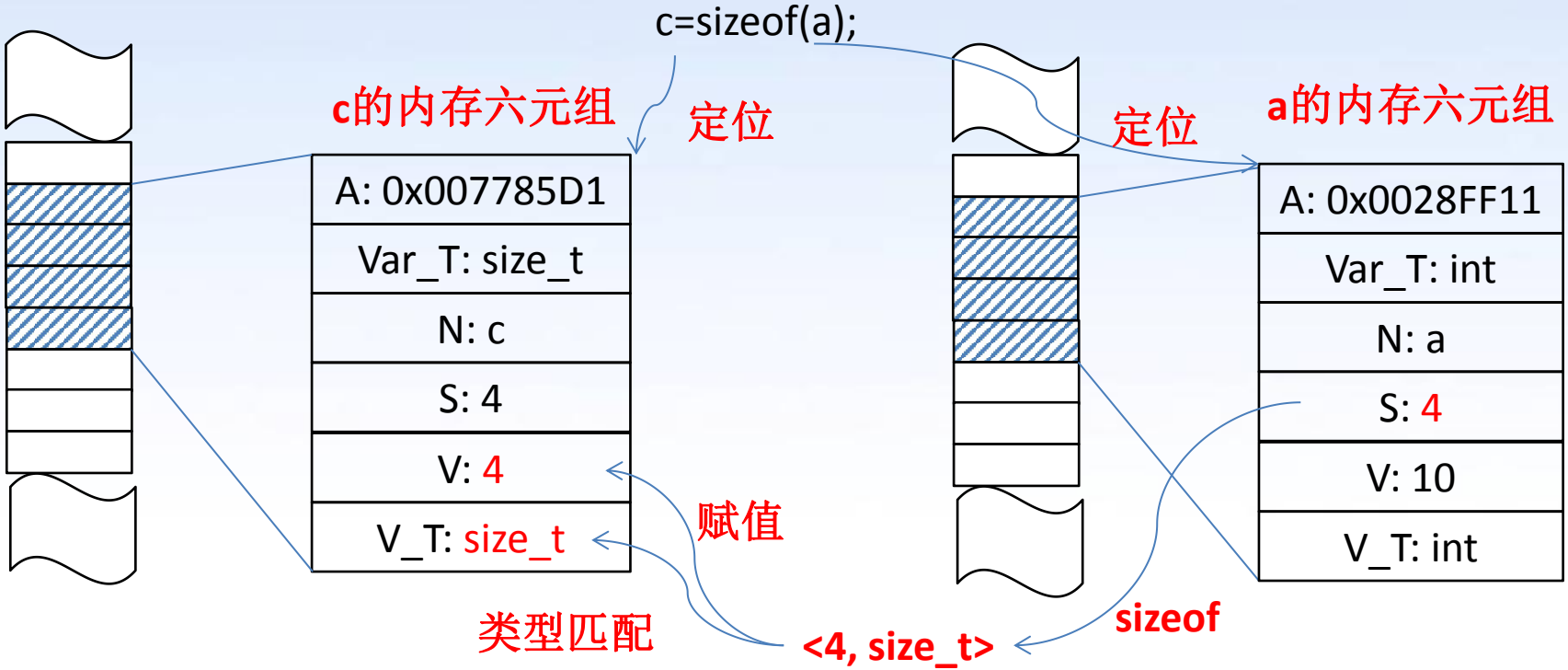
size_t c=sizeof(a)发生了什么?



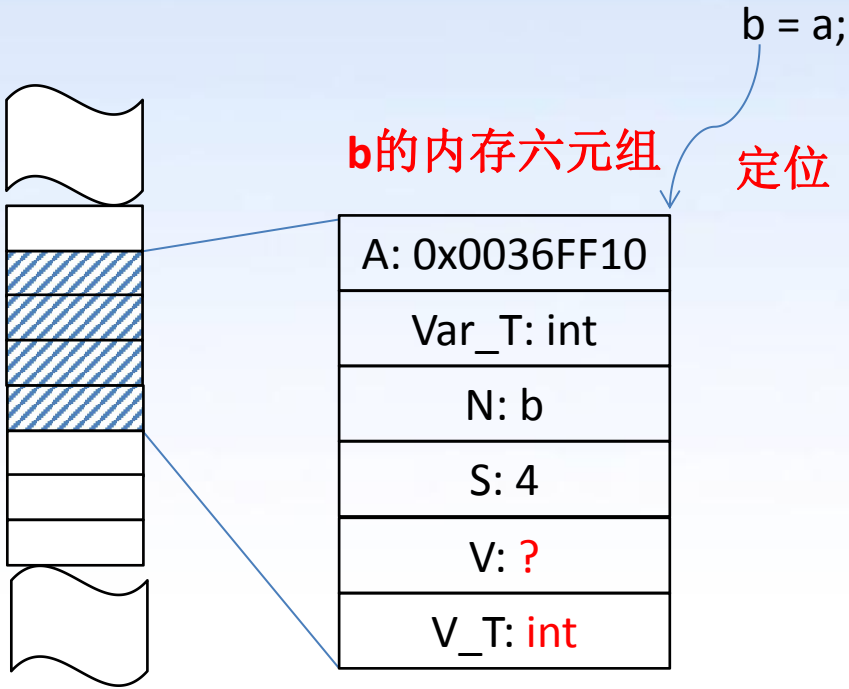
size_t c=sizeof(a)发生了什么?



size_t c=sizeof(a)发生了什么?

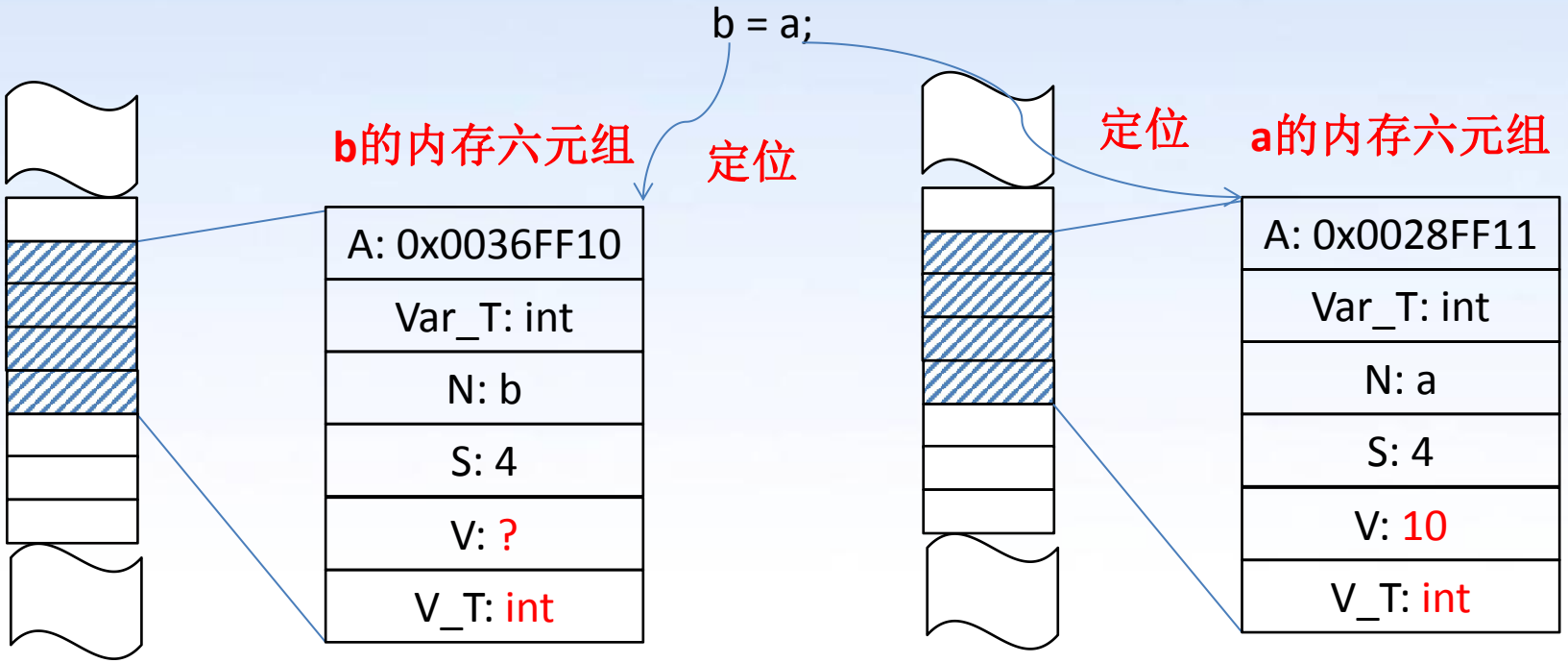


int b=a时发生了什么？

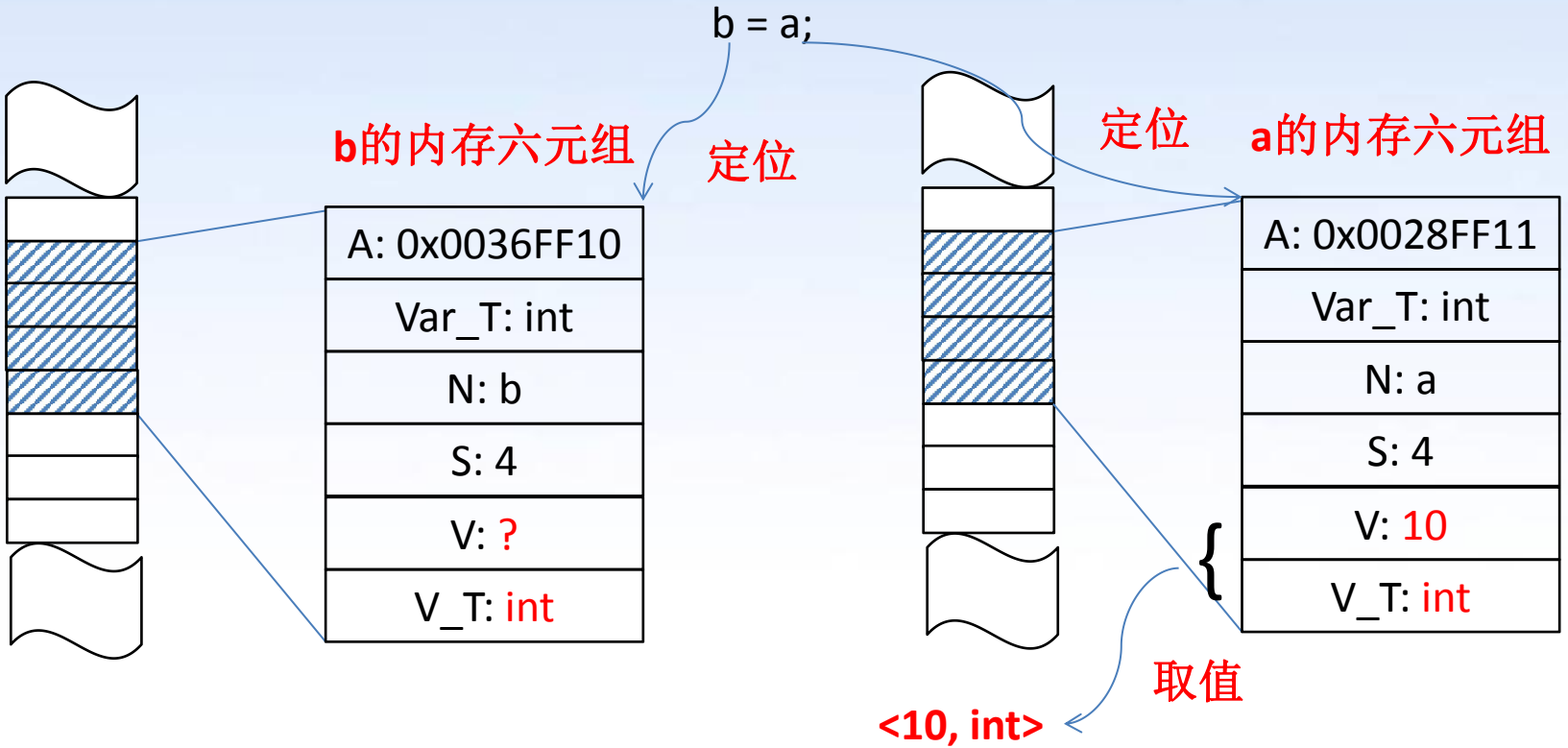


思考：为什么Value=?

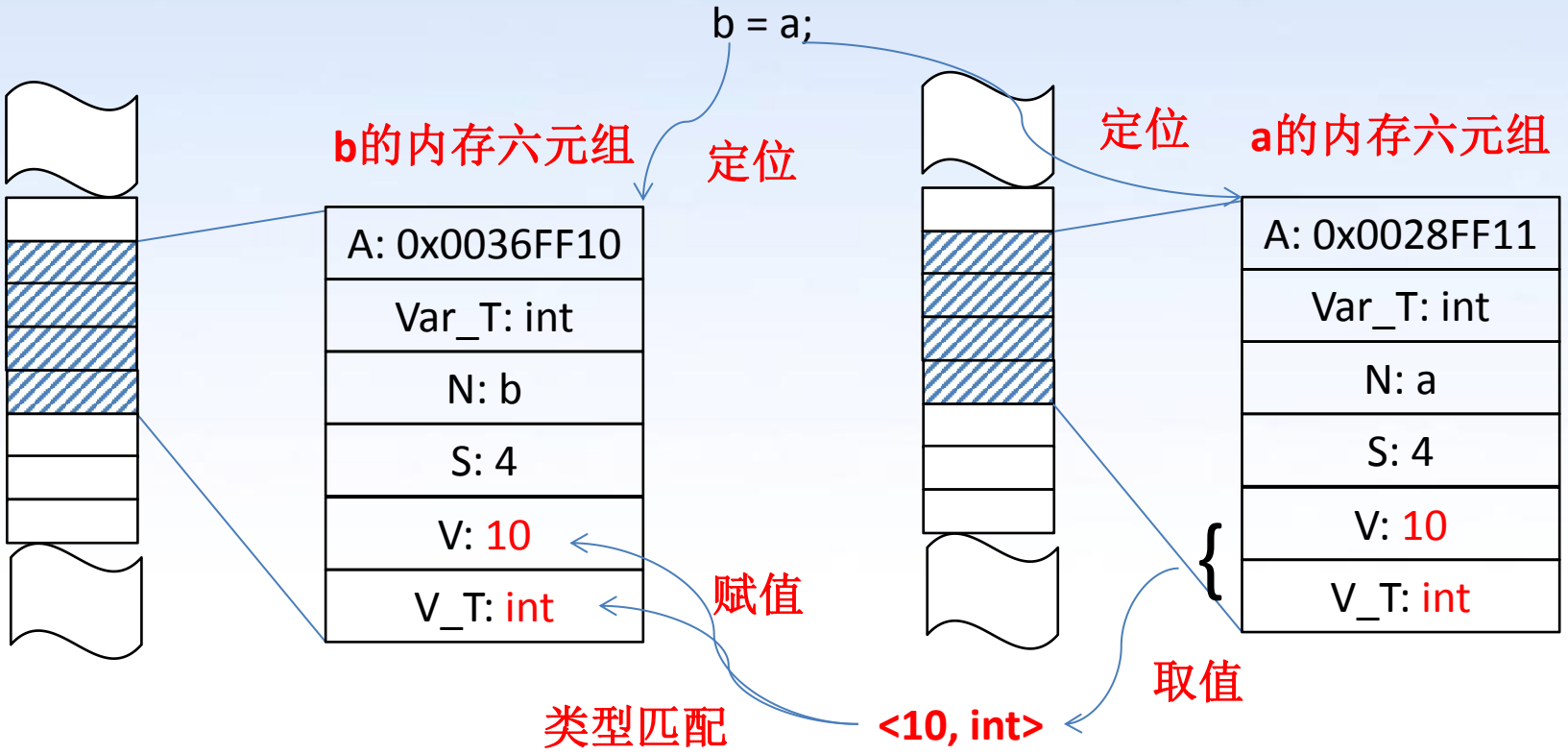
int b=a时发生了什么？



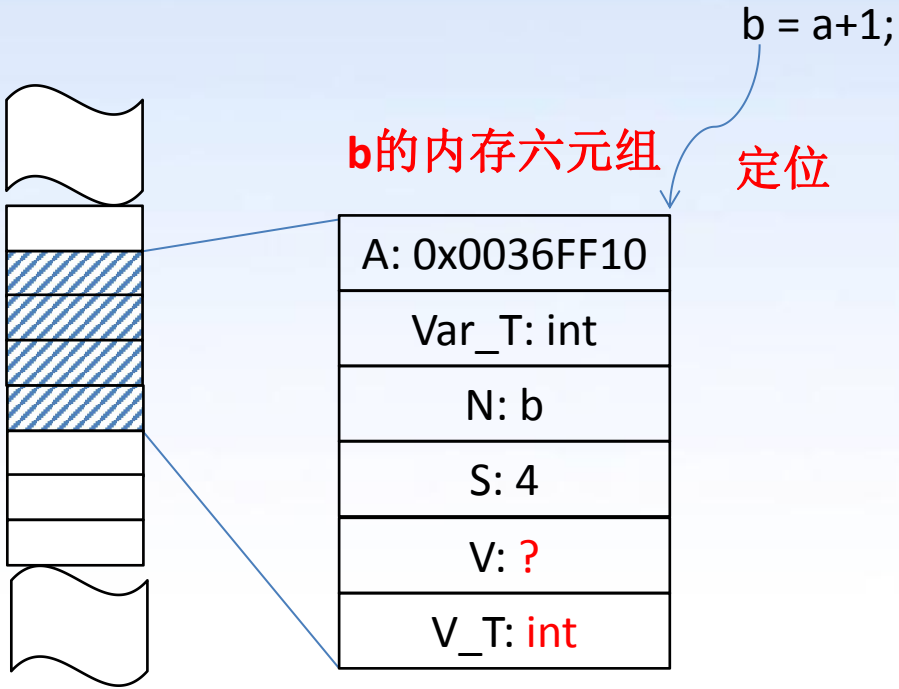
int b=a时发生了什么？



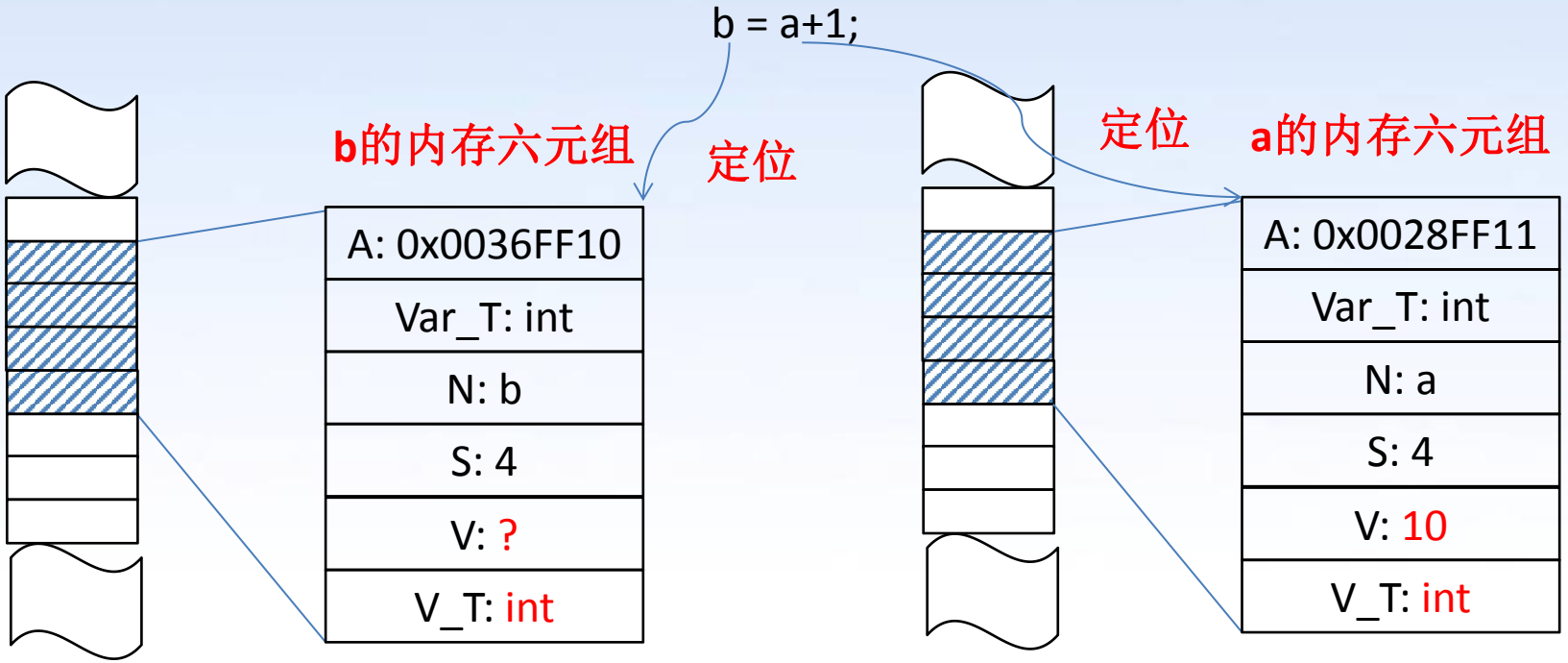
int b=a时发生了什么？



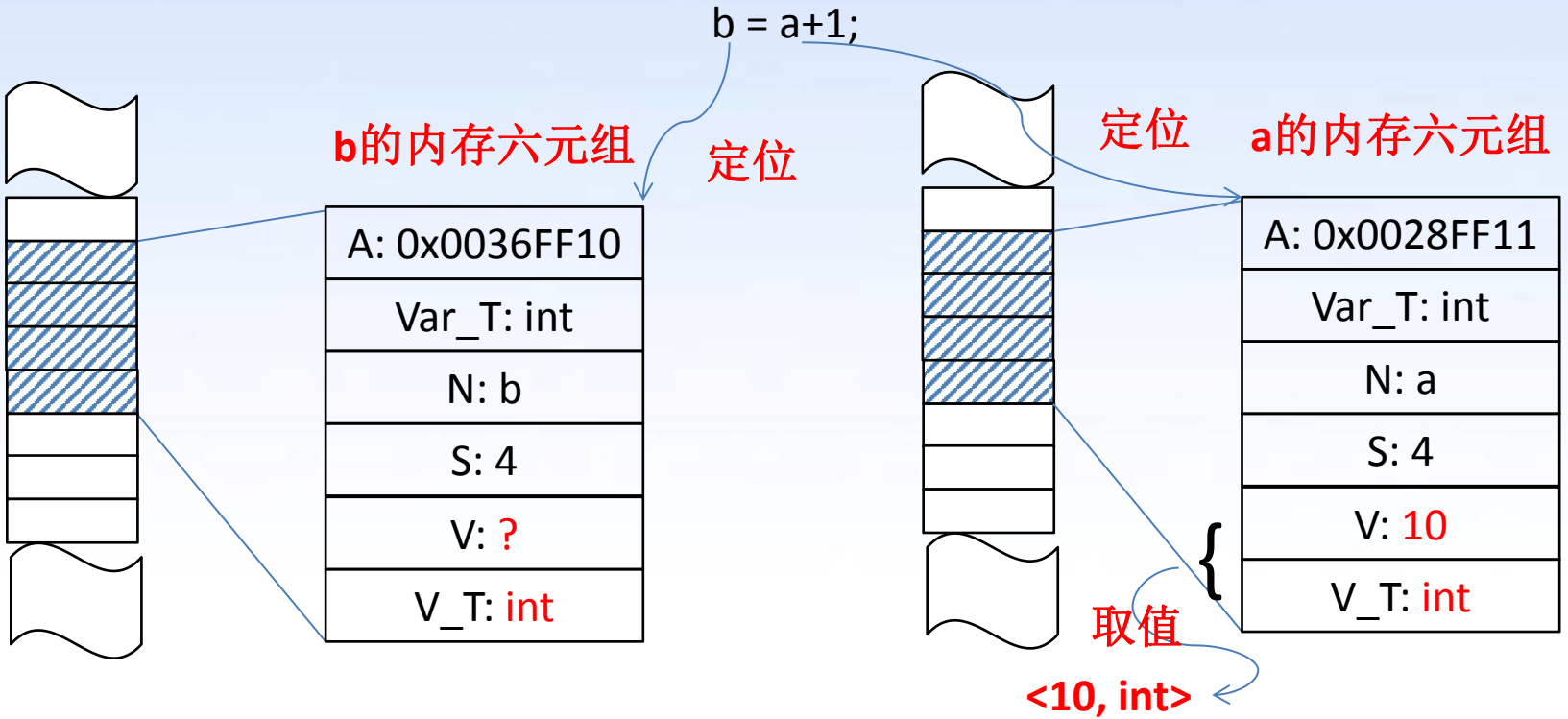
int b=a+1的结果是什么？



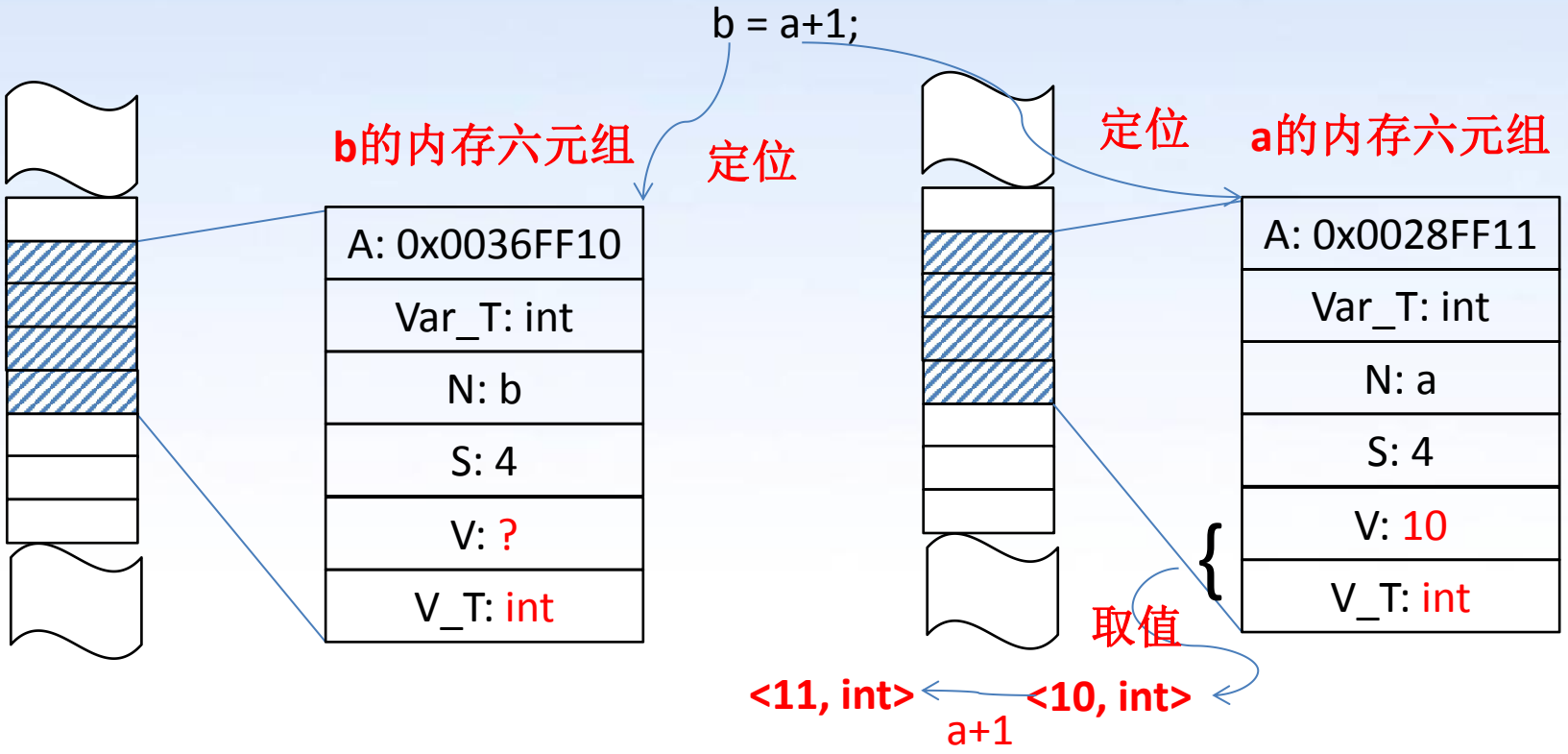
int b=a+1的结果是什么？



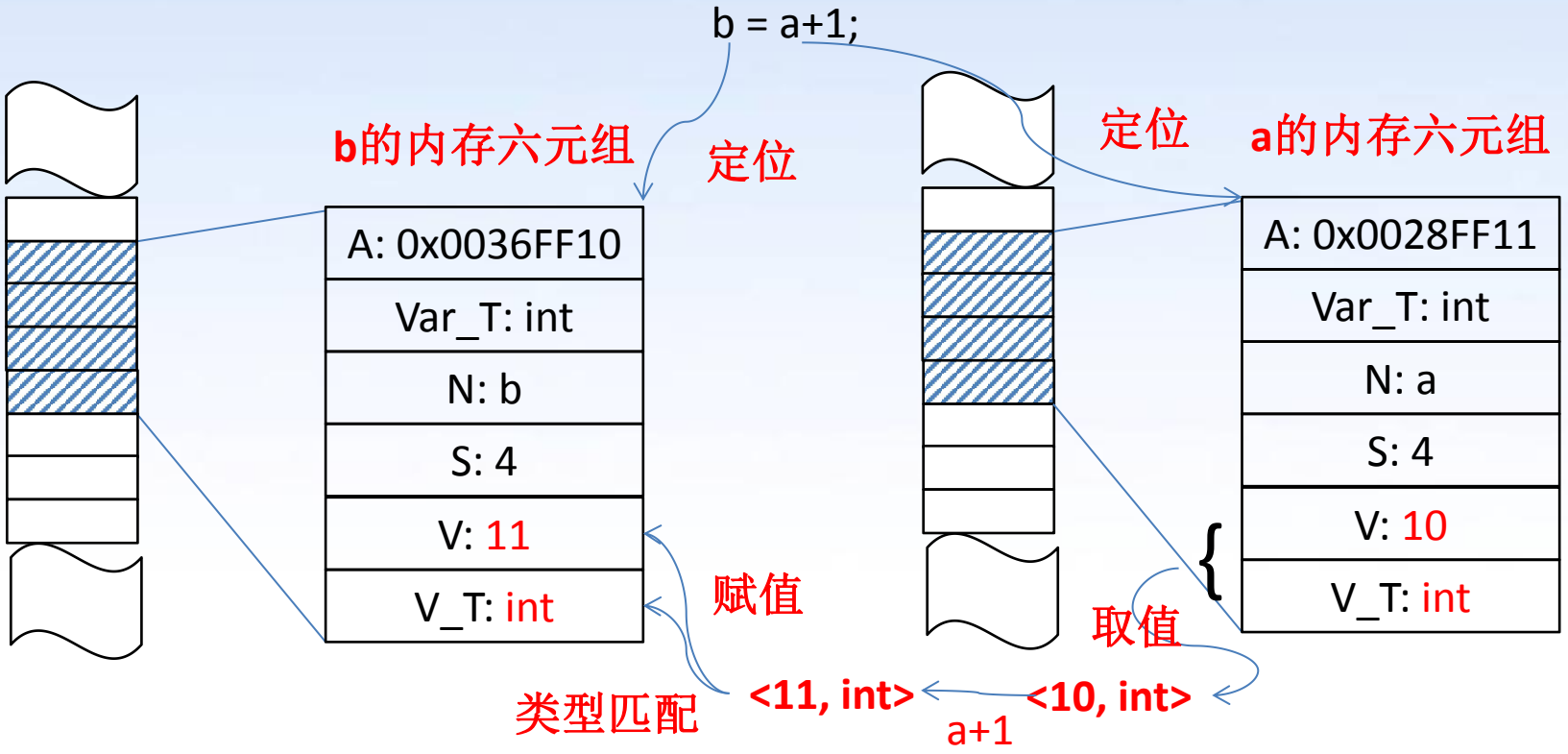
int b=a+1的结果是什么？



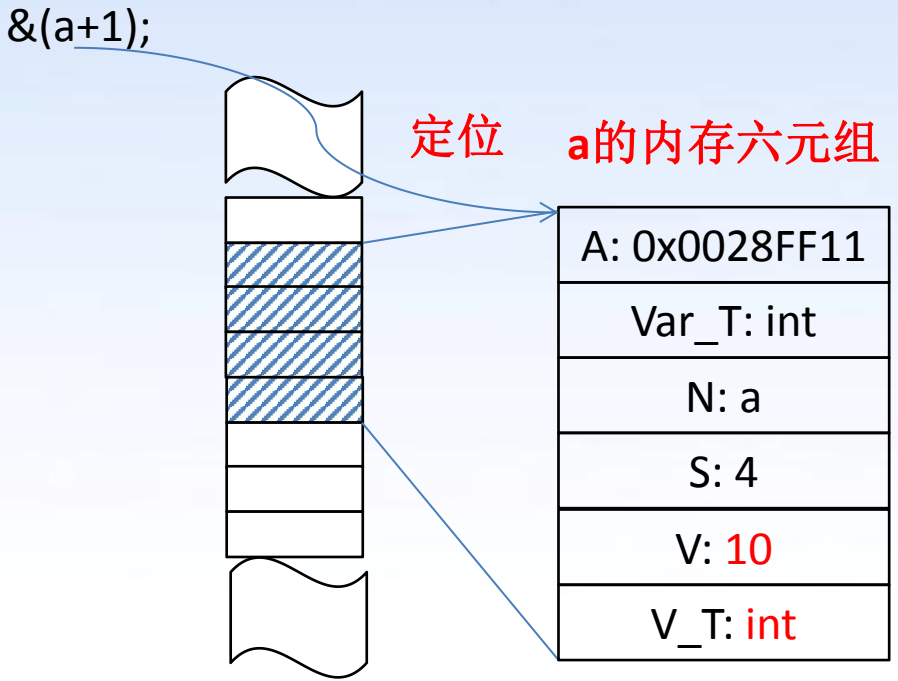
int b=a+1的结果是什么？



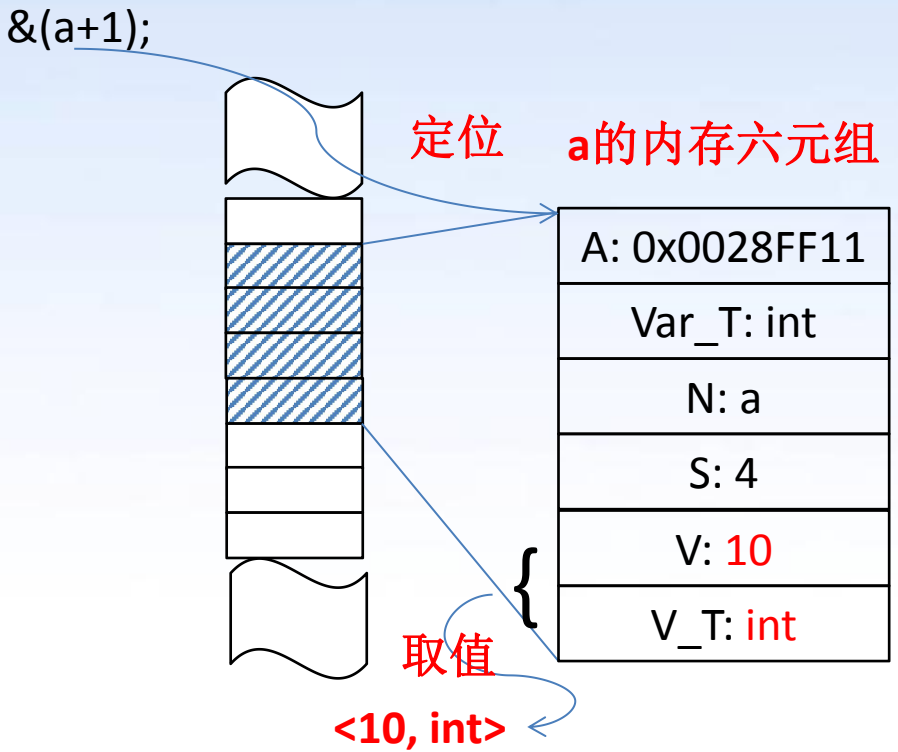
int b=a+1的结果是什么？



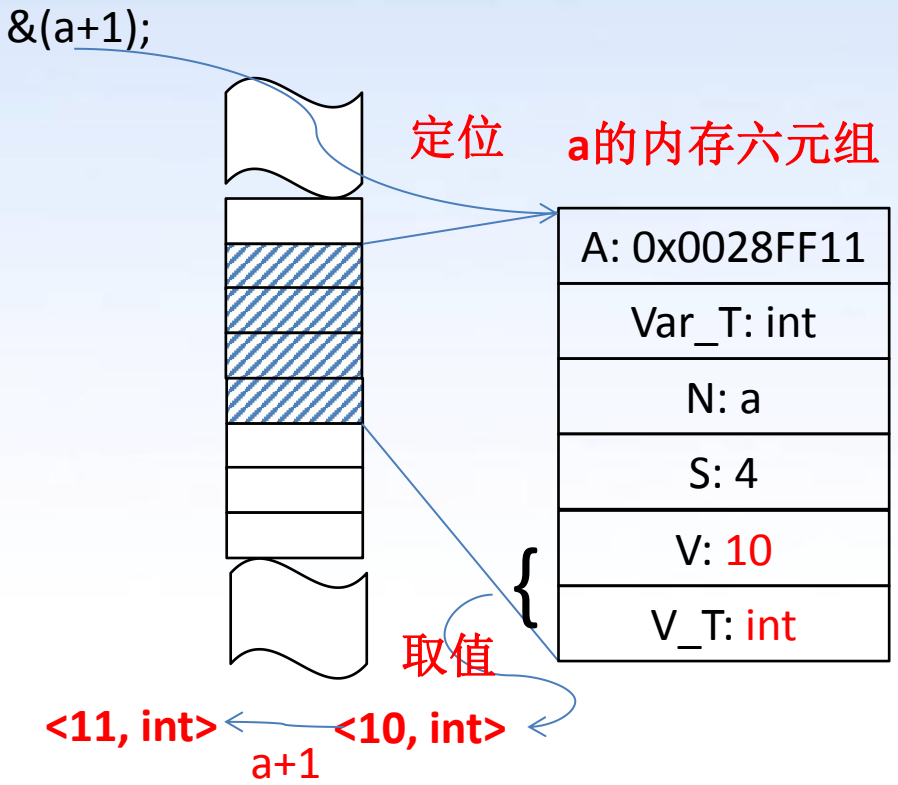
&(a+1)：这个表达式有值吗？



&(a+1)：这个表达式有值吗？



&(a+1)：这个表达式有值吗？


&(a+1);
定位
a的内存六元组
A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int
取值
<11, int>
<10, int>
a+1

&(a+1)：这个表达式有值吗？

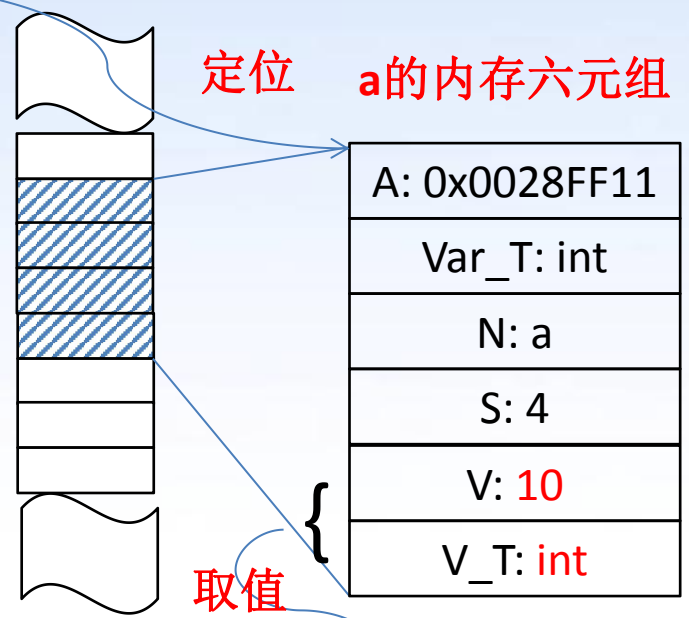
&(a+1);

只能对一块内存进行&操作

×

定位

a的内存六元组



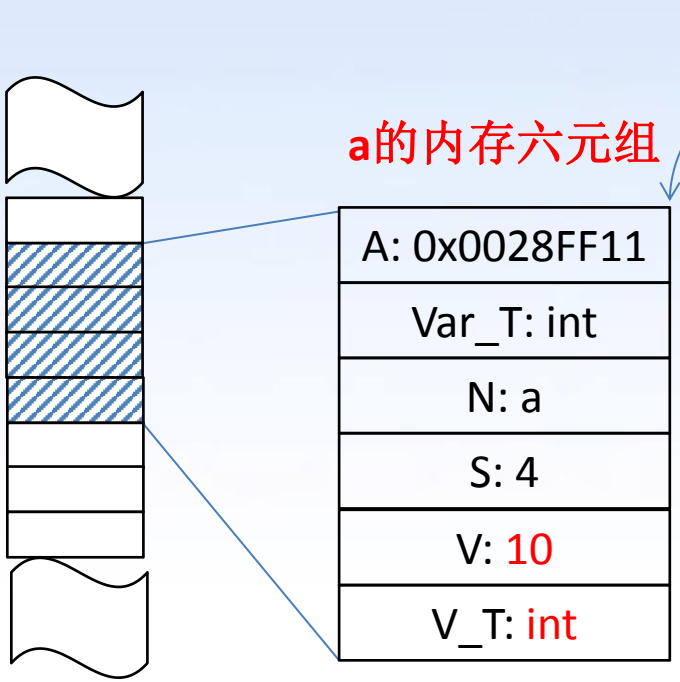
取值

<11, int>

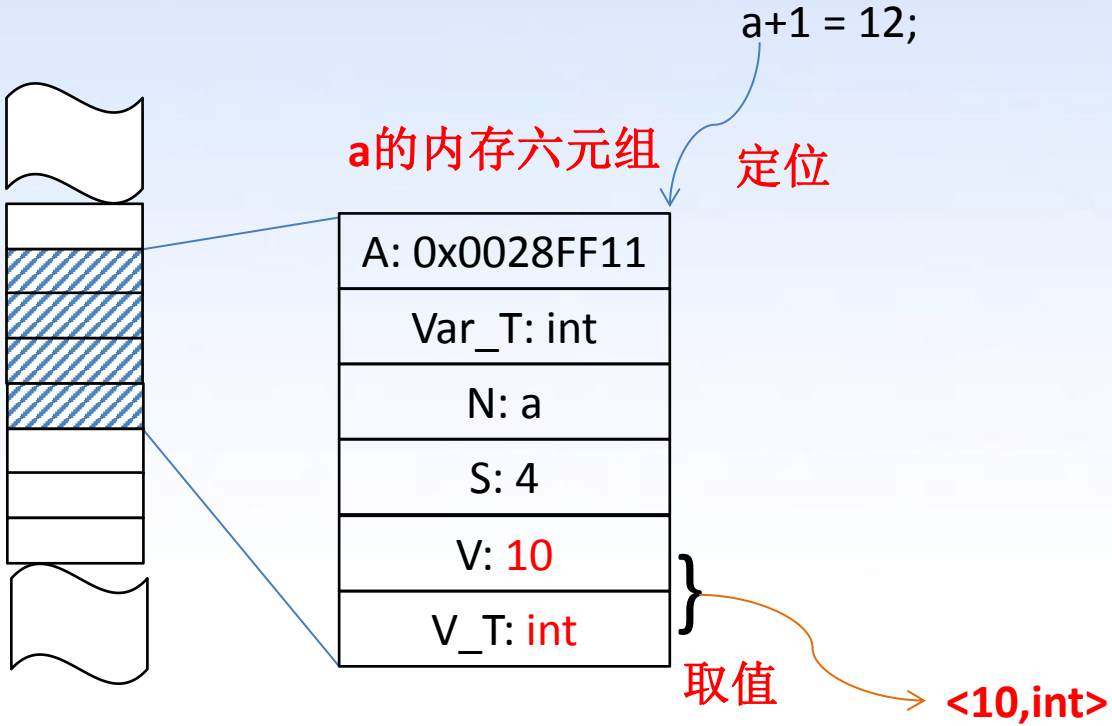
<10, int>

a+1

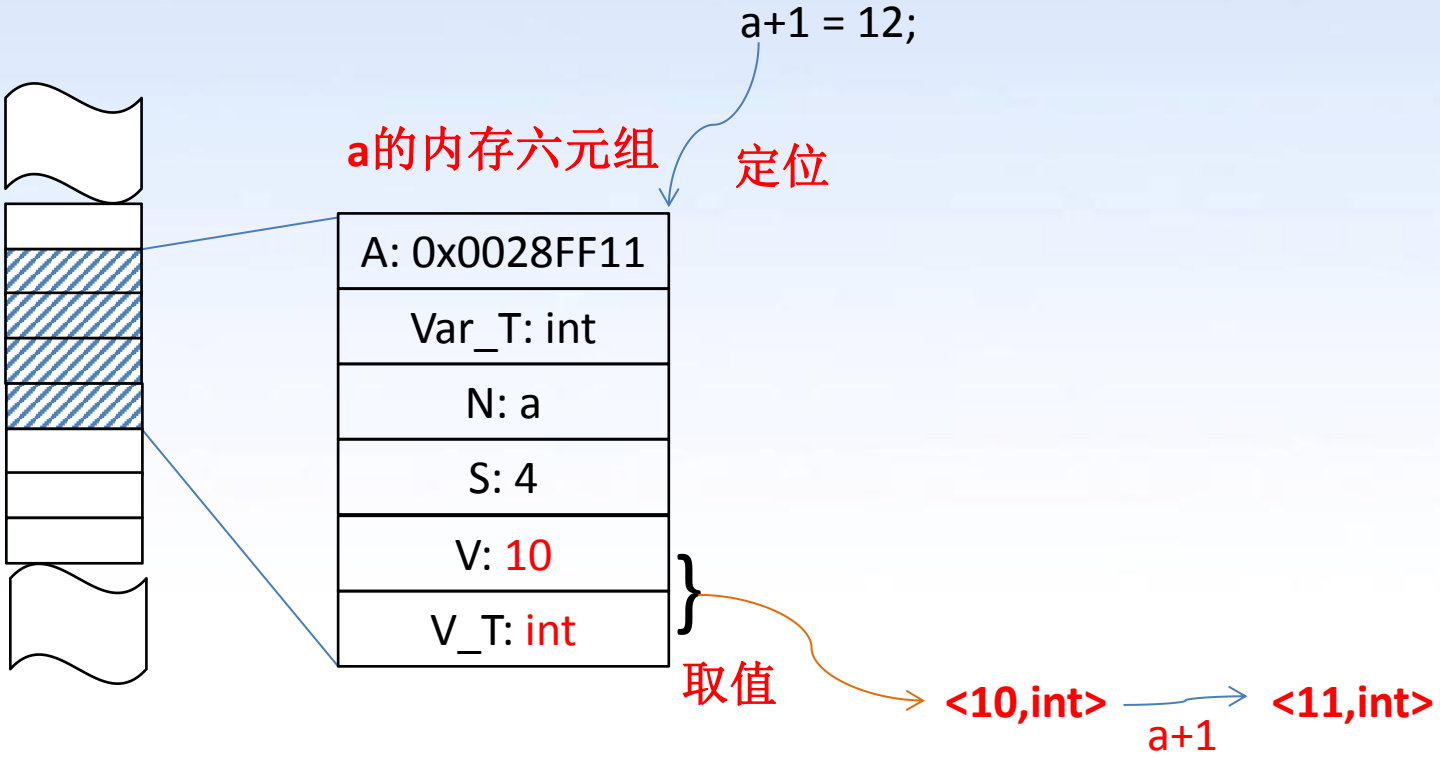
a+1=12会怎么样



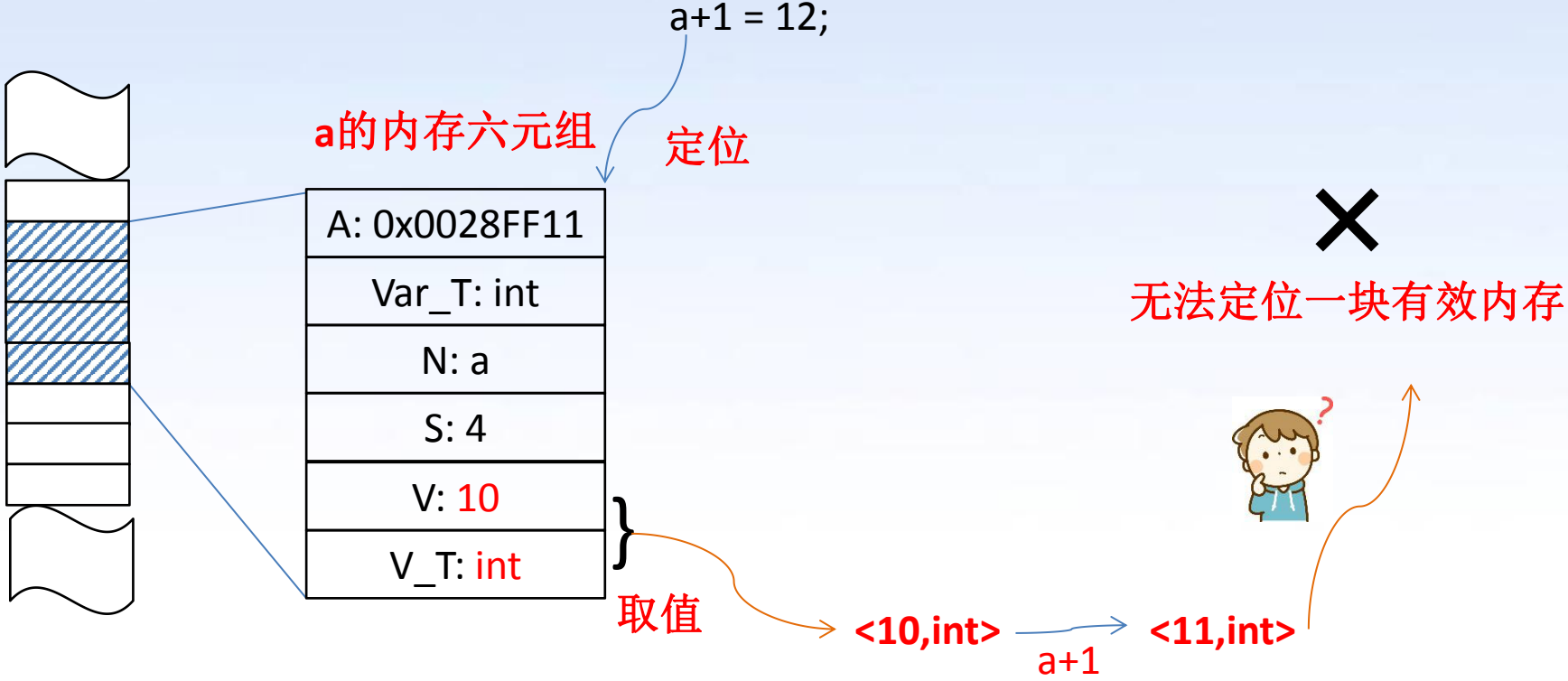
a+1=12会怎么样


a+1 = 12;
a的内存六元组
定位
A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int
取值
<10,int>

a+1=12会怎么样


a+1 = 12;
a的内存六元组
定位
A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int
取值
<10,int>
a+1
<11,int>

a+1=12会怎么样


a+1 = 12;
a的内存六元组
定位
A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int
取值
<10,int>
a+1
<11,int>
无法定位一块有效内存

进一步思考内存在左值时取值的问题

```
a = 12;
```

```
a+1 = 12;
```

```
error: lvalue required as left operand of assignment
```

1、左值要求必须是一块有效内存，a=12中，根据变量名a识别出了这块内存，没有结合任何操作符，因此还是一块合法的内存

2、在a+1=12中，因为有操作符存在，因此首先取得a所在内存的表示值<10, int>，<10, int>+<1, int>=<11, int>，左值现在是一个表达式的值，无法定位一块合法的内存

左值中识别出来的内存，一旦跟任何操作符结合，就变成了取值操作

思考题

1、float b;
float c;
&b=&c是否有错？

思考题

1、float b;
float c;
&b=&c是否有错？

答案：假设变量b和变量c所在内存首地址分别是0x00FCBB11和0x00FFAB11

1、左值发现变量名b，识别b对应的内存，和&操作符结合，得到结果<0x00FCBB11, float*>，这是一个表达式的值，不是一个合法内存

error: lvalue required as left operand of assignment

思考题

1、float* b;
float c;
b=&c是否有错？

思考题

1、float* b;
float c;
b=&c是否有错？

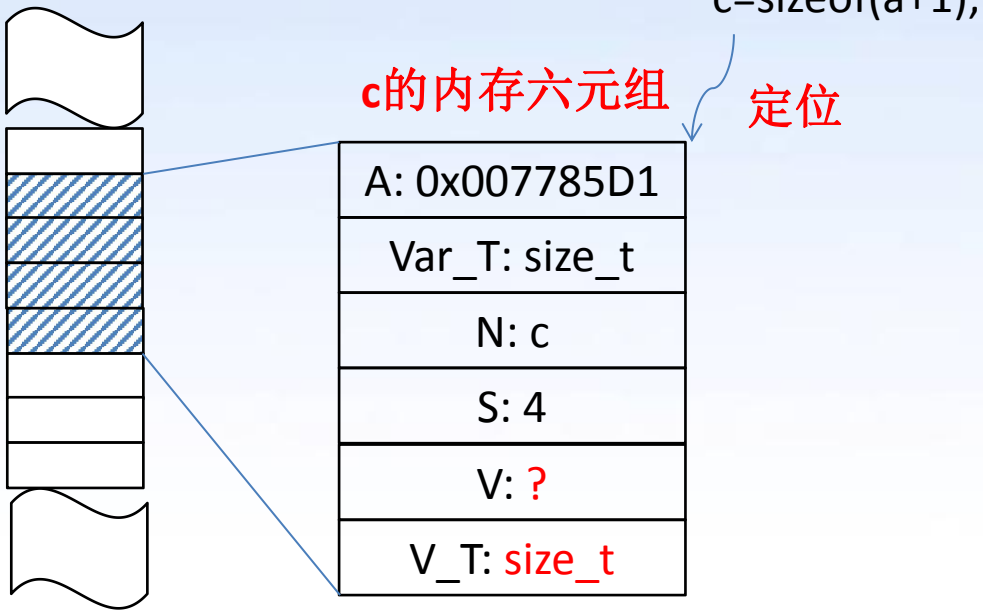
答案：假设变量b和变量c所在内存首地址分别是0x00FCBB11和0x00FFAB11

1、左值发现变量名b，识别b对应的内存，没有和任何操作符结合，是一个合法内存

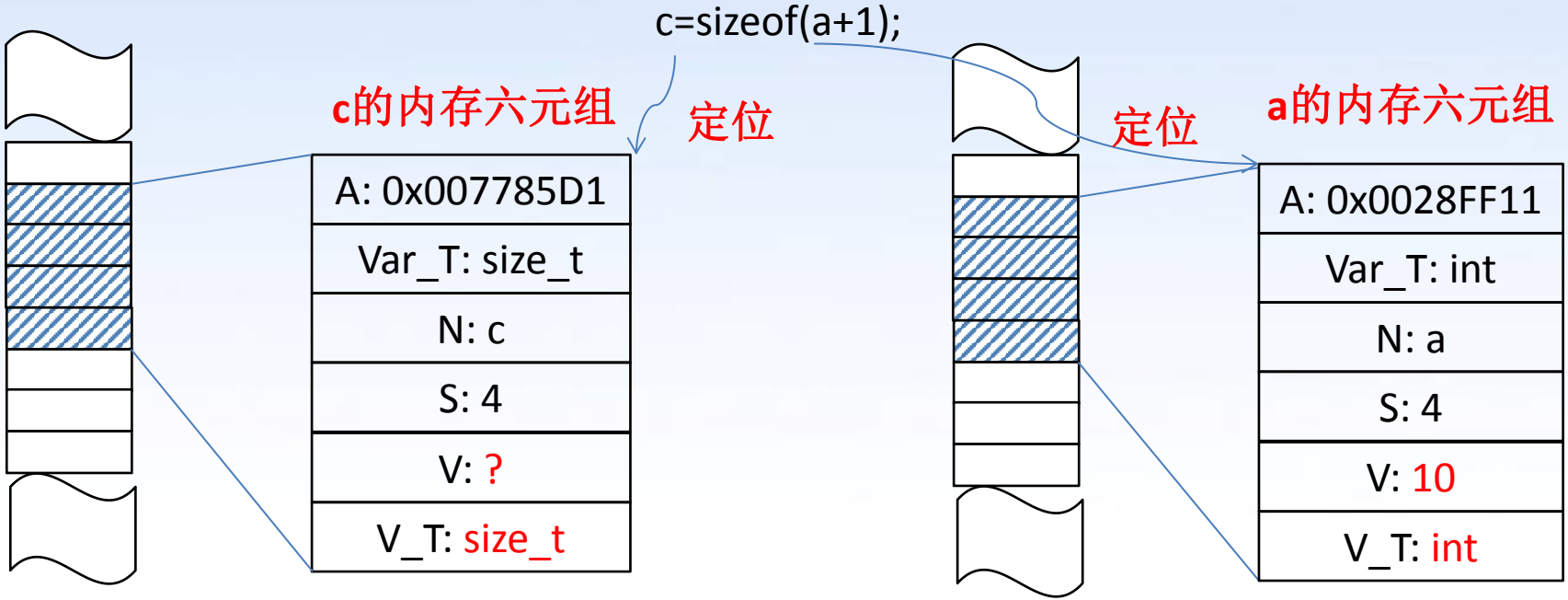
2、右值发现变量名c，识别c对应的内存，和&操作符结合，得到结果<0x00FFAB11, float*>

3、将&c的取值<0x00FFAB11, float*>通过变量赋值赋给变量b对应的内存

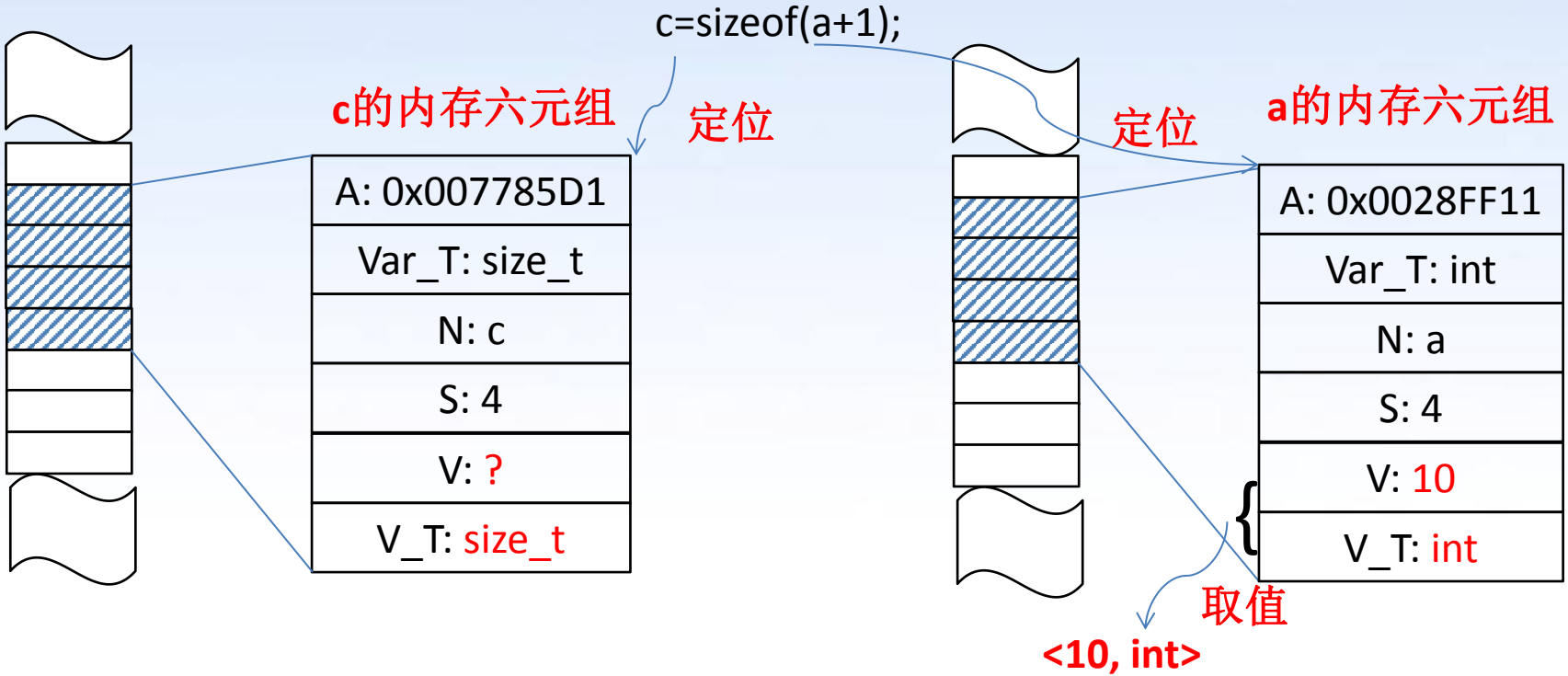
size_t c=sizeof(a+1)发生了什么？



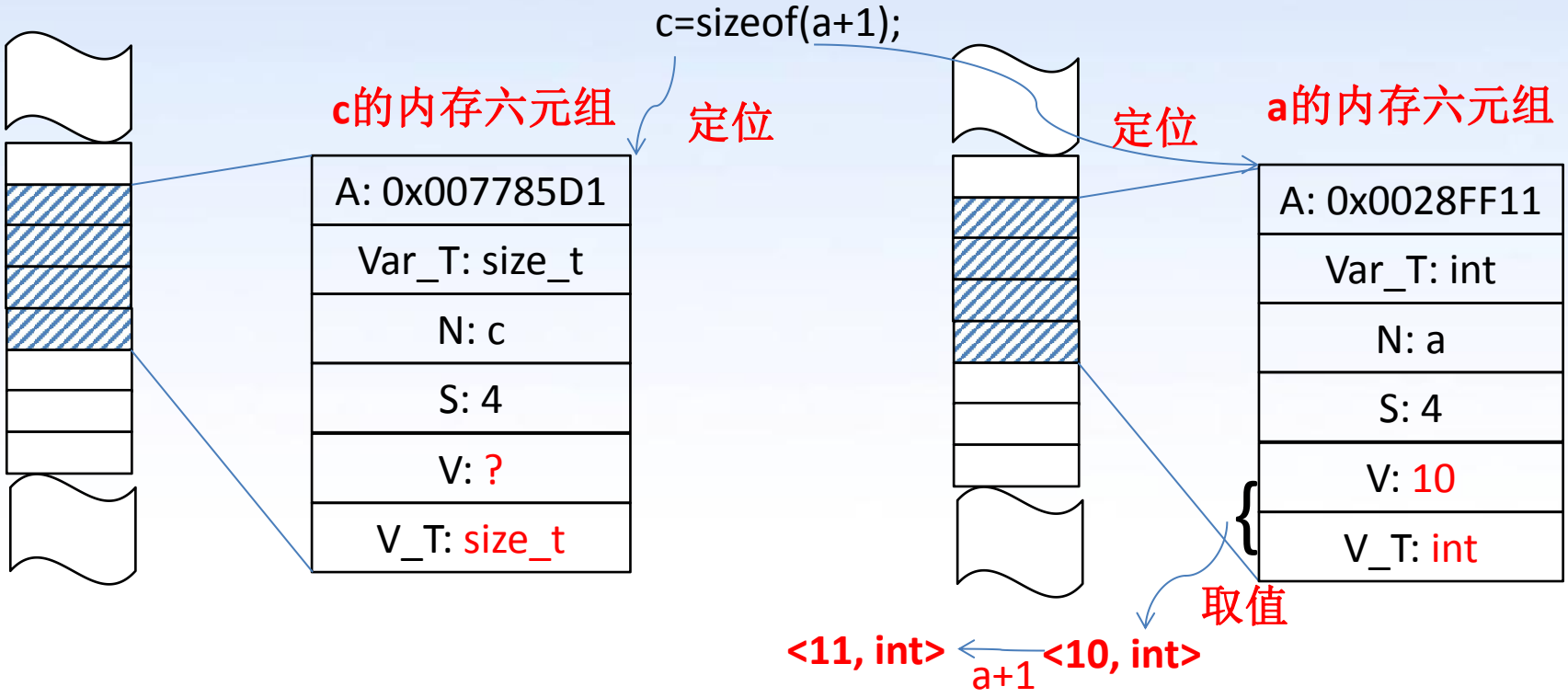
size_t c=sizeof(a+1)发生了什么？



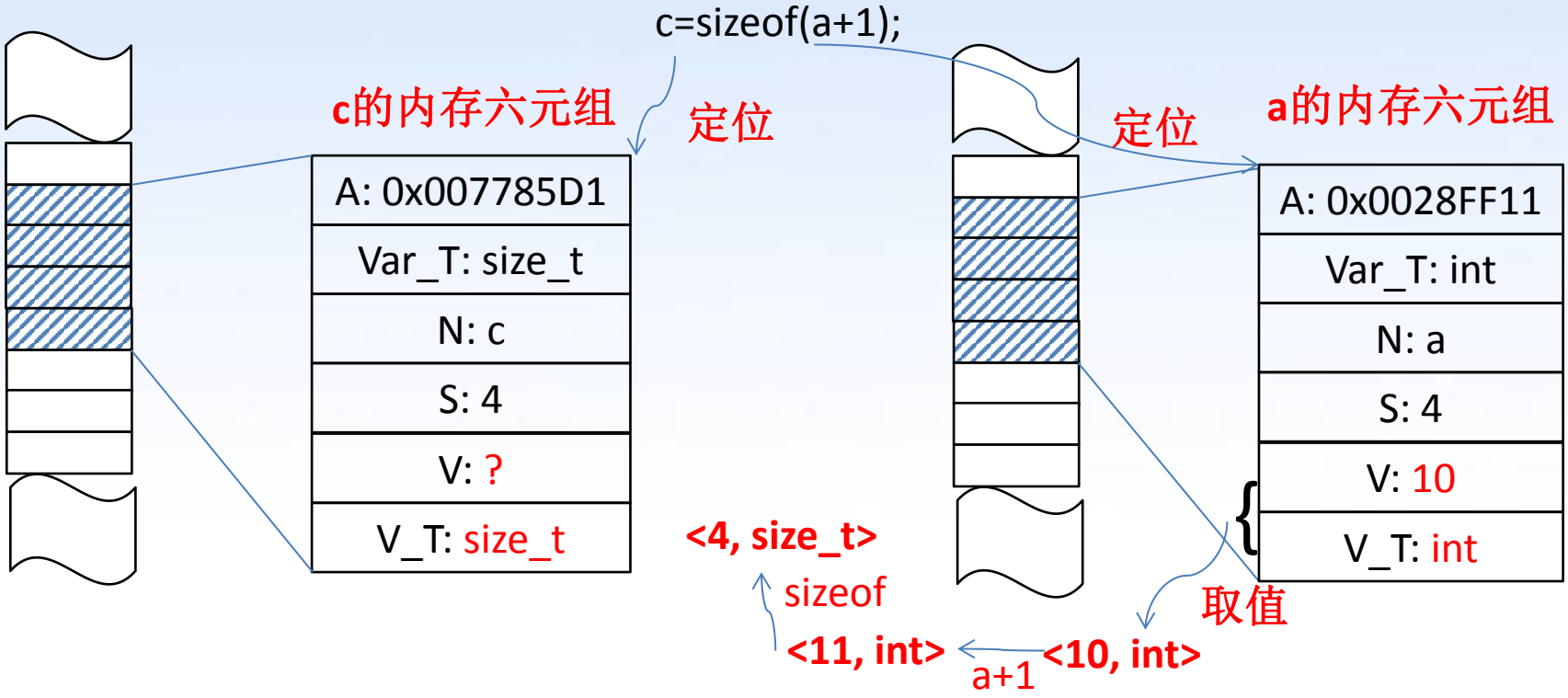
size_t c=sizeof(a+1)发生了什么？



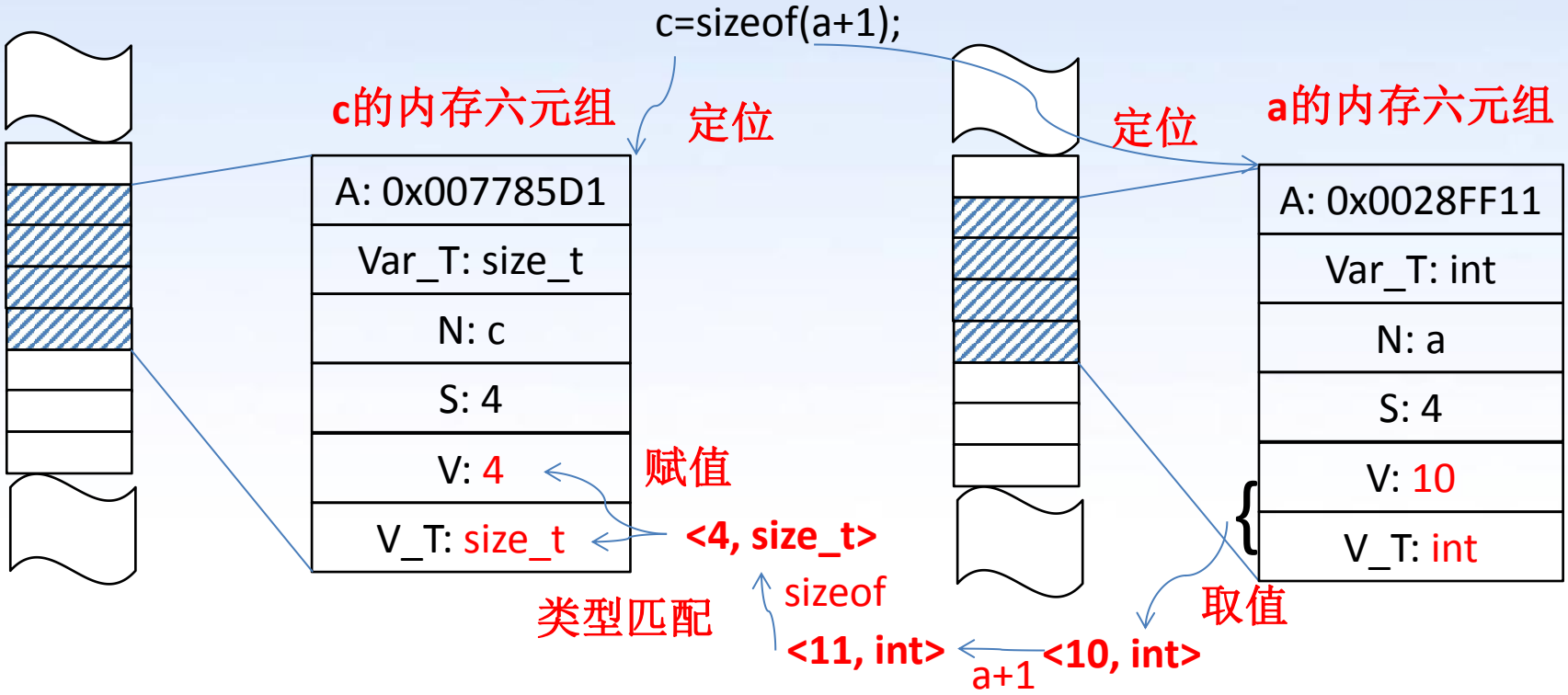
size_t c=sizeof(a+1)发生了什么？



size_t c=sizeof(a+1)发生了什么？



size_t c=sizeof(a+1)发生了什么？



深入了解sizeof()

int a，以下表达式返回值是什么？

sizeof(int) vs. sizeof(a) vs. sizeof(a+1)

假设变量a的表示值为<10, int>，即a=10;

sizeof(内容有三类)

sizeof(int)： 4， 获得一个变量类型的大小
sizeof(a)： 4， 获得一个变量对应的内存的大小
sizeof(a+1)： 4， 获得一个表达式返回值类型的大小
a+1: <11, int>

复习一下：内存如何定位

1、通过变量名定位内存：即通过变量名来定位这段内存，例如：

int a; float b; double c;

通过a，b，c就可以定位到对应的内存

2、通过*运算符来定位内存：假设一个表达式expression的取值为<Value, Value_Type>，如果Value_Type是一个指针类型，则可以用*expression的方式来定位到Value对应字节编号开头的一段内存

我们再来看通过*运算符定位内存

用“*”来间接内存定位

int a=10; int* p=&a;

为什么*p可以间接定位内存？

*p的时候发生了什么？

1、p是一个变量名，定位p的内存

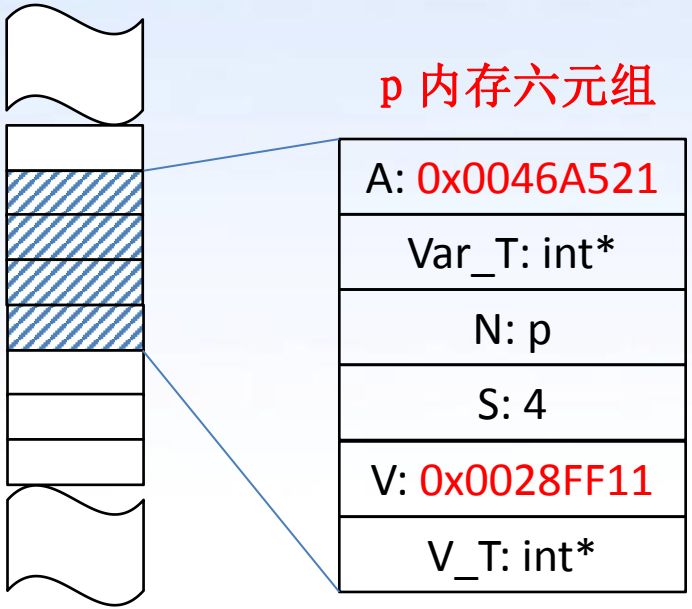
2、p前面没有&和sizeof，获得p的表示值<V, V_T>

3、根据<V, V_T>还原*p内存，规则如下：

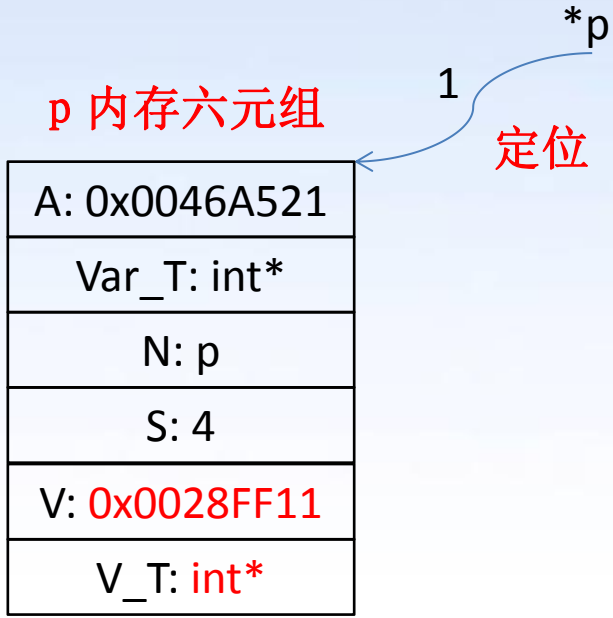
1）V的值是*p对应内存六元组的Address(A)

2）V_T是一个指针类型，其指向的变量类型就是*p内存六元组的Variable_Type(Var_T)

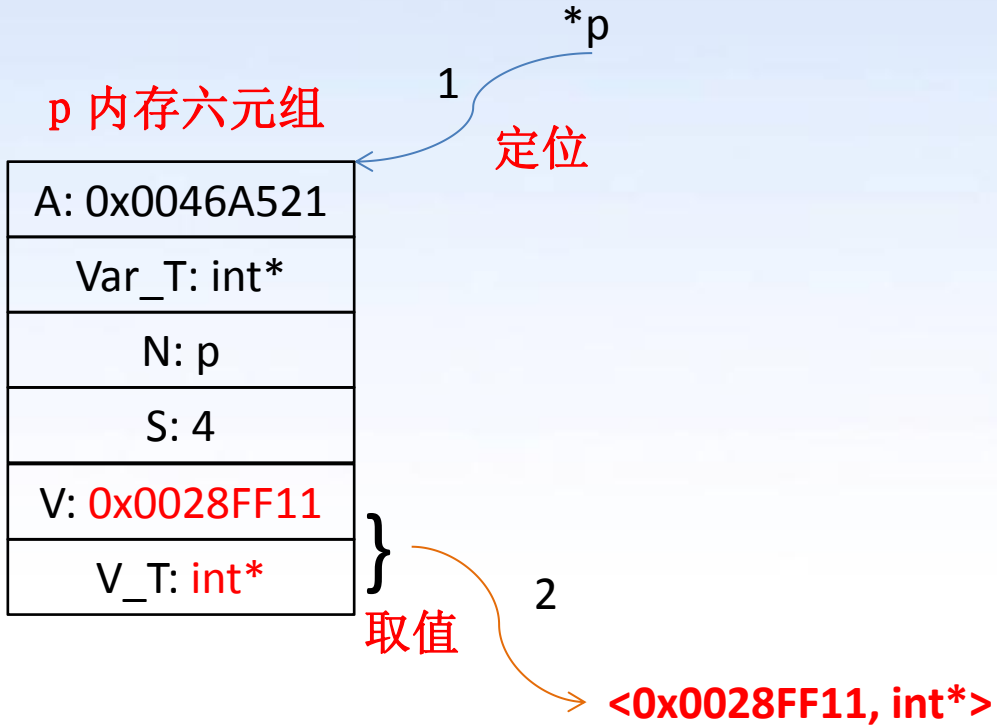
3）*p内存六元组其他属性根据V_T依次确定



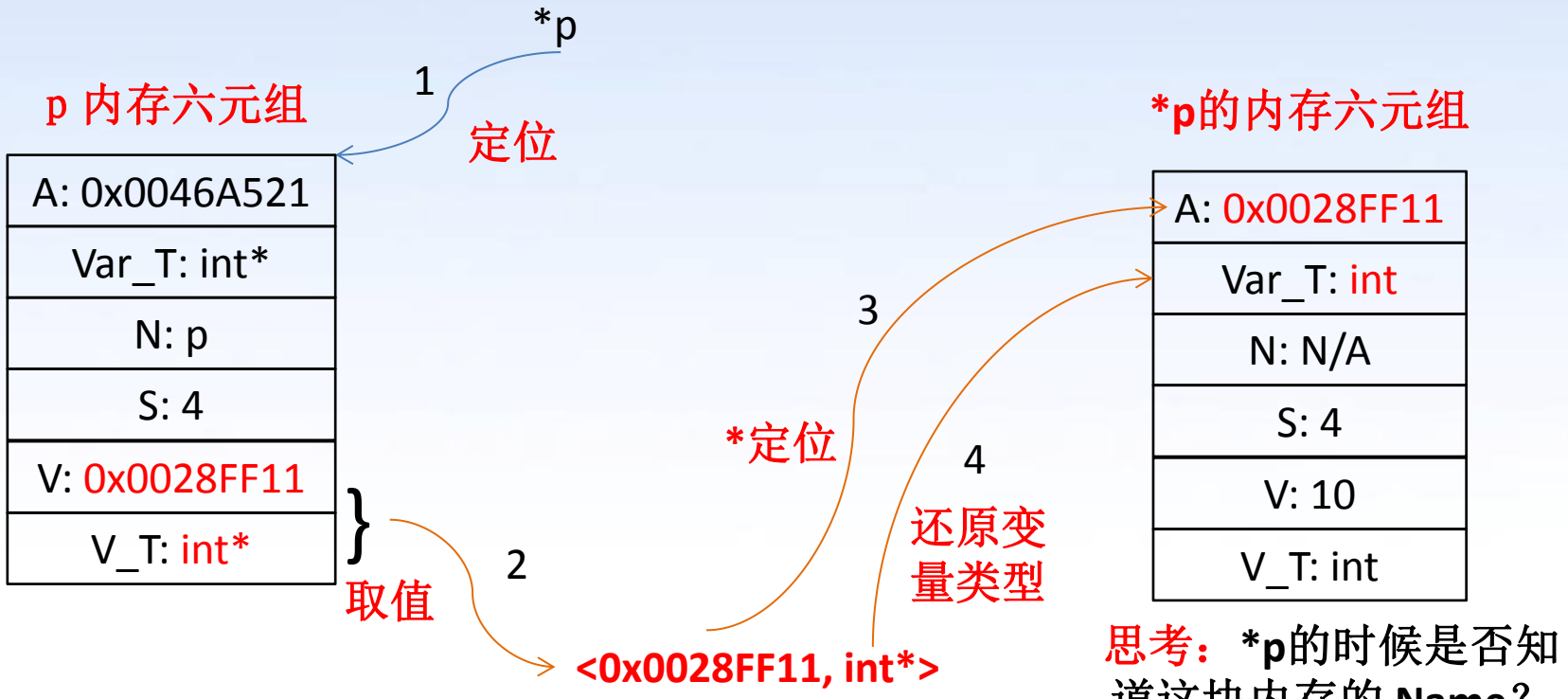
*p发生了什么？



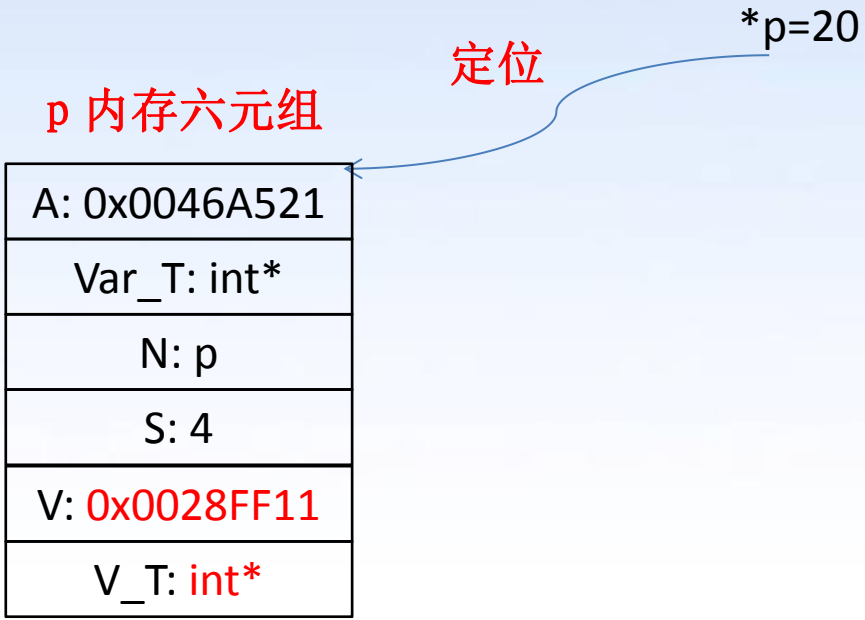
*p发生了什么？


*p
1
定位
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
2
取值
<0x0028FF11, int*>

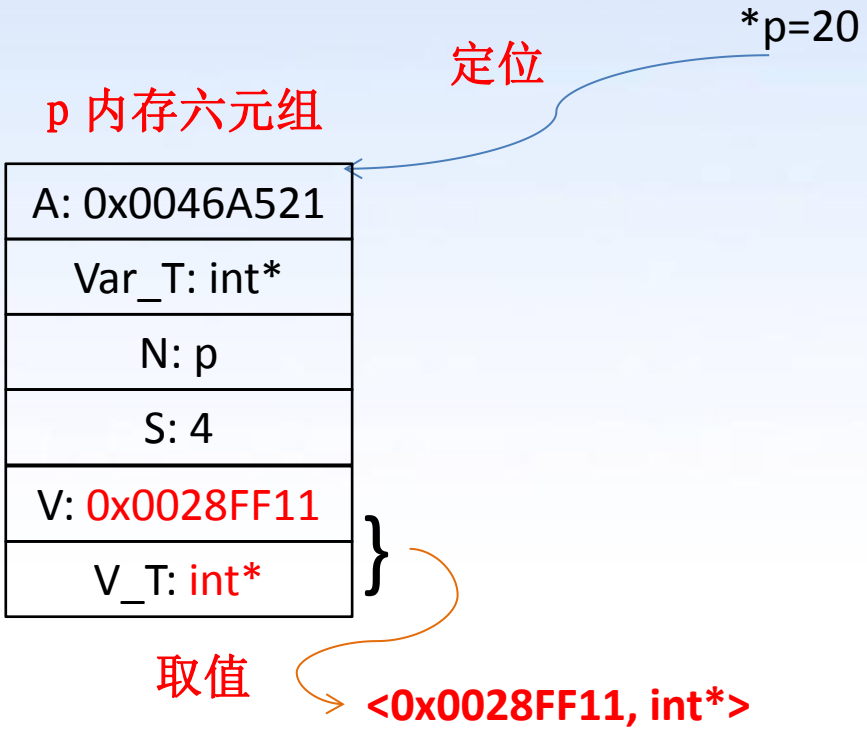
*p发生了什么？


*p
1
定位
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
取值
2
<0x0028FF11, int*>
*定位
3
4
还原变量类型
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
思考：*p的时候是否知道这块内存的 Name？

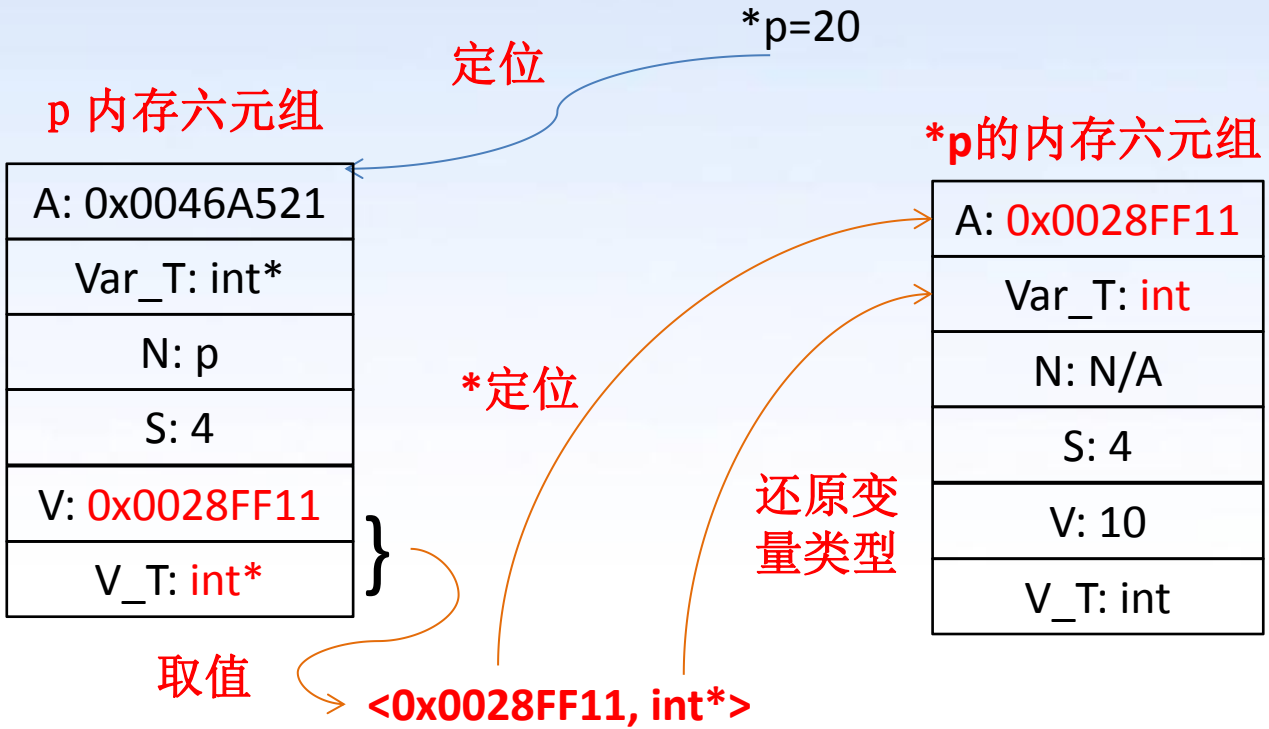
*p=20发生了什么？



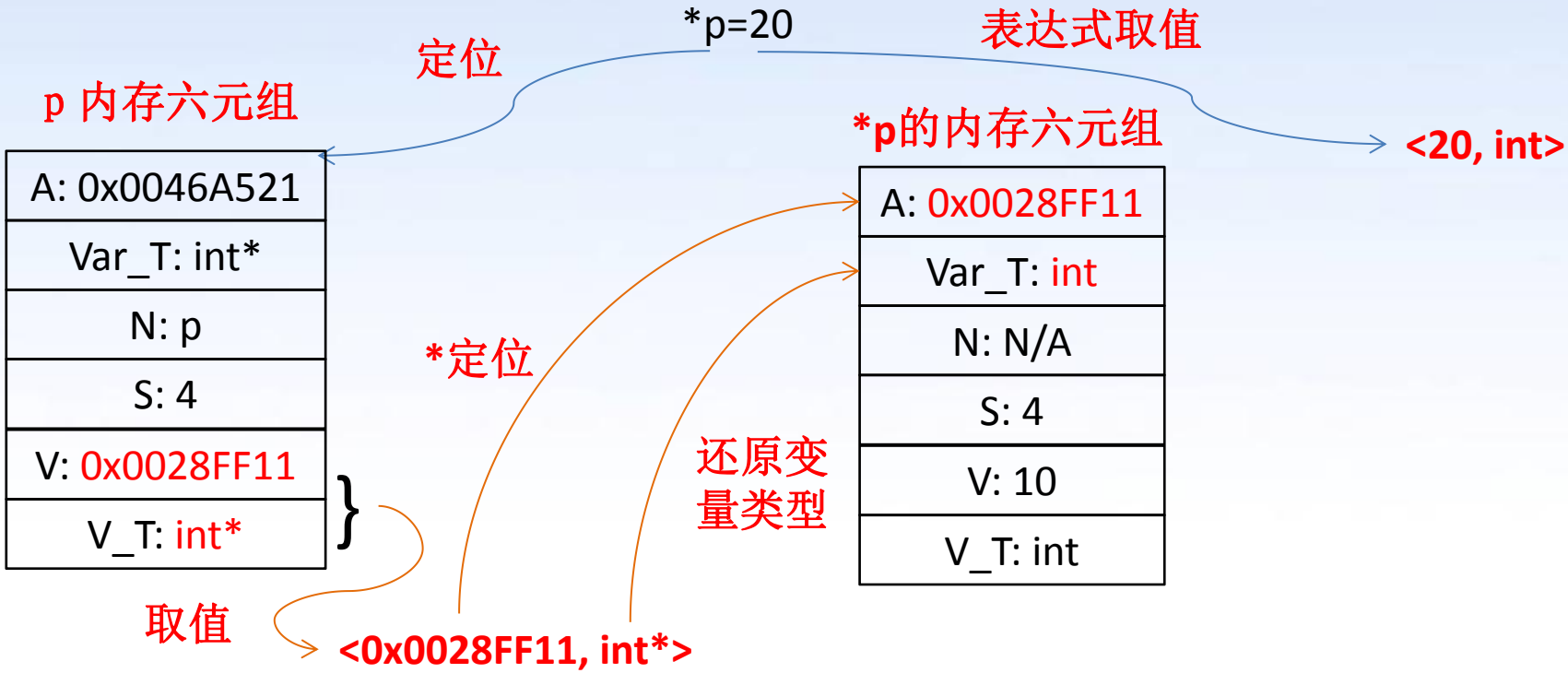
*p=20发生了什么？


*p=20
定位
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
取值
<0x0028FF11, int*>

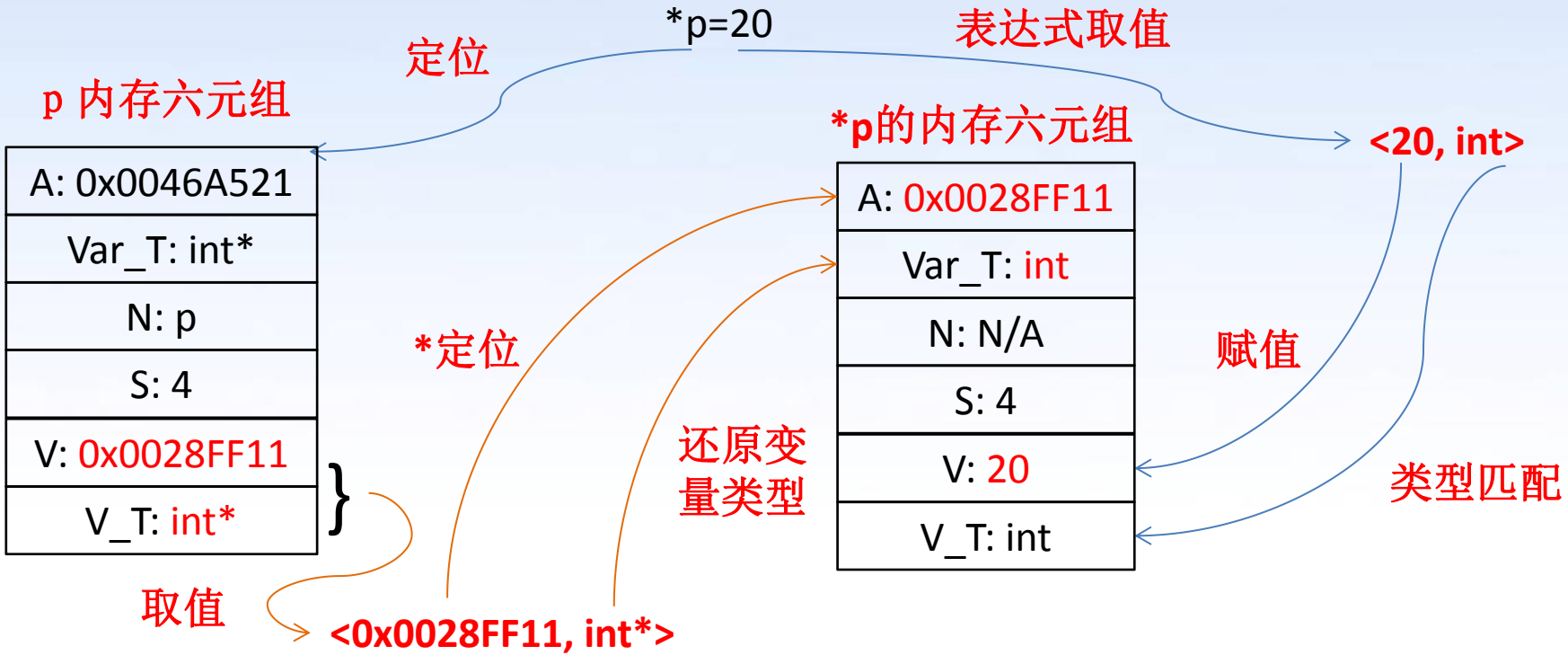
*p=20发生了什么？


*p=20
定位
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
*定位
还原变量类型
取值
<0x0028FF11, int*>

*p=20发生了什么？


*p=20
定位
表达式取值
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
<20, int>
*定位
还原变量类型
取值
<0x0028FF11, int*>

*p=20发生了什么？


*p=20
定位
表达式取值
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 20
V_T: int
<20, int>
*定位
赋值
还原变量类型
类型匹配
取值
<0x0028FF11, int*>

int* q=&(*p)发生了什么？

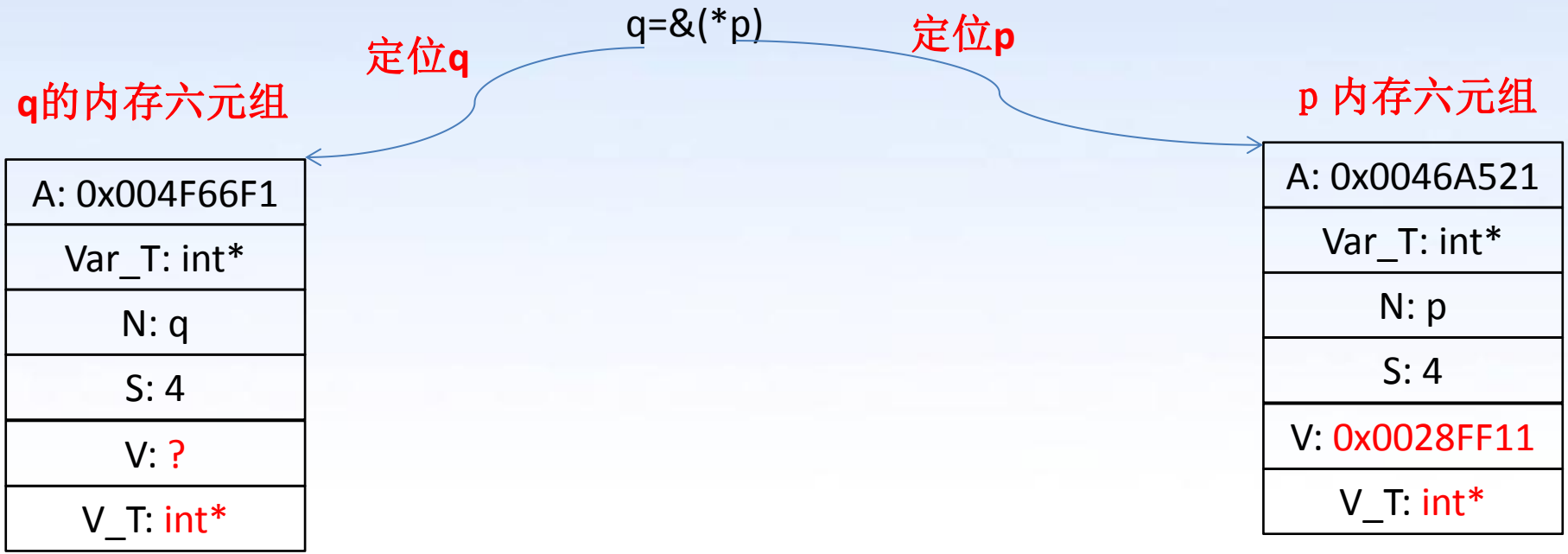
q=&(*p)

定位q

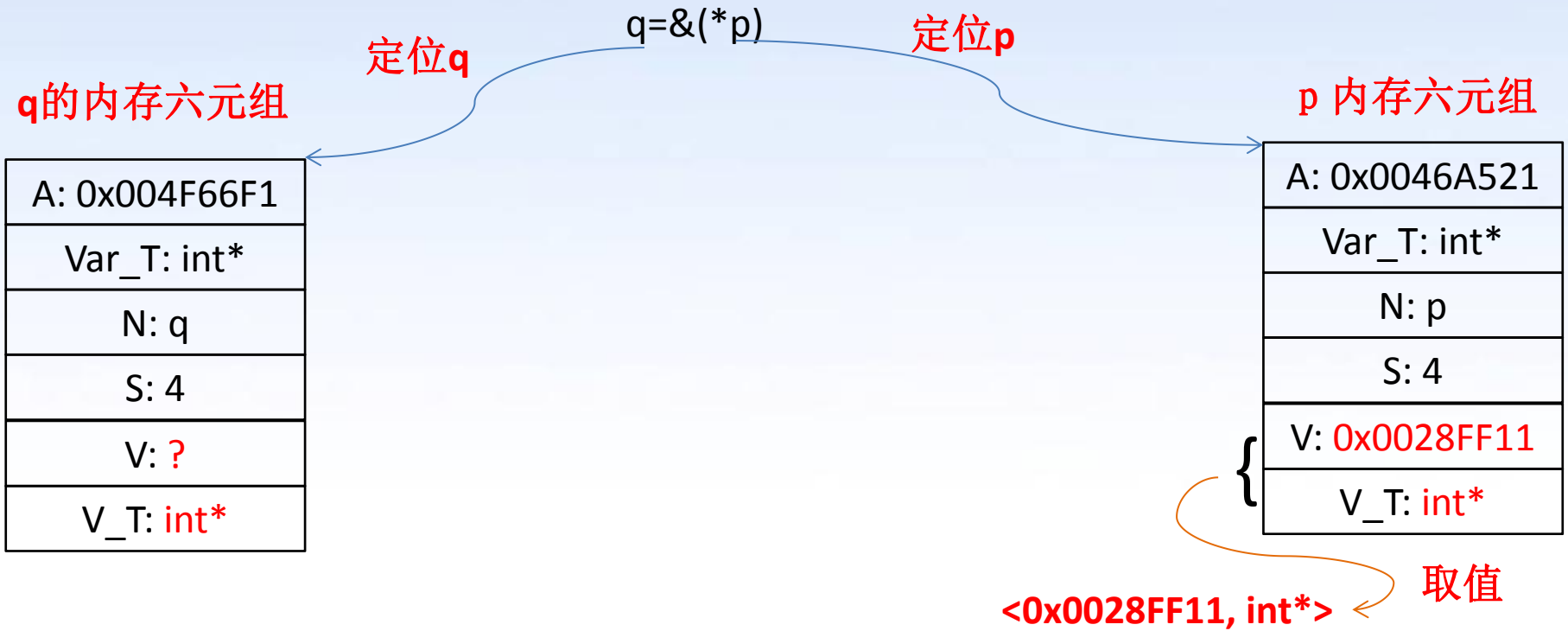
q的内存六元组

A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: ?
V_T: int*

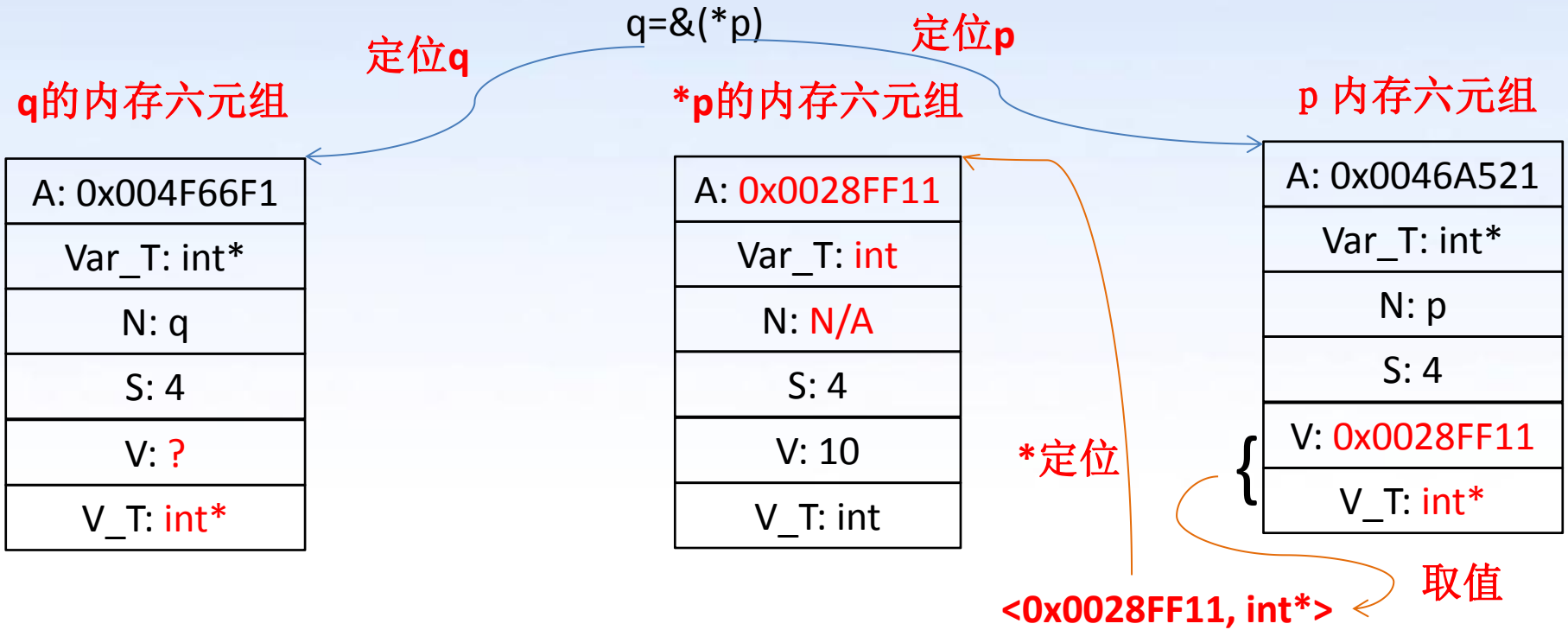
int* q=&(*p)发生了什么？


q=&(*p)
定位q
定位p
q的内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: ?
V_T: int*
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

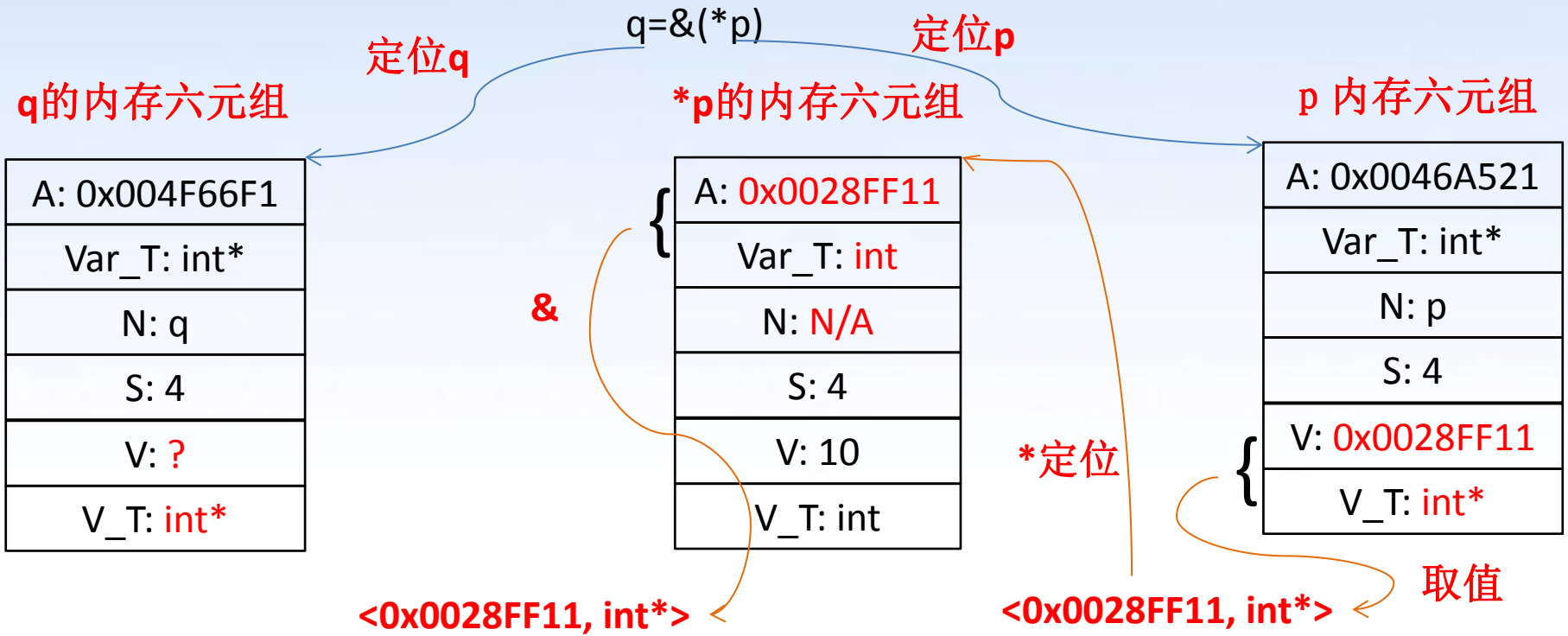
int* q=&(*p)发生了什么？


q=&(*p)
定位q
定位p
q的内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: ?
V_T: int*
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
取值
<0x0028FF11, int*>

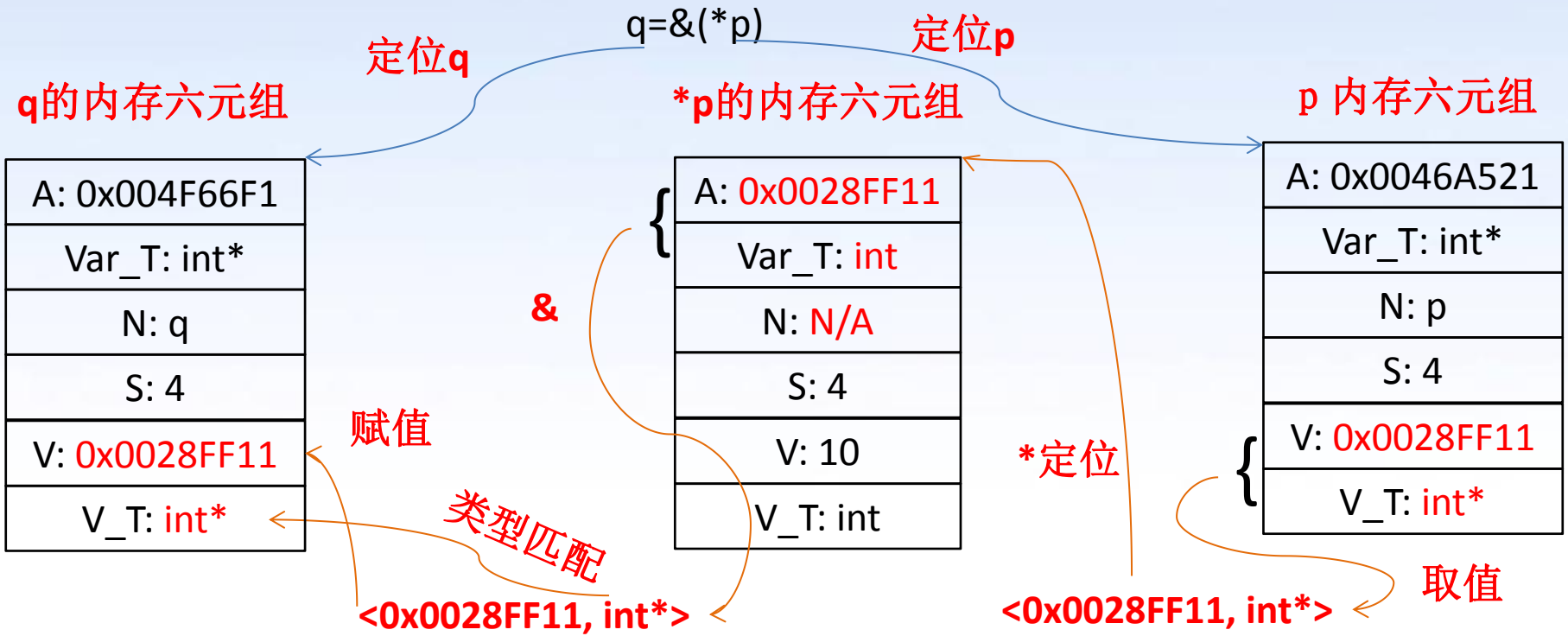
int* q=&(*p)发生了什么？


q=&(*p)
定位q
定位p
q的内存六元组
*p的内存六元组
p 内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: ?
V_T: int*
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*定位
取值
<0x0028FF11, int*>

int* q=&(*p)发生了什么？


q=&(*p)
定位q
定位p
q的内存六元组
*p的内存六元组
p 内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: ?
V_T: int*
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
&
*定位
取值
<0x0028FF11, int*>
<0x0028FF11, int*>

int* q=&(*p)发生了什么？

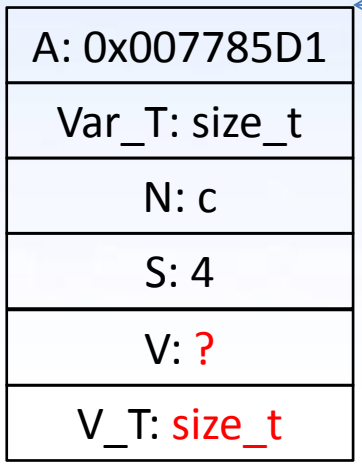

q=&(*p)
定位q
定位p
q的内存六元组
*p的内存六元组
p 内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: 0x0028FF11
V_T: int*
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
&
赋值
类型匹配
*定位
取值
<0x0028FF11, int*>
<0x0028FF11, int*>

size_t c=sizeof(*p)发生了什么？

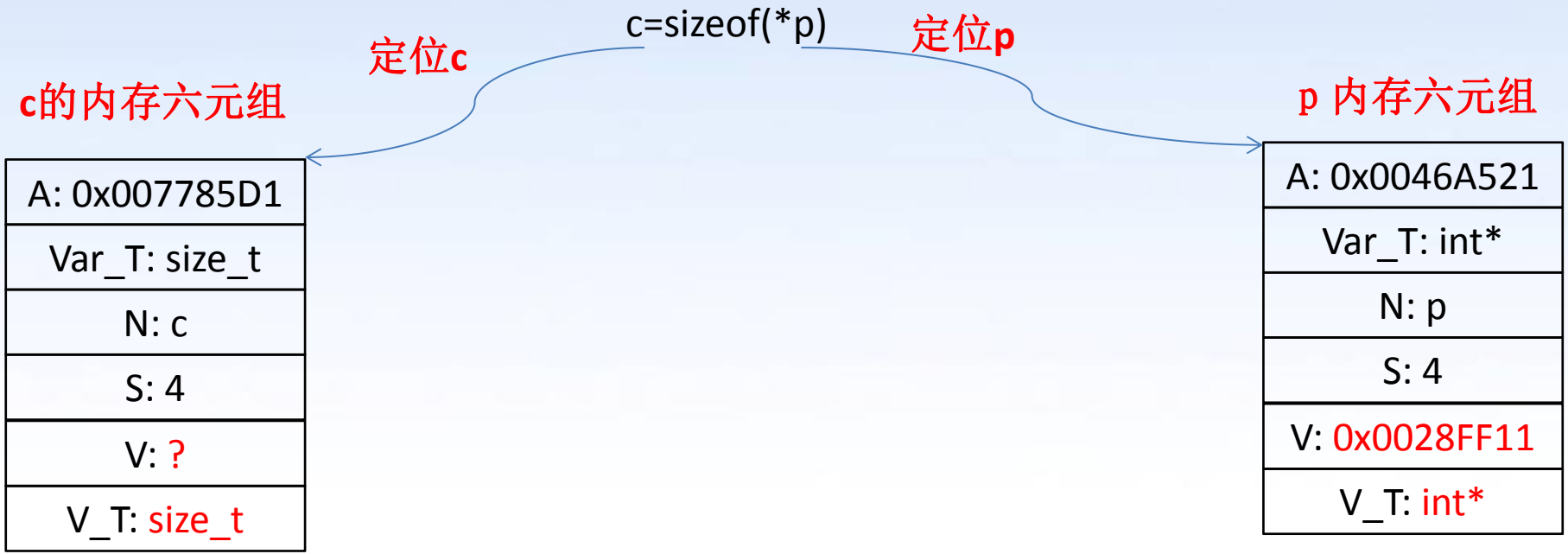
c=sizeof(*p)

定位c

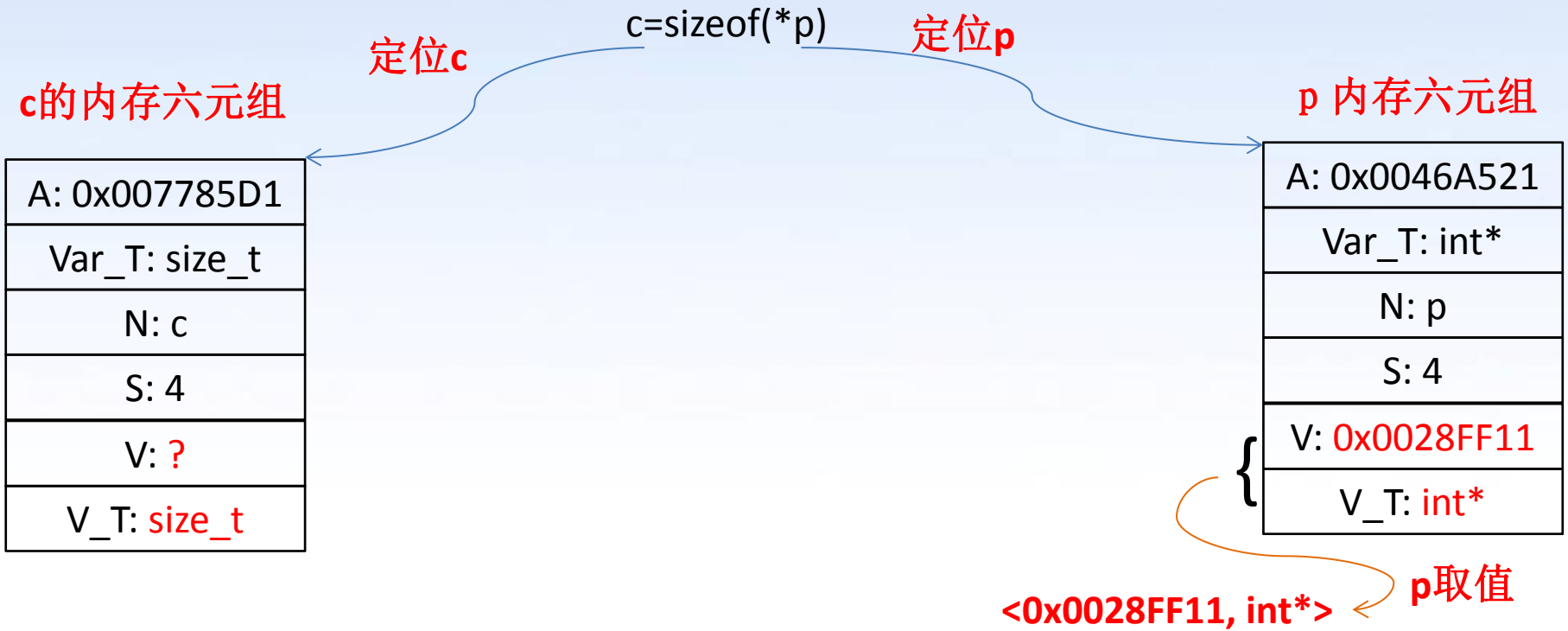
c的内存六元组



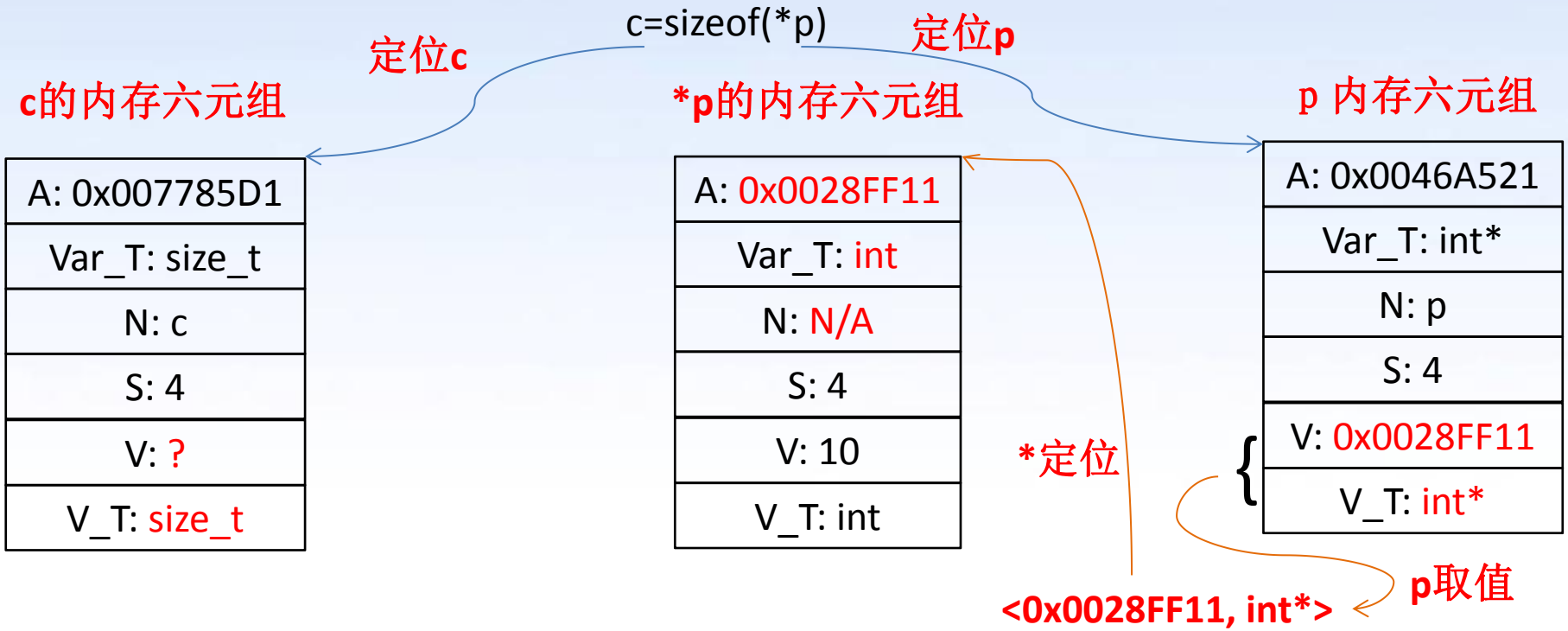
size_t c=sizeof(*p)发生了什么？



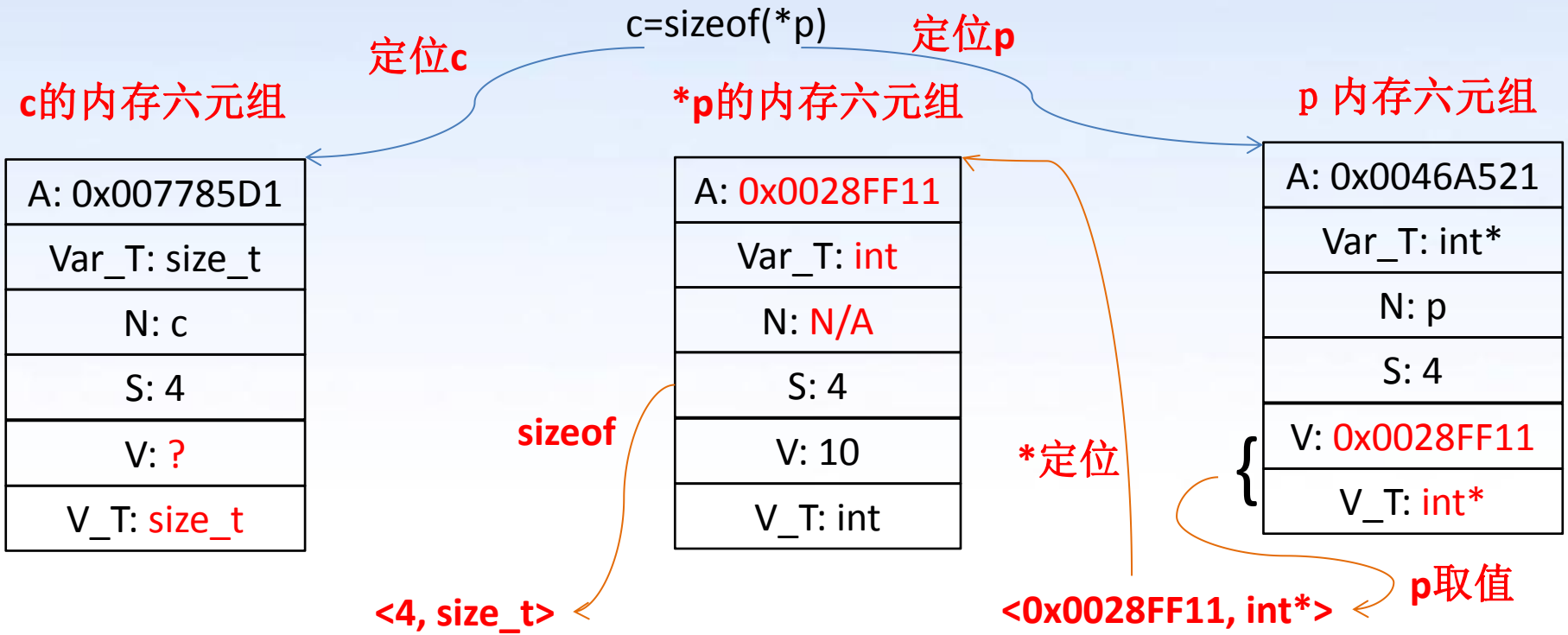
size_t c=sizeof(*p)发生了什么？



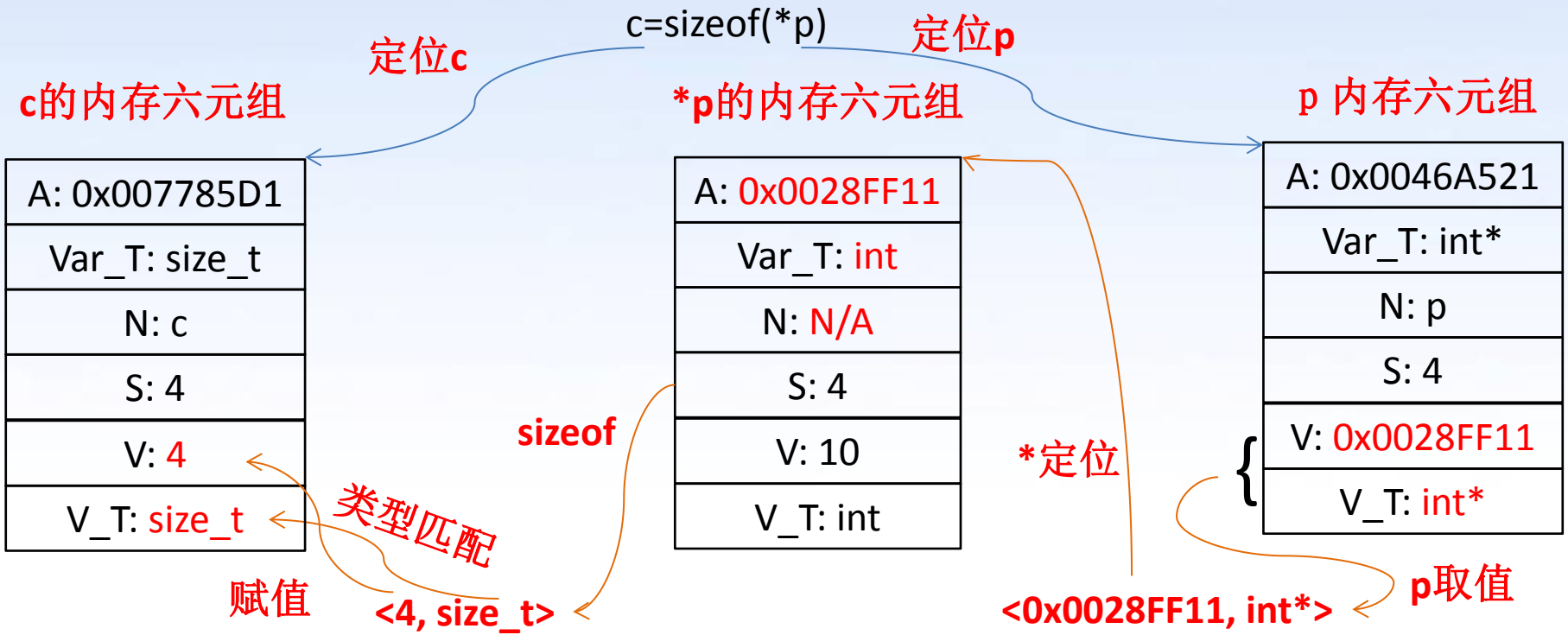
size_t c=sizeof(*p)发生了什么？



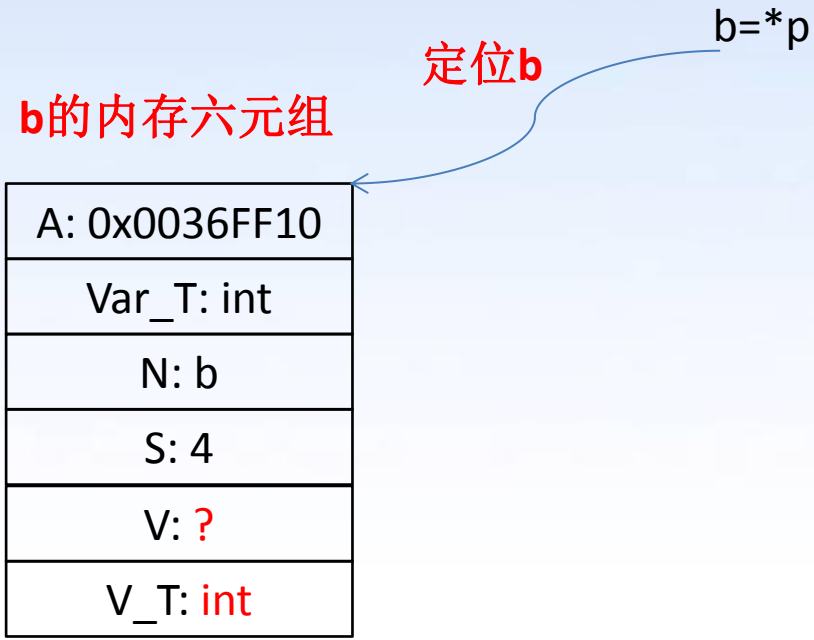
size_t c=sizeof(*p)发生了什么？



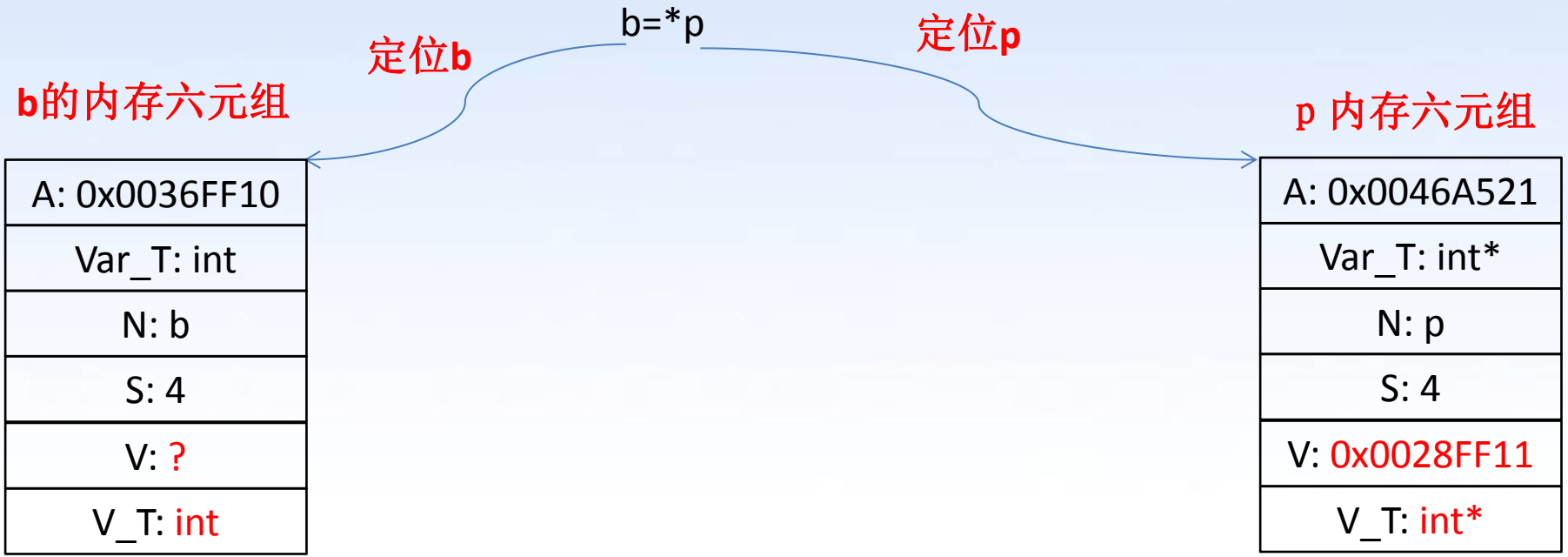
size_t c=sizeof(*p)发生了什么？



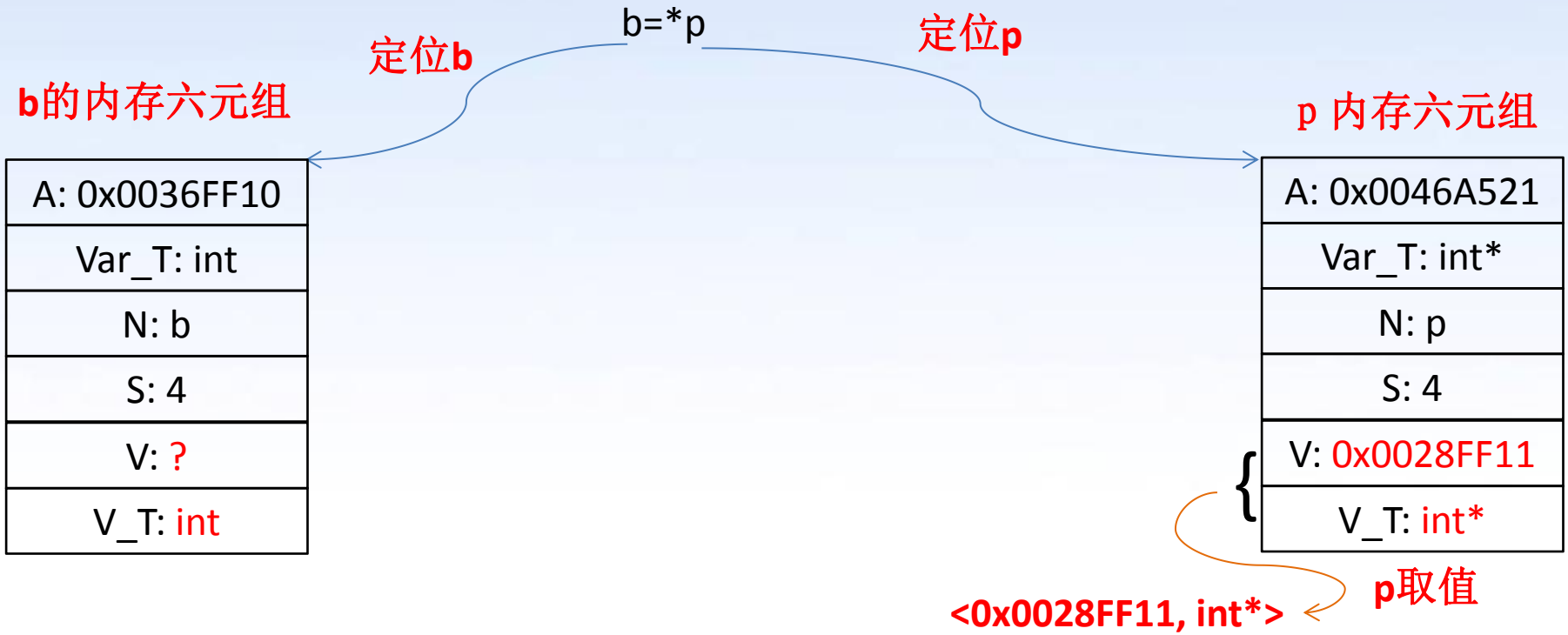
int b=*p发生了什么？



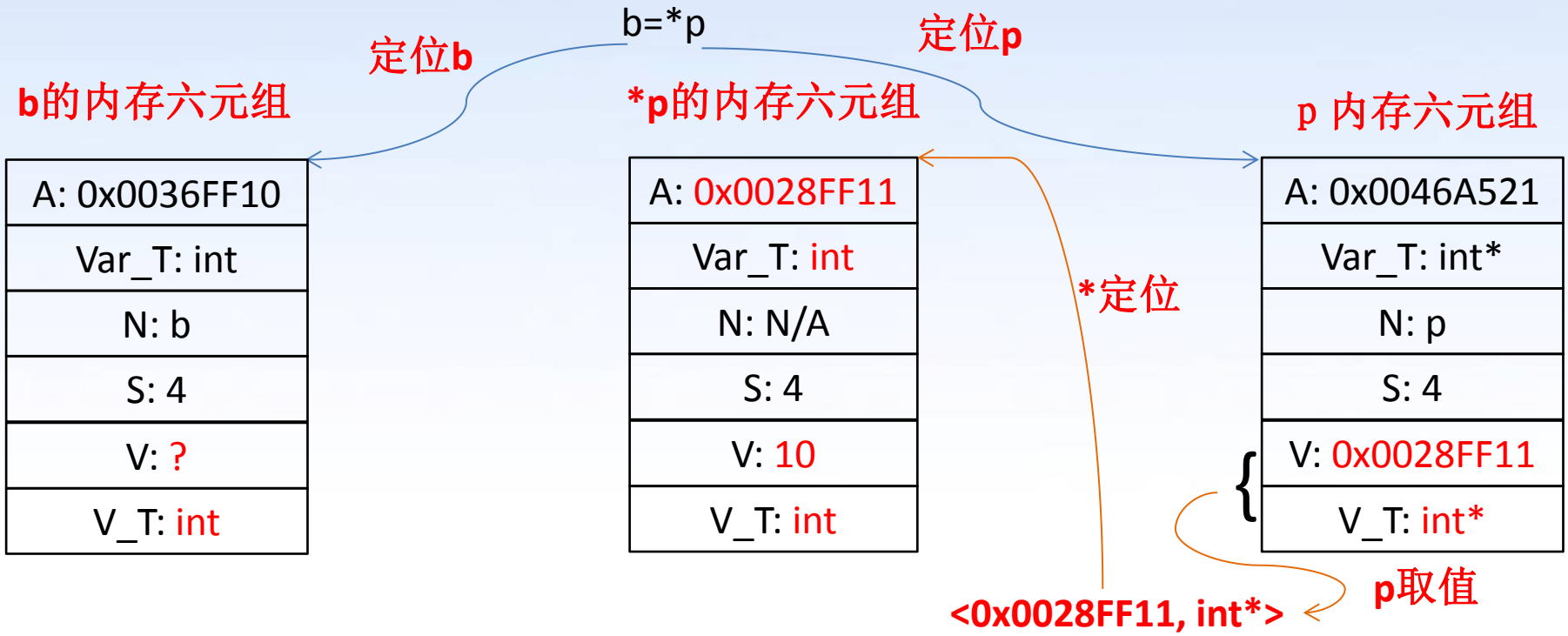
int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: ?
V_T: int
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

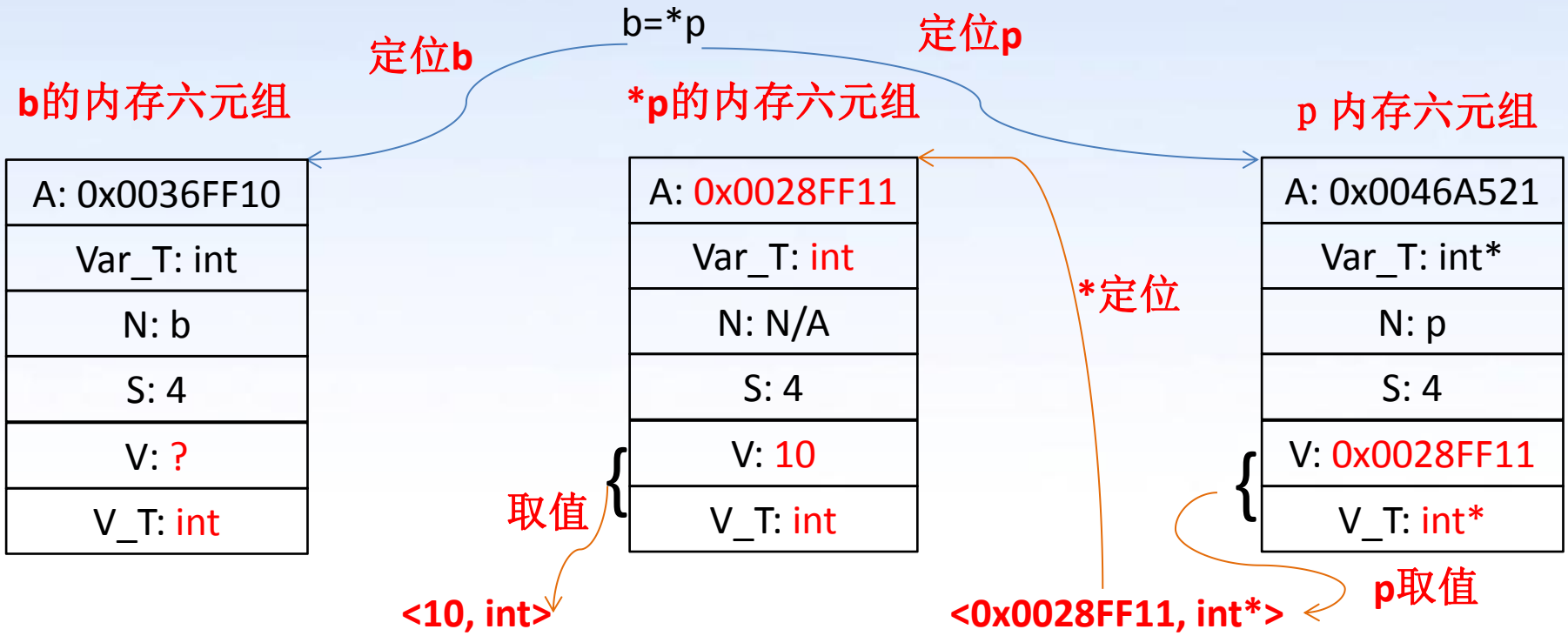
int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: ?
V_T: int
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
<0x0028FF11, int*>
p取值

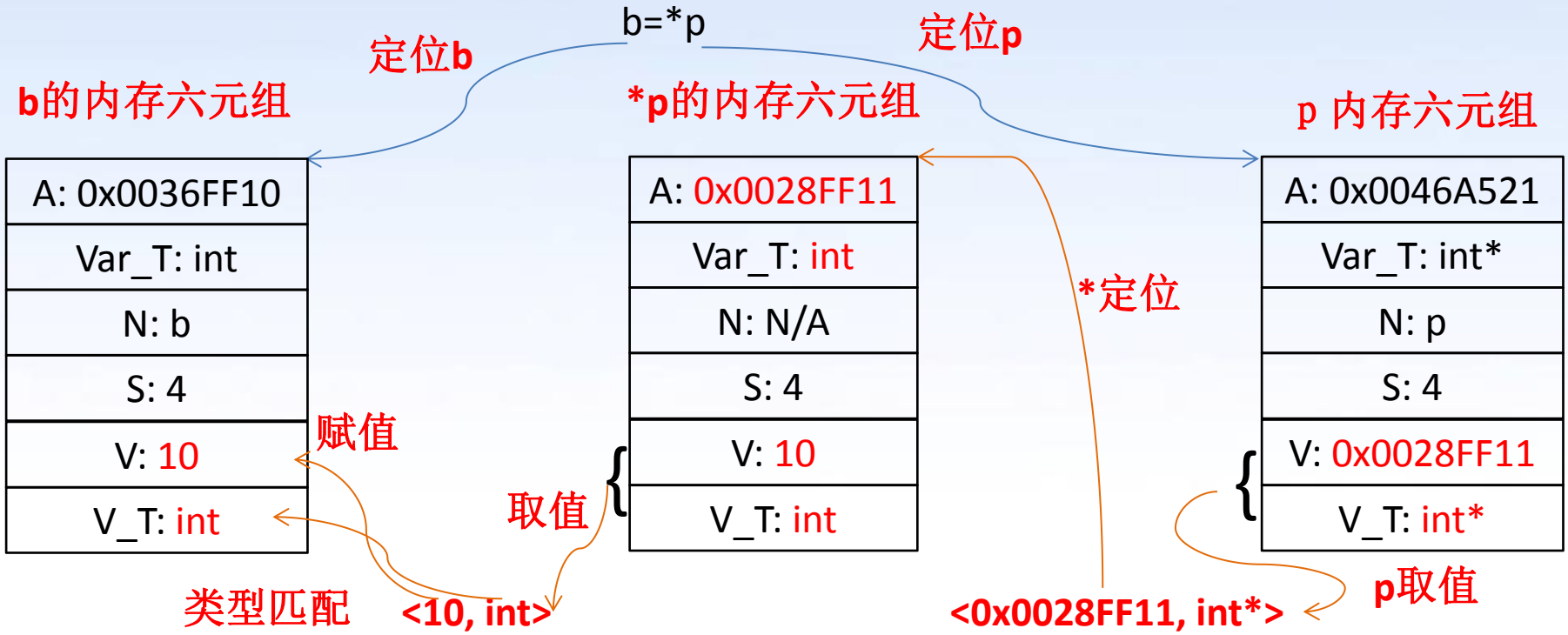
int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
*p的内存六元组
p 内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: ?
V_T: int
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*定位
<0x0028FF11, int*>
p取值

int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
*p的内存六元组
p 内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: ?
V_T: int
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*定位
取值
<10, int>
<0x0028FF11, int*>
p取值

int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
*p的内存六元组
p 内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: 10
V_T: int
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
*定位
赋值
取值
类型匹配
<10, int>
<0x0028FF11, int*>
p取值

总结：使用变量名定位内存

a;

定位a

a的内存六元组

A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int

int a = 10;

三个取值操作：

1、&a

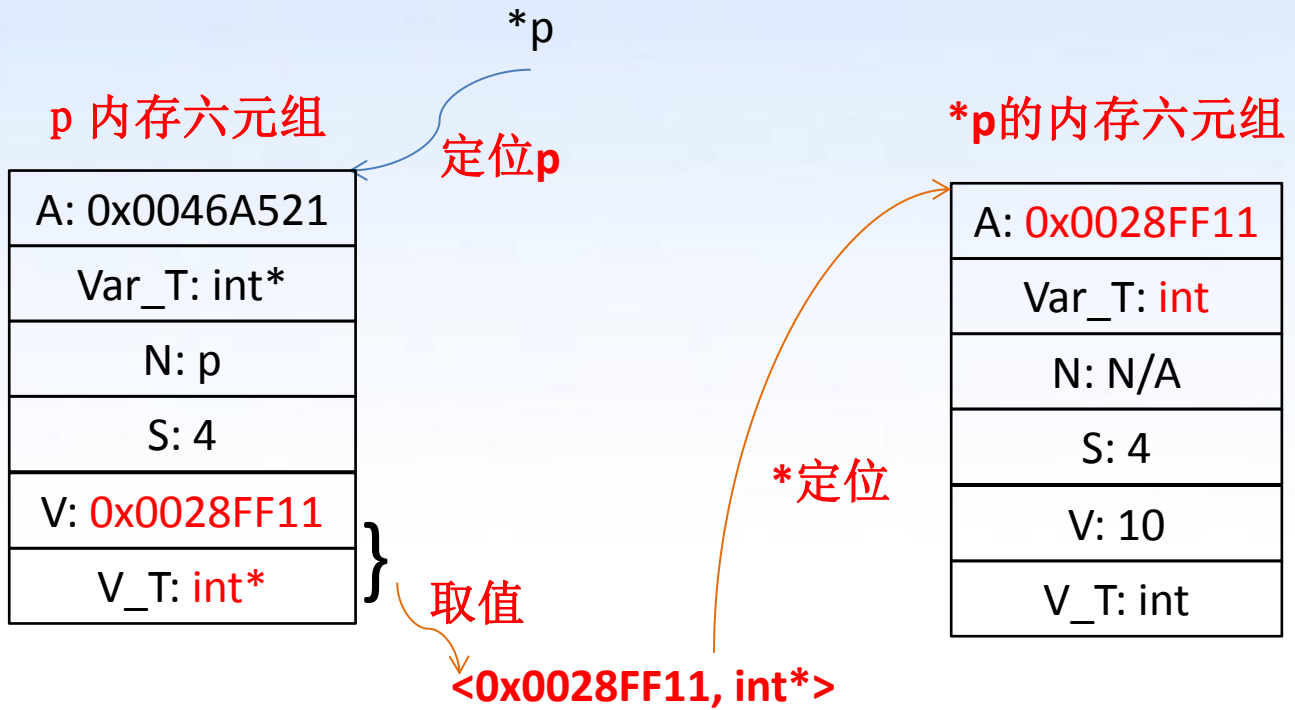
2、sizeof(a)

3、a

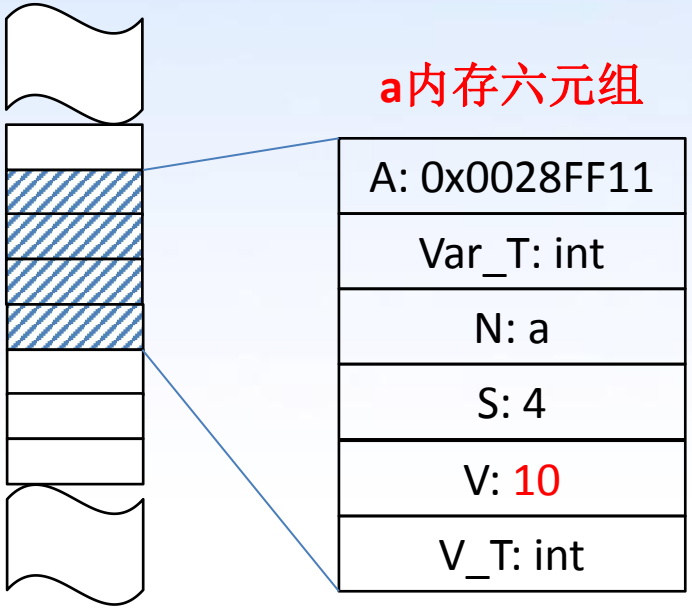
总结：使用指针变量类型定位内存

int a = 10;
int* p=&a;

三个取值操作：
1、&(*p)
2、sizeof(*p)
3、*p



只有指针类型的值能用*进行间接定位




```
int a=10;
*a=20;
```

```
error: invalid type argument of unary '*' (have 'int')
```

*a的时候发生了什么？

1、定位a的内存

2、获得a的表示值<10, int>

不是一个有效的指针类型，不能使用*操作符

int b=*p+1 vs int b=*(p+1)

```
int a = 10;
int* p=&a;
```

int b = *p+1;

1、定位p的内存
2、p取值<value, value_type>
3、定位*p的内存
4、取*p内存的表示值
5、表示值+1
6、将值写入变量b

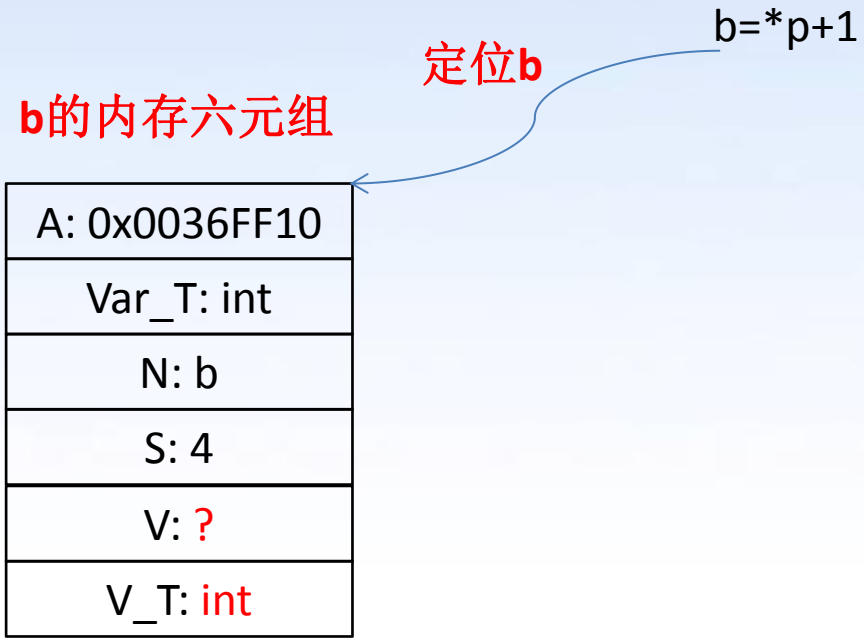
int b = *(p+1);

1、定位p的内存
2、p取值<value, value_type>
3、计算p+1的值
4、定位*(p+1)的内存
5、取*(p+1) 内存的表示值
6、将值写入变量b

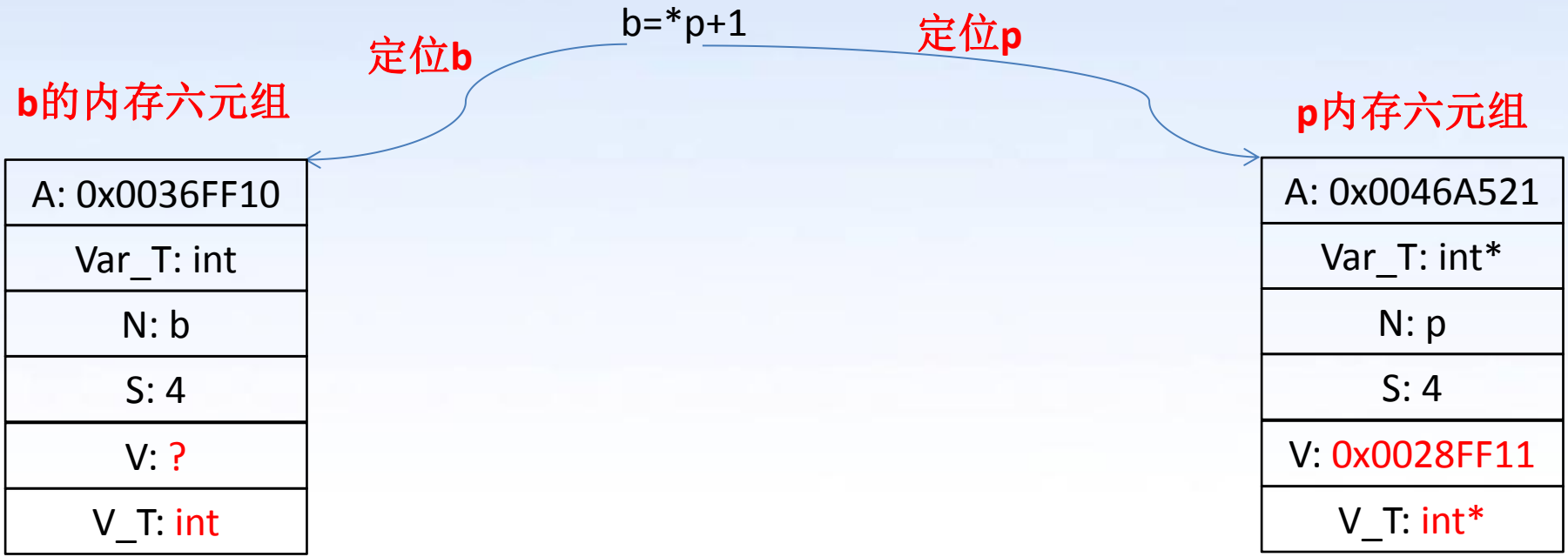
p的内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

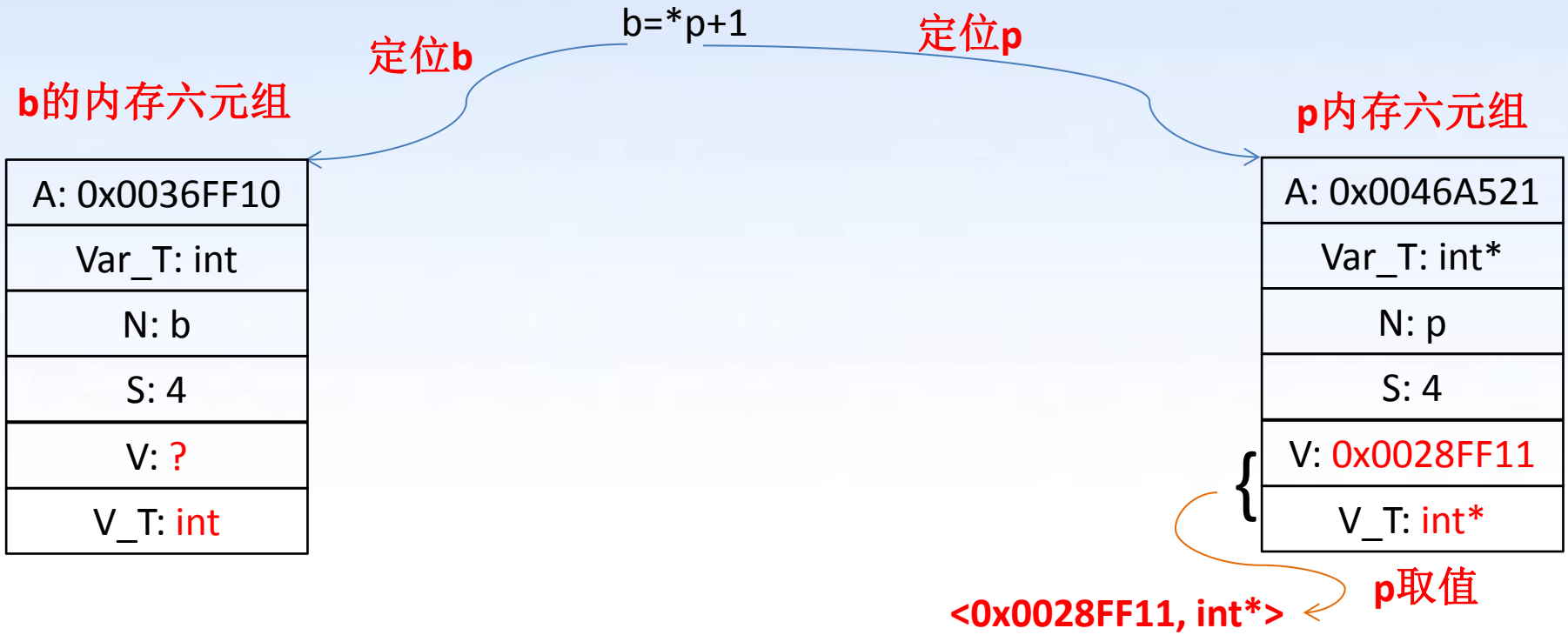
int b=*p+1发生了什么？



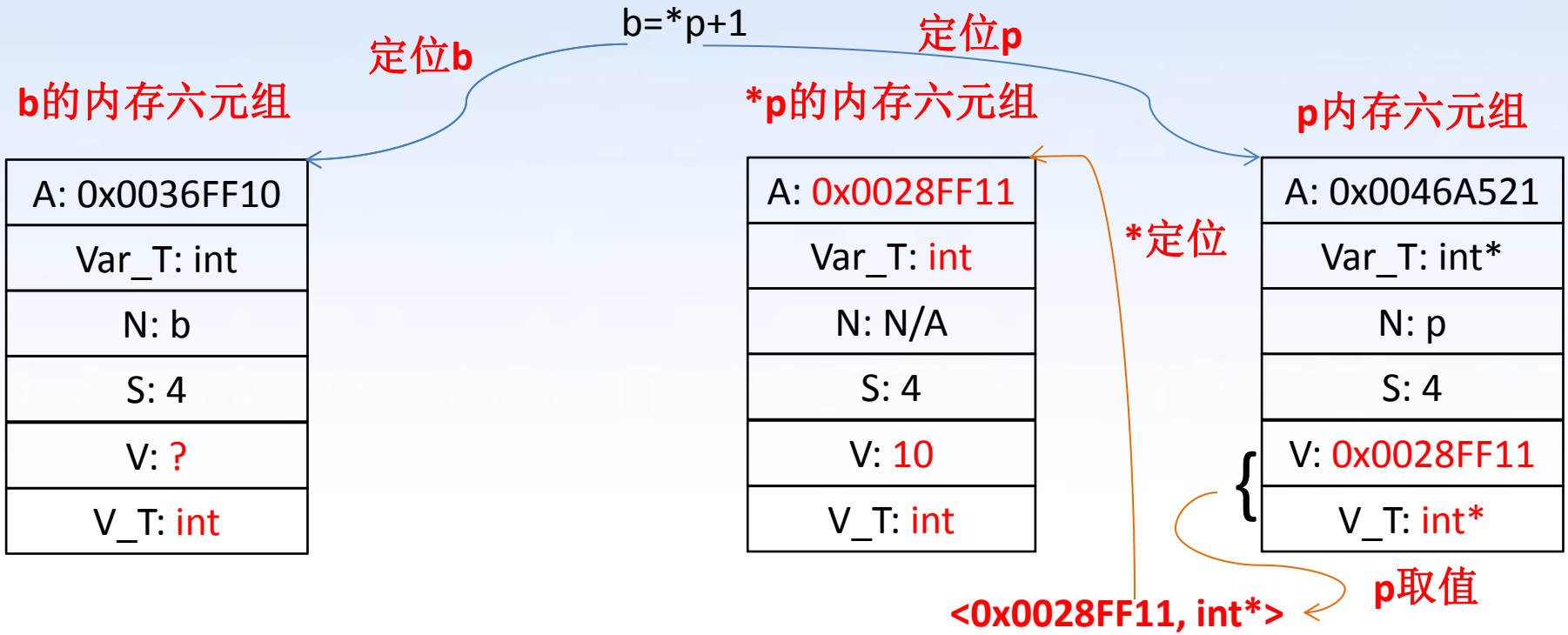
int b=*p+1发生了什么？



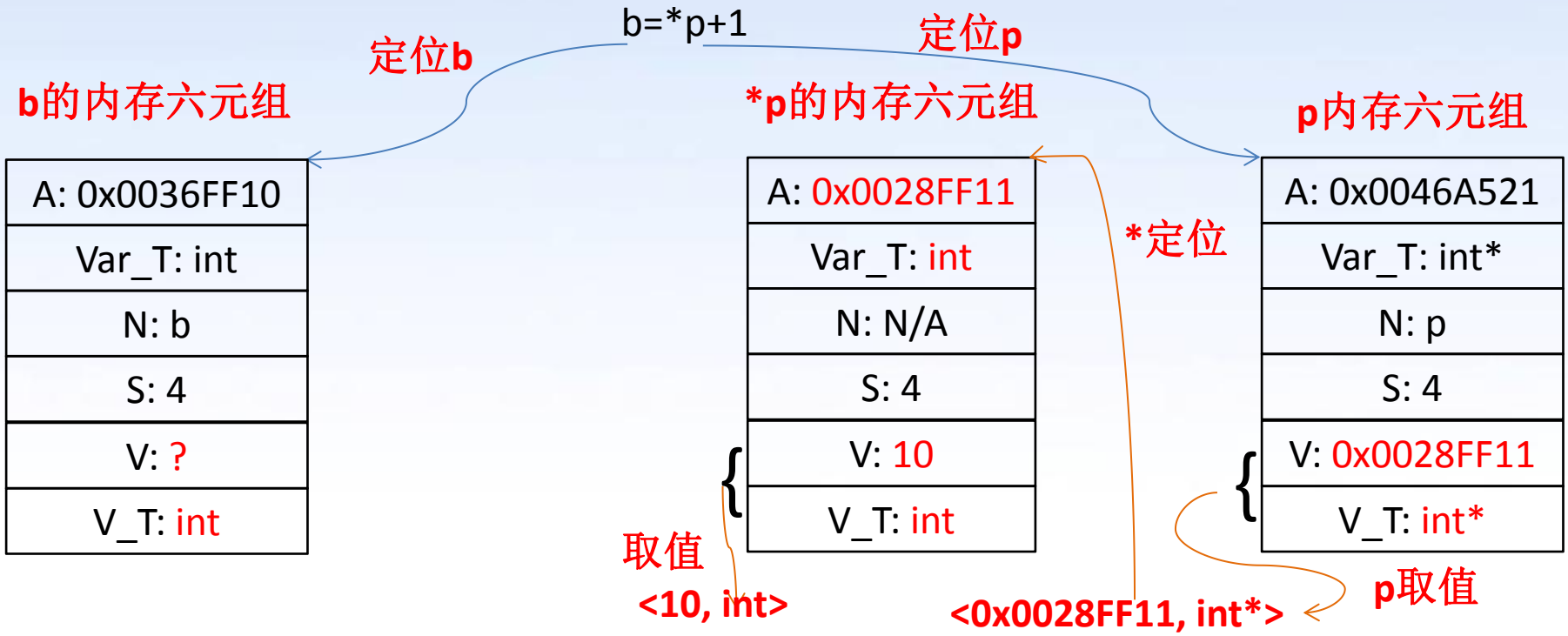
int b=*p+1发生了什么？



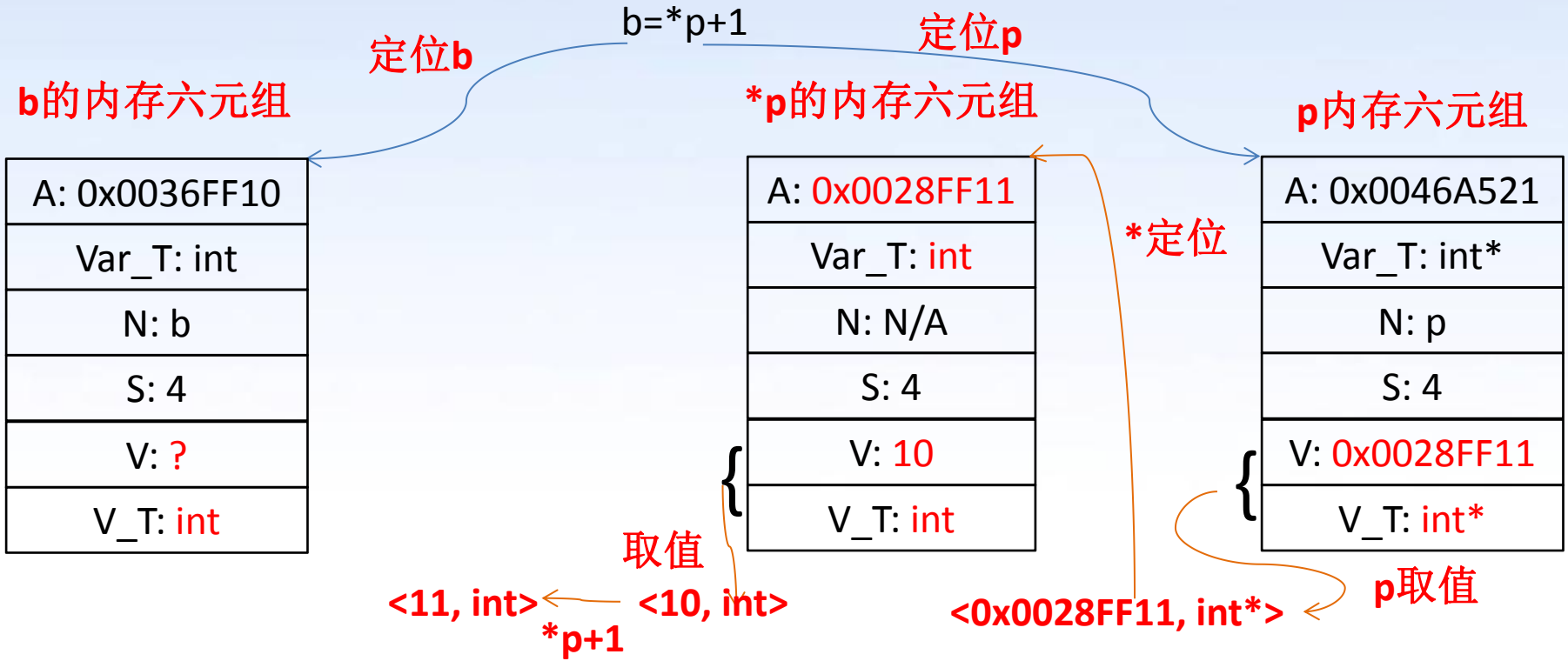
int b=*p+1发生了什么？



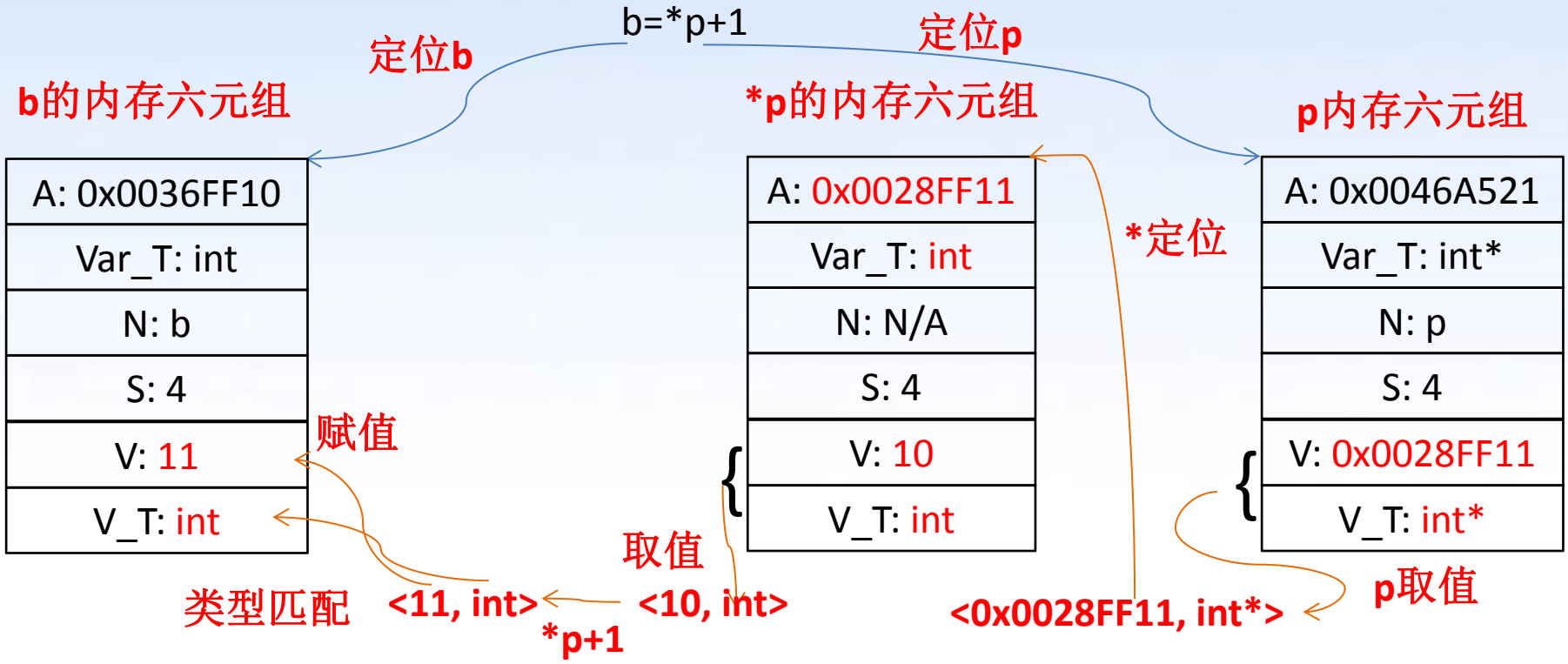
int b=*p+1发生了什么？



int b=*p+1发生了什么？



int b=*p+1发生了什么？



int b=*(p+1)的计算过程

```
int a = 10;
int* p=&a;
```

int b = *(p+1);

1、定位p的内存
2、p取值<value, value_type>
3、计算p+1的值
4、定位*(p+1)的内存
5、取*(p+1) 内存的表示值
6、将值写入变量b

p的内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

指针变量p+n到底是什么意思？

int a = 10;
int* p=&a;

p+n=？

1、定位p的内存
2、p取值<value, value_type>
3、p+n的value=
p的value+sizeof(*p)*n
4、p+n的Value_Type不变

p的内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

指针变量p+n到底是什么意思？

int a = 10;
int* p=&a;

p+n=？

1、定位p的内存
2、p取值<value, value_type>
3、p+n的value=
p的value+sizeof(*p)*n
4、p+n的Value_Type不变

p的内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

取值

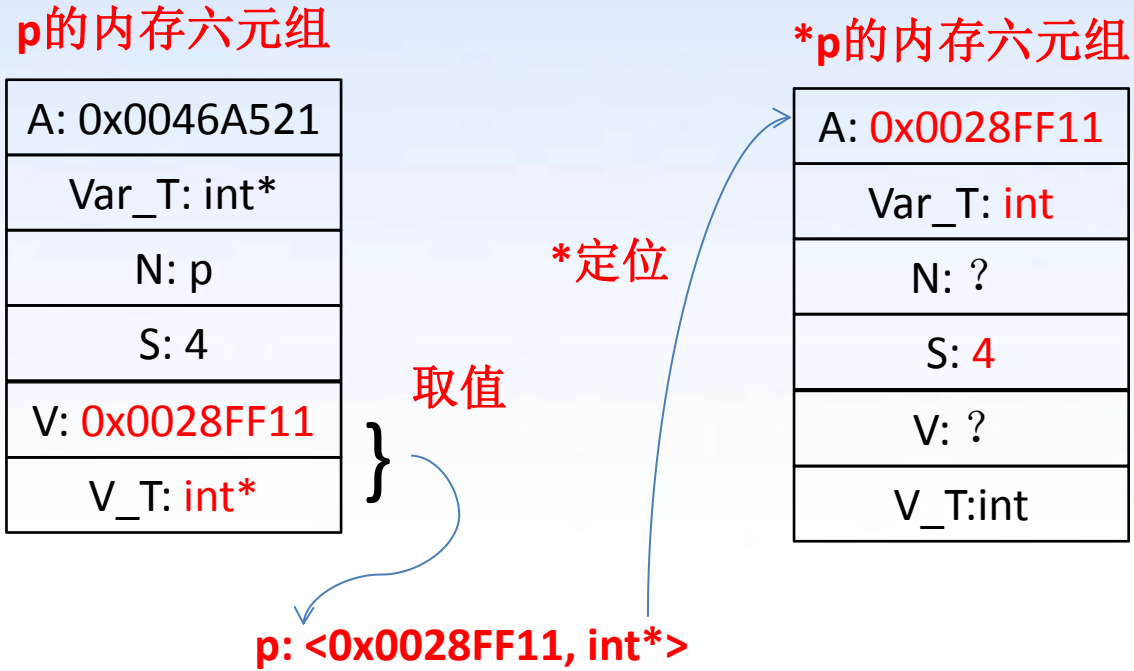
p: <0x0028FF11, int*>

指针变量p+n到底是什么意思？

int a = 10;
int* p=&a;

p+n=？

1、定位p的内存
2、p取值<value, value_type>
3、p+n的value=
p的value+sizeof(*p)*n
4、p+n的Value_Type不变

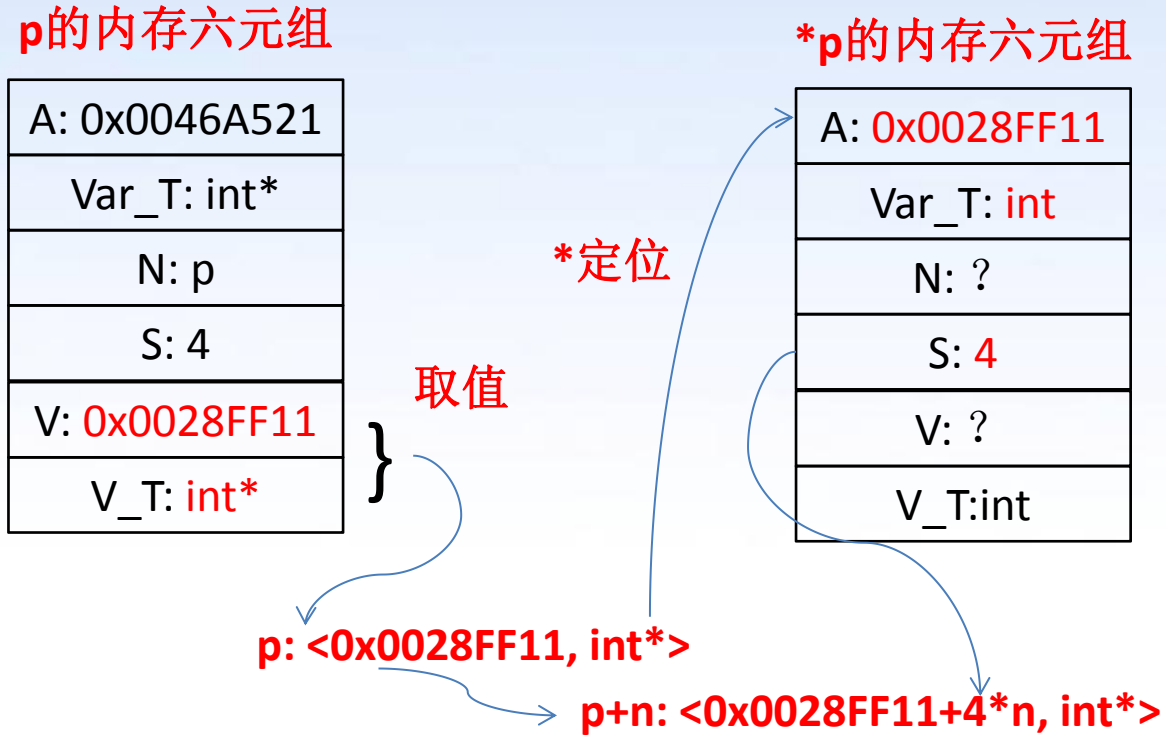


指针变量p+n到底是什么意思？

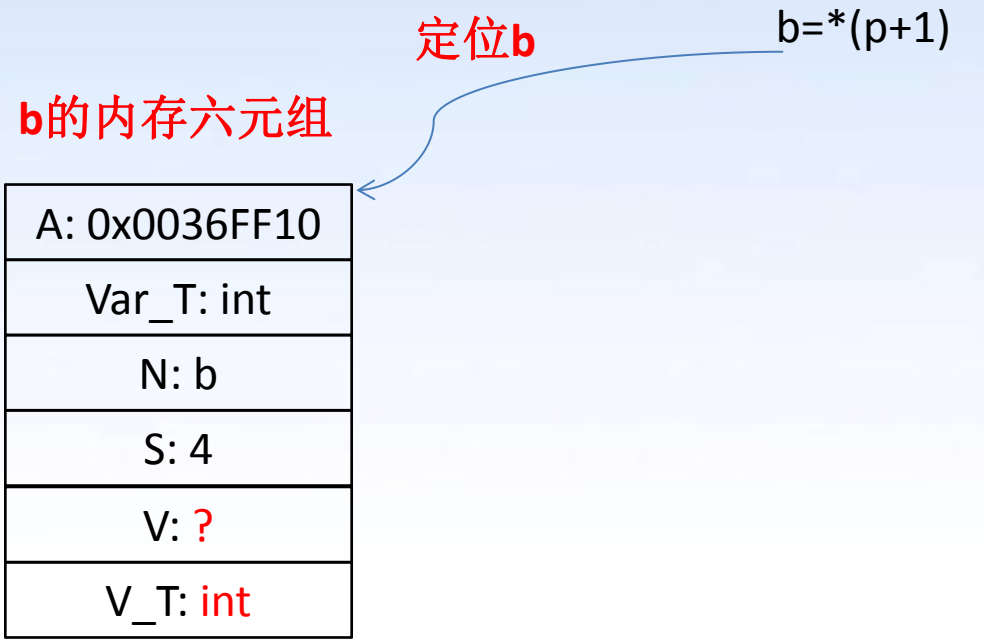
```
int a = 10;
int* p=&a;
```

p+n=？

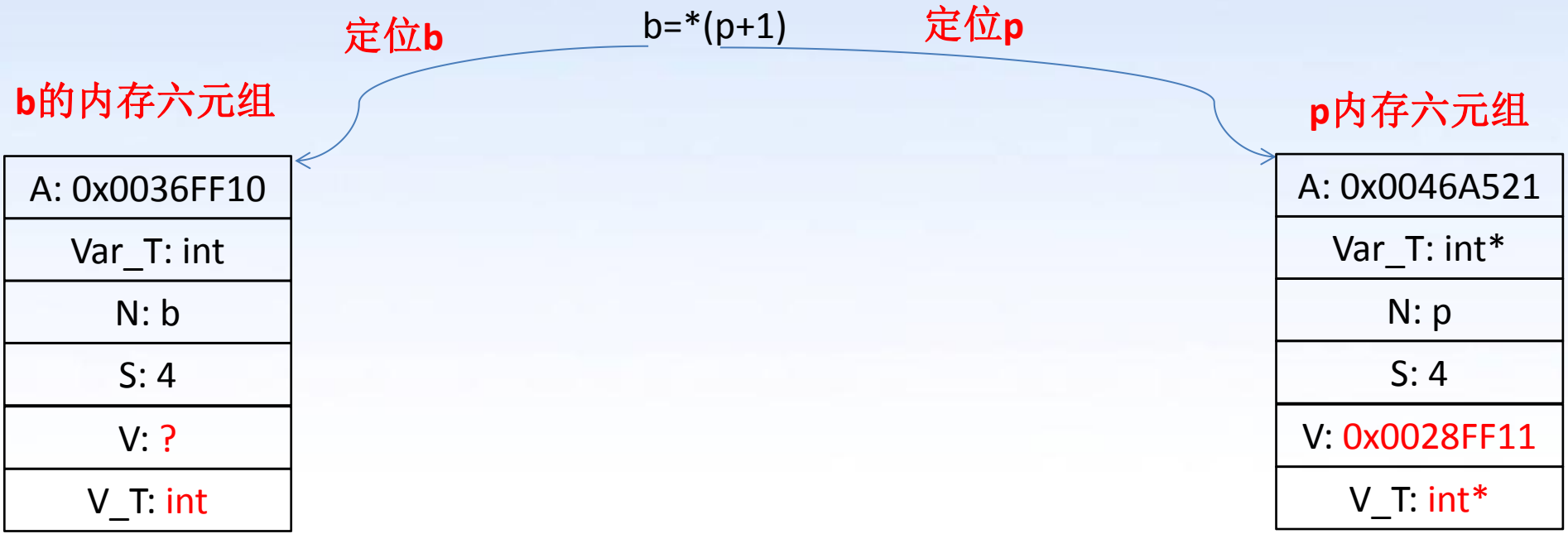
1、定位p的内存
2、p取值<value, value_type>
3、p+n的value=
p的value+sizeof(*p)*n
4、p+n的Value_Type不变



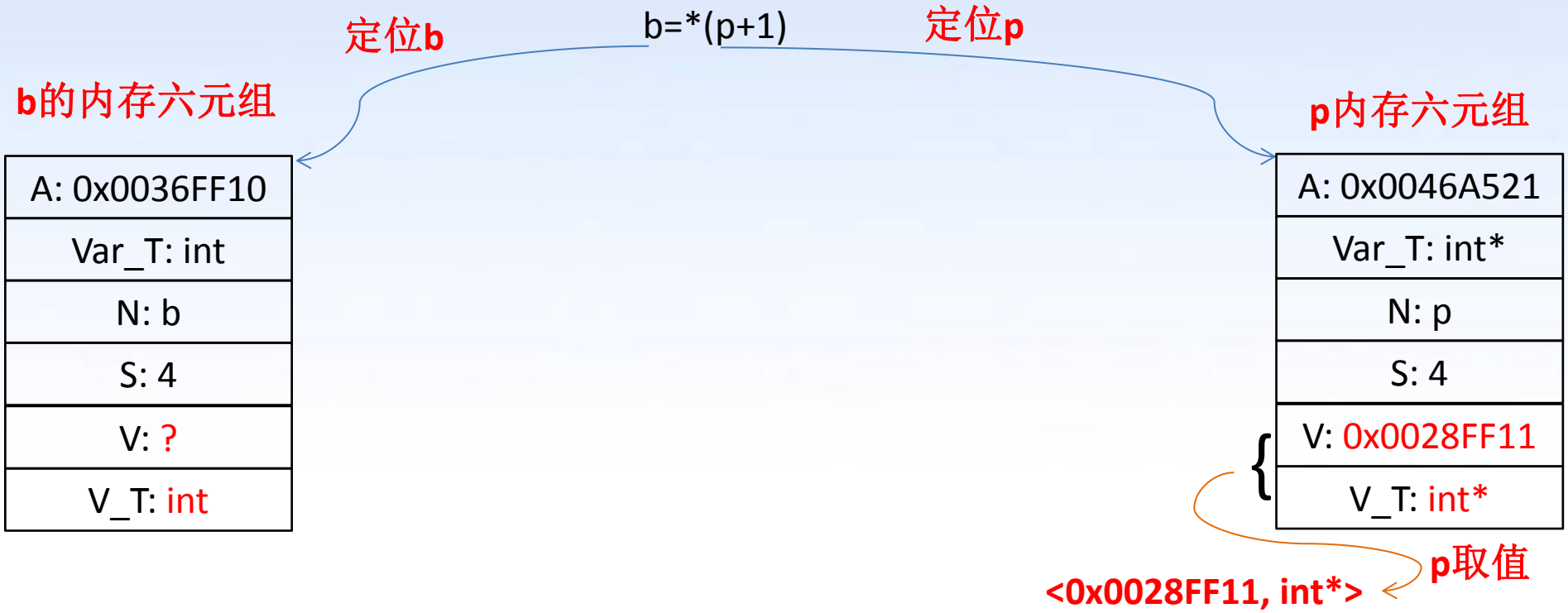
int b=*(p+1)发生了什么？



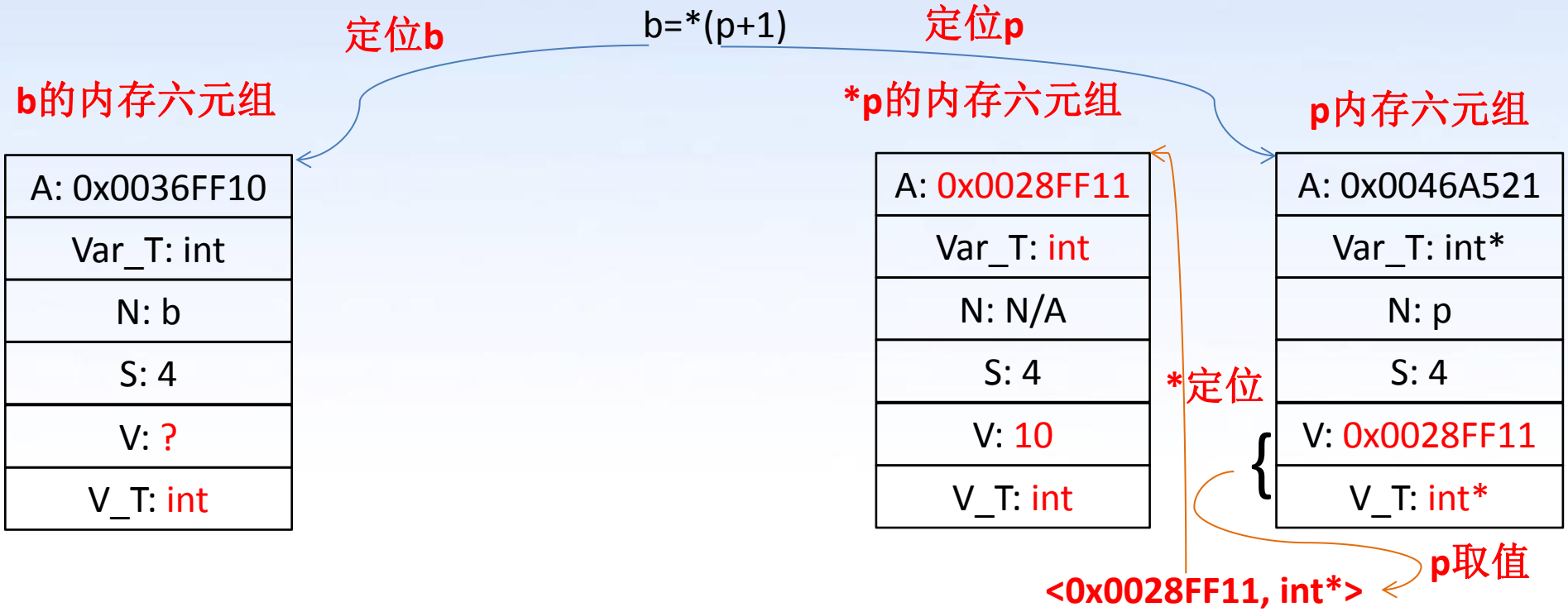
int b=*(p+1)发生了什么？



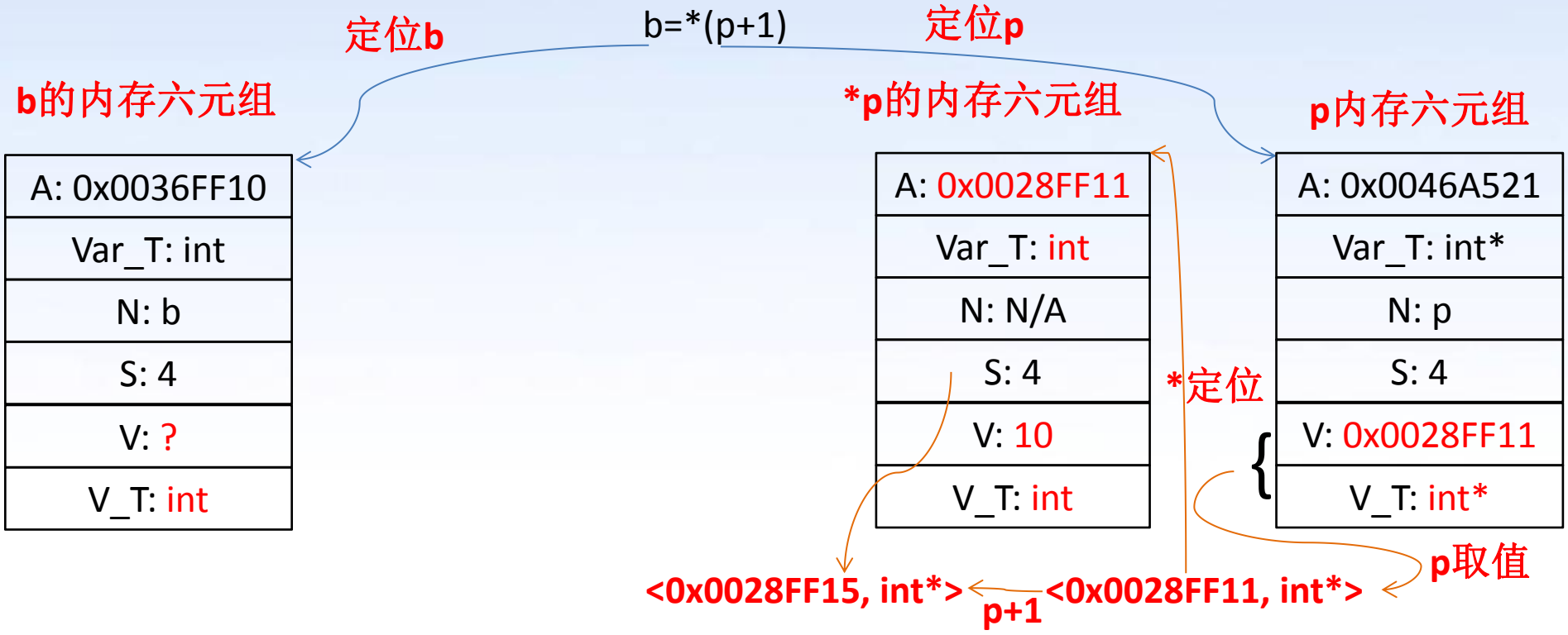
int b=*(p+1)发生了什么？



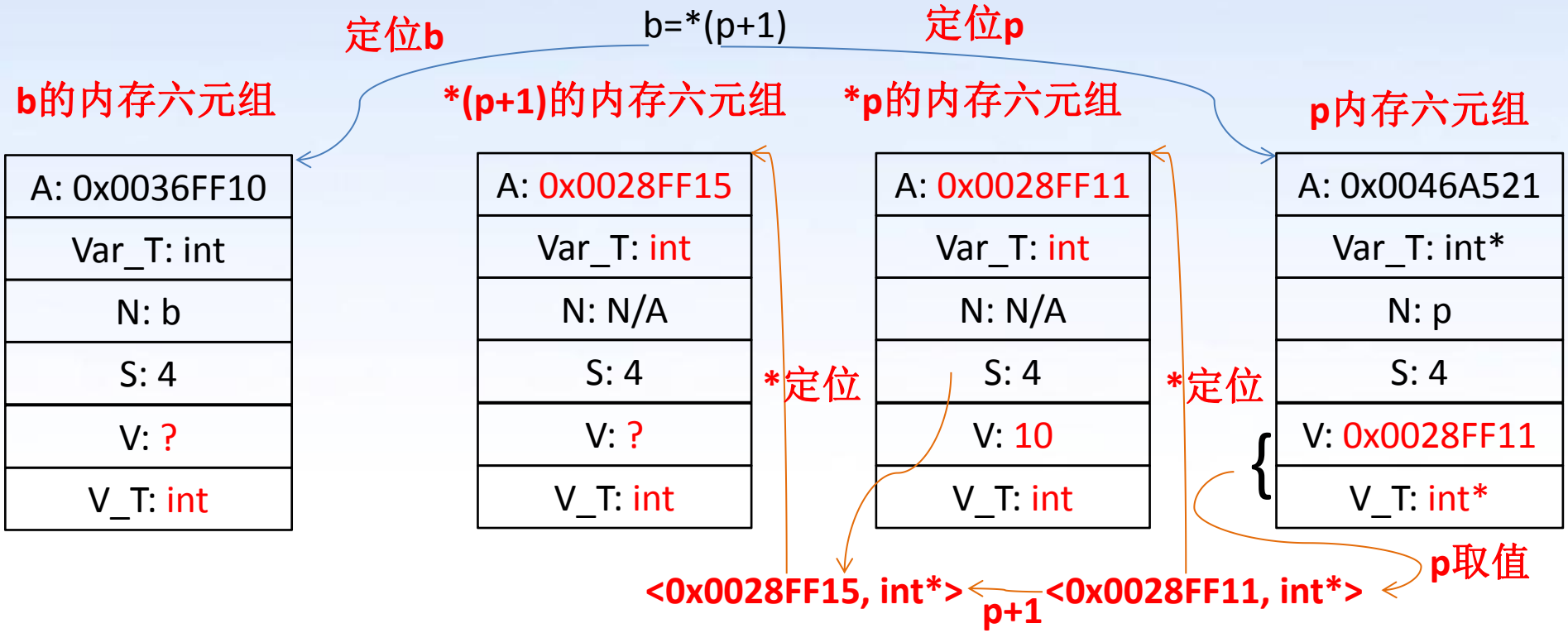
int b=*(p+1)发生了什么？



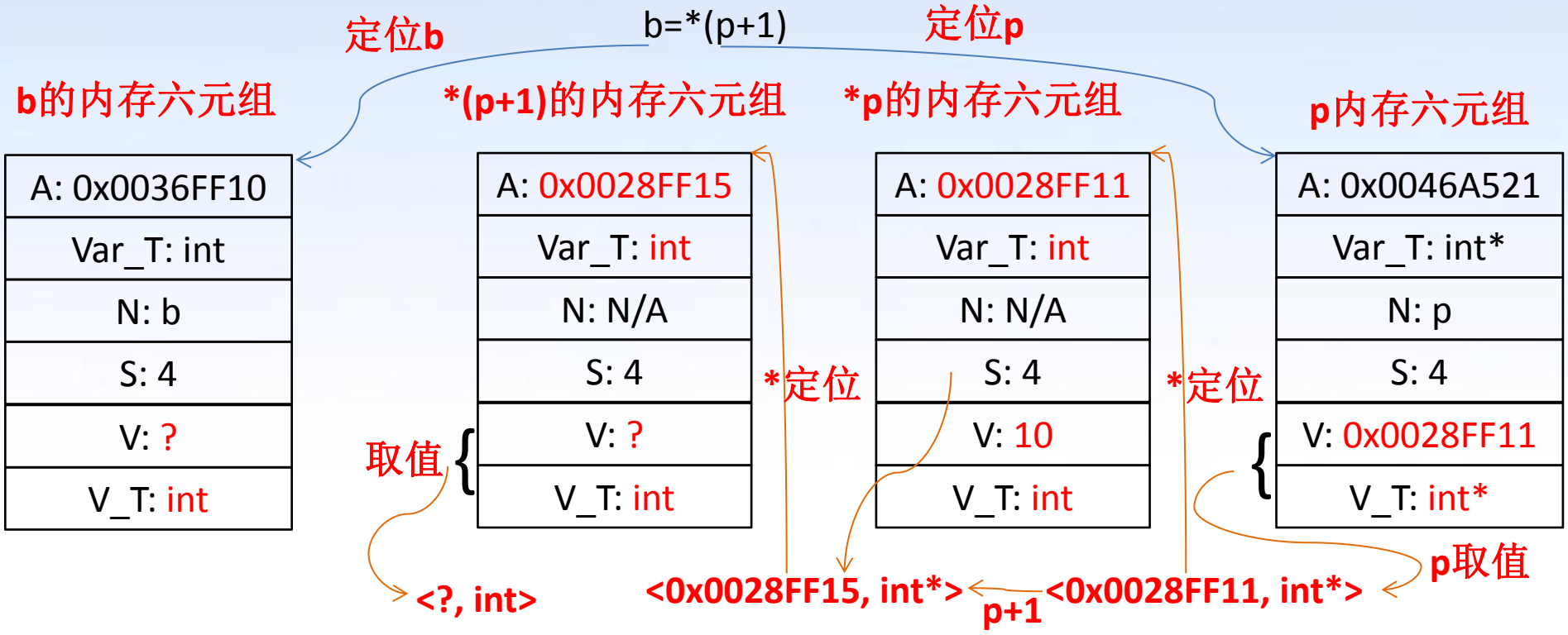
int b=*(p+1)发生了什么？



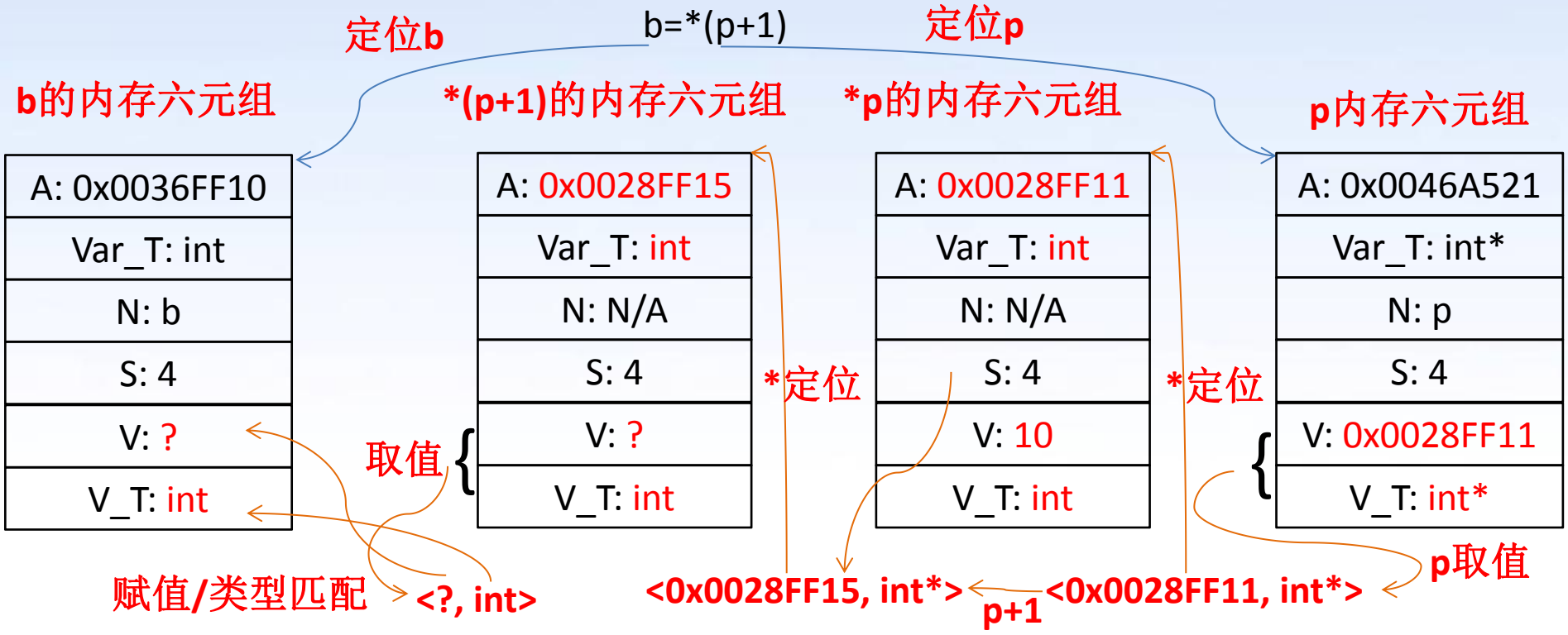
int b=*(p+1)发生了什么？



int b=*(p+1)发生了什么？



int b=*(p+1)发生了什么？



这里有什么问题？溢出的危害

b=*(p+1)

b的内存六元组

A: 0x0036FF10
Var_T: int
N: b
S: 4
V: ?
V_T: int

***(p+1)的内存六元组**

A: 0x0028FF15
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

取值

赋值/类型匹配 <?, int>

0x0028FF11是变量a所在内存的首地址

*(p+1)内存的起始位置是0x0028FF15

以0x0028FF15开始的内存是一块有效内存吗？

左值示例：*(p+2)=30，同样的问题

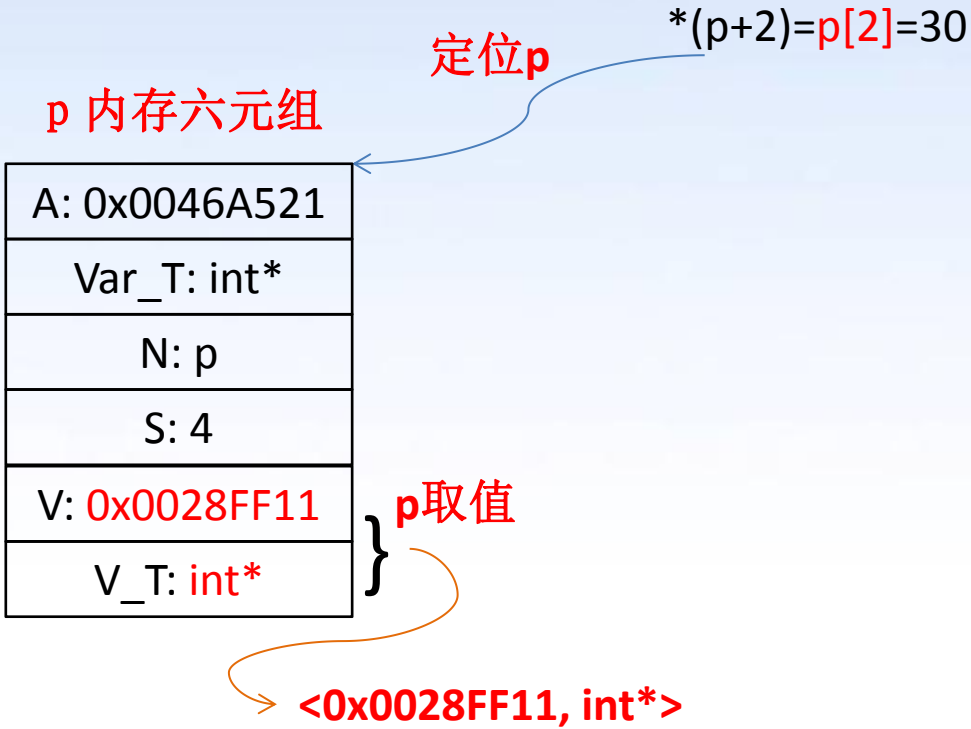
*(p+2)=p[2]=30

定位p

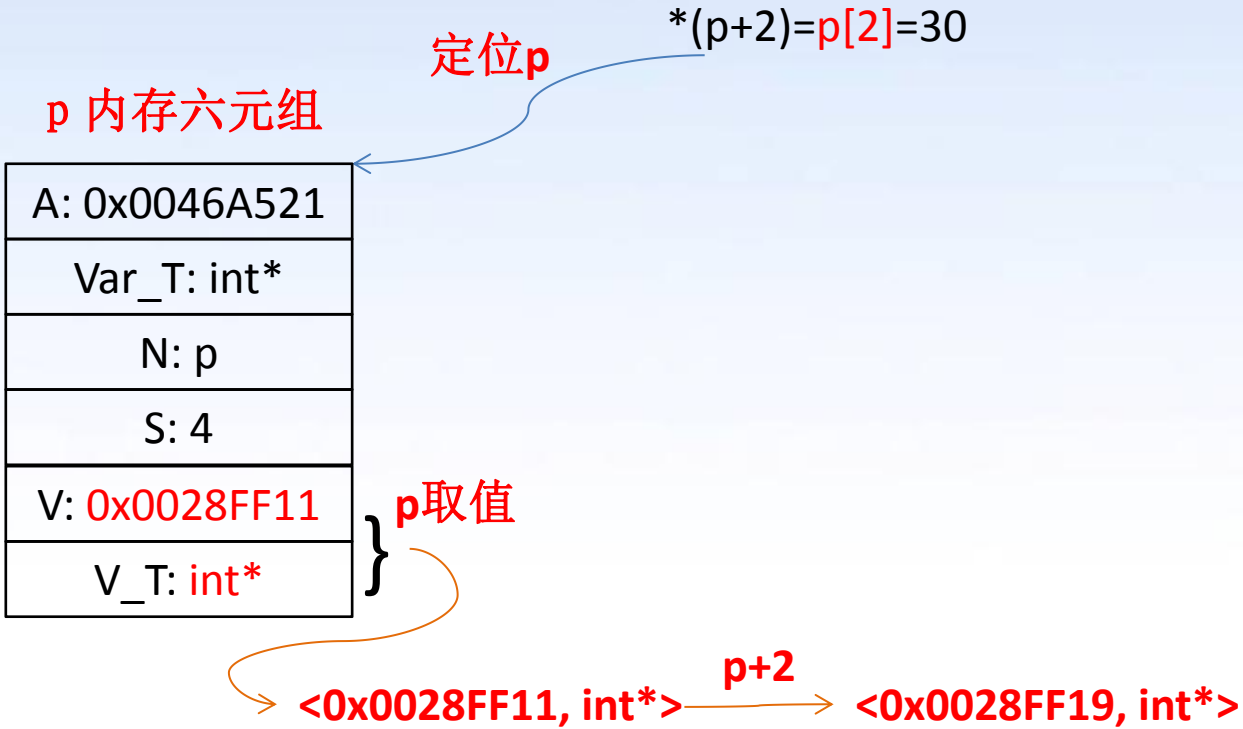
p 内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

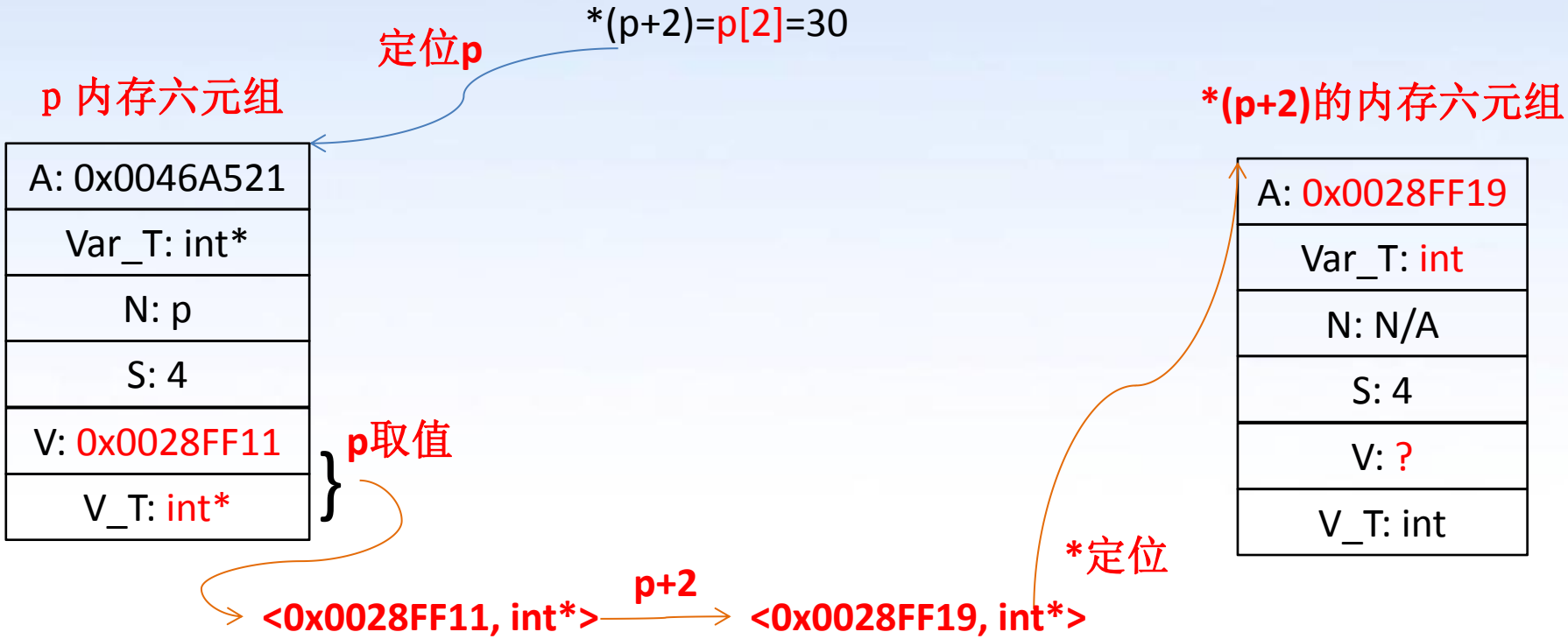
左值示例：*(p+2)=30，同样的问题



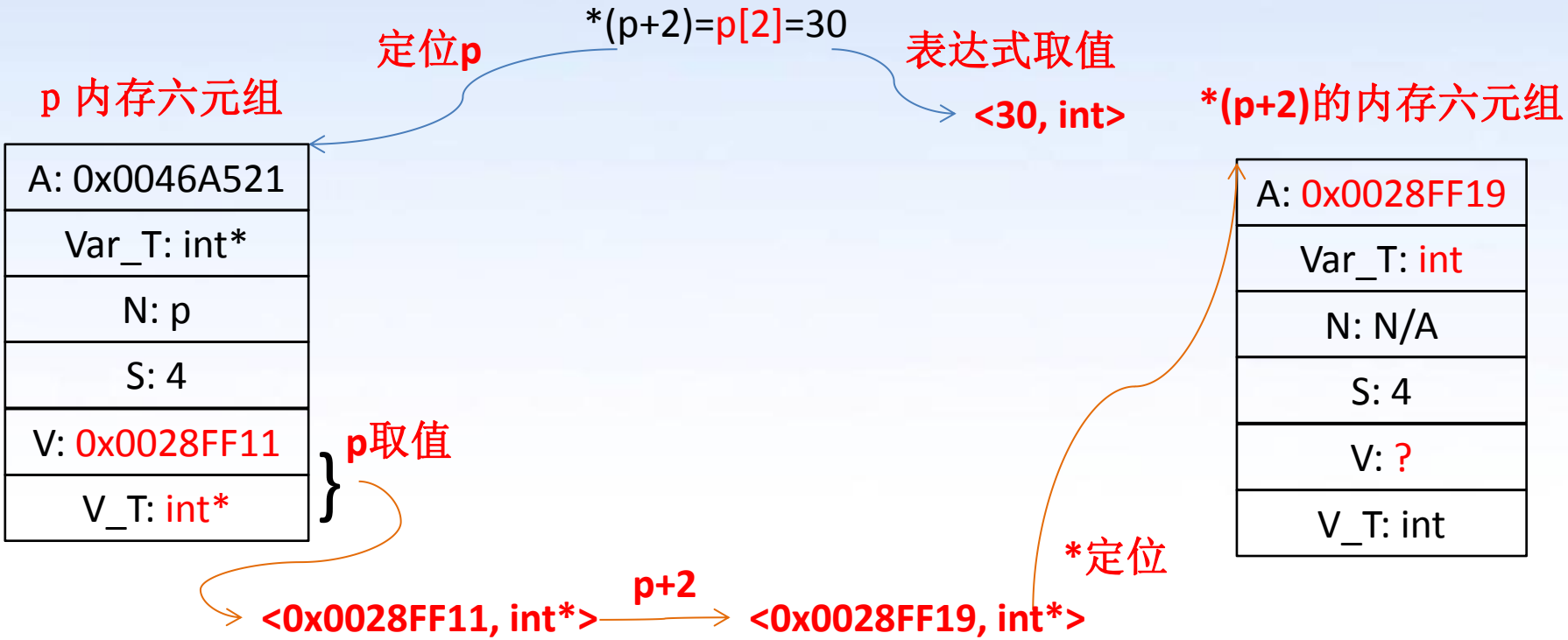
左值示例：*(p+2)=30，同样的问题



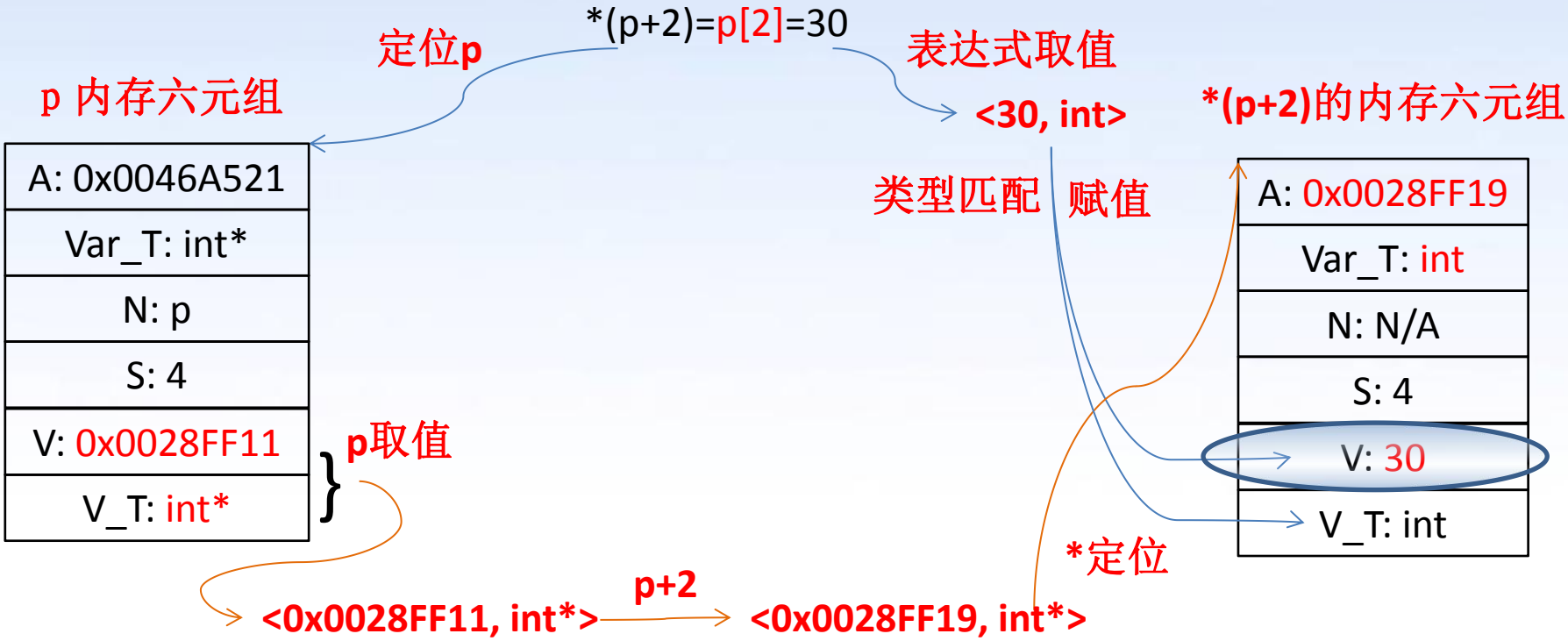
左值示例：*(p+2)=30，同样的问题



左值示例：*(p+2)=30，同样的问题



左值示例：*(p+2)=30，同样的问题



再来回顾一下指针到底是什么？

int* p;

1、我们首先看到int*，知道只是一个指针类型，其指向的变量类型为int
2、我们看到p，这是存储指针的4字节空间名称
3、这个p的value开始的空间是一个int
4、对p的Value指向的那个内存定位方法有*p或p[0]
对任何指针类型，*(p+n) = p[n]

p 内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

思考：为什么有p的Value和Value_Type能重构出一个int的内存描述

Tips：书上一般写int *p; 我们建议写成int* p，把int*写成一个整体，看成一个类型

指针指向的内存是数组吗？

给定一个int* p，可以通过偏移量随意进行内存访问p[n]，例如：
p[0], p[1], p[2], …., p[99], … ， 或
*p, *(p+1), *(p+2)… *(p+99),…

任何一个指针类型变量，蕴含着指向的那块内存为一个数组，元素为指针变量类型对应的变量类型

但大小未知

C语言指针的偏移访问灵活，但危险性也很大

两个指针相减是什么意思？

```
int* p;
int* q;

p – q = ?
```

1、相减的两个指针类型必须一致

2、假设p: <Value1, int*>, q: <Value2, int*>

p – q: <(Value1-Value2)/sizeof(*p), int>

思考题

1、int(*p)[2][3]， p+2=?
假设p的值是<0x0035AB11, int(*)[2][3]>

2、int (*r)[30], int (*t)[30], r-t=?
假设t的值是<0x0035AB11, int(*)[30], r的值是<0x0035AC01, int(*)[30]

思考题

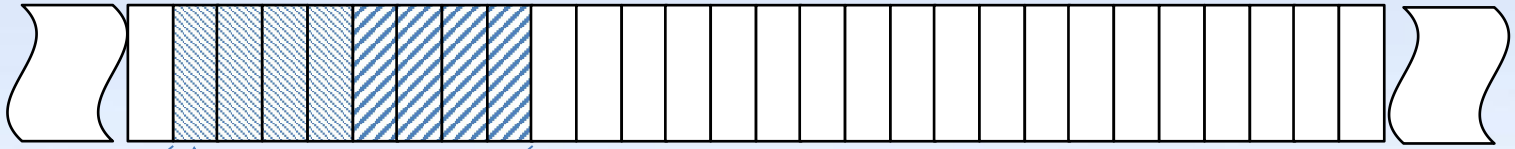
1、int(*p)[2][3]， p+2=?
假设p的值是<0x0035AB11, int(*)[2][3]>

2、int (*r)[30], int (*t)[30], r-t=?
假设t的值是<0x0035AB11, int(*)[30], r的值是<0x0035AC01, int(*)[30]

答案
1、p+2的值是<0x0035AB41, int(*)[2][3]>

2、r-t的值: <2, int>
(0x0035AC01-0x0035AB11)/sizeof(int[30]) = 2;

再来观察int e[2]

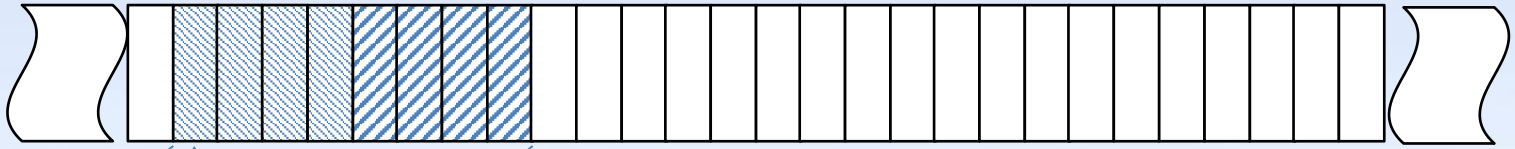


0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

思考1：size为什么是8？
思考2：Variable_Type是int[2]
思考3：value为什么是和Address一样？
思考4：Type为什么是int*

&e的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

} &e

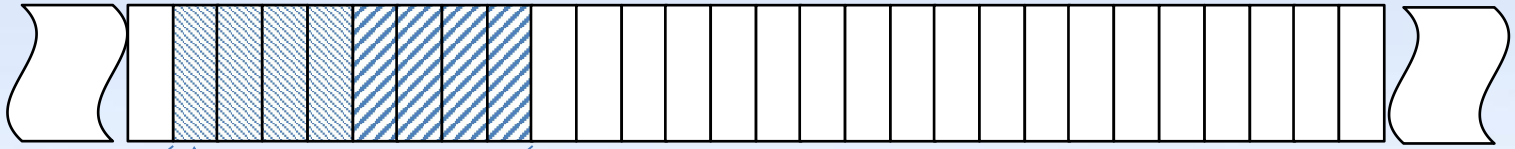
<0x0076AB11, int(*)[2]>

如何定义一个变量x=&e

x的类型必须是int(*)[2]

int (*x)[2] = &e

sizeof(e)的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

sizeof → <8, size_t>

如何定义一个变量x=sizeof(e)

x的类型必须是size_t

size_t x = sizeof(e)

再看sizeof()

int e[2]，以下表达式返回值是什么？

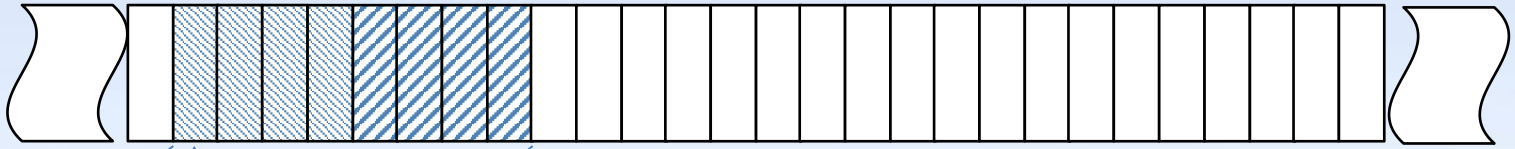
sizeof(int[2]) vs. sizeof(e) vs. sizeof(e+1)

假设变量e的内存首地址为0x0076AB11

sizeof(内容有三类)

sizeof(int[2])：8 获得一个变量类型的大小
sizeof(e)：8 获得一个变量内存的大小
sizeof(e+1)：4 获得一个表达式返回值变量类型的大小
e+1: <0x0076AB15, int*>

e的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

取值

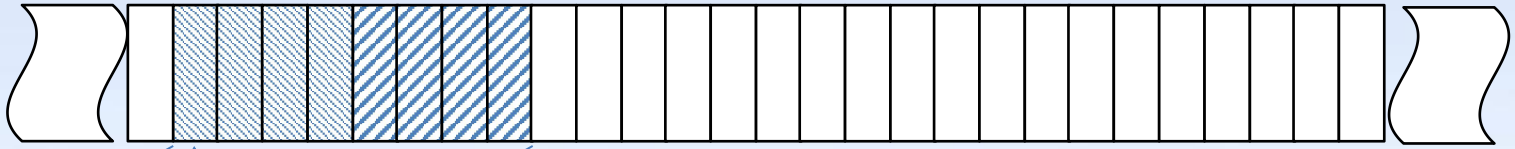
<0x0076AB11, int*>

如何定义一个变量x=e

x的类型必须是int*

int* x = e

e[0]到底是什么？

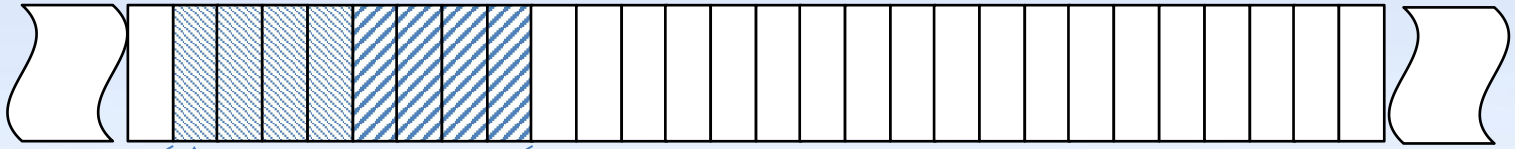


0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

e[0]到底是什么？



0x0076AB11

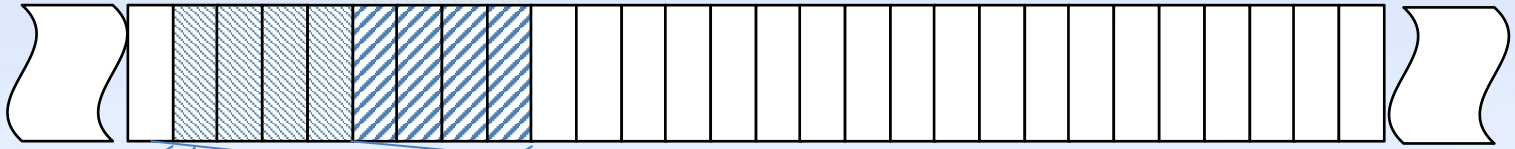
A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

取值

<0x0076AB11, int*>

e的内存六元组

e[0]到底是什么？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

取值

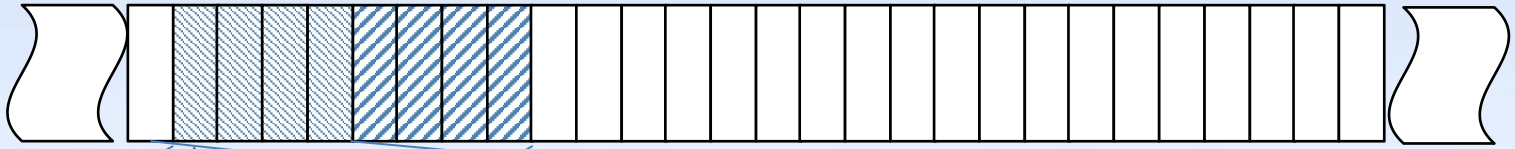
<0x0076AB11, int*>

*定位

A: 0x0076AB11
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

*e的内存六元组

e[0]到底是什么？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

e[0]=*(e+0)
e的value+sizeof(*e)*0
e的type保持不变

取值

<0x0076AB11, int*>

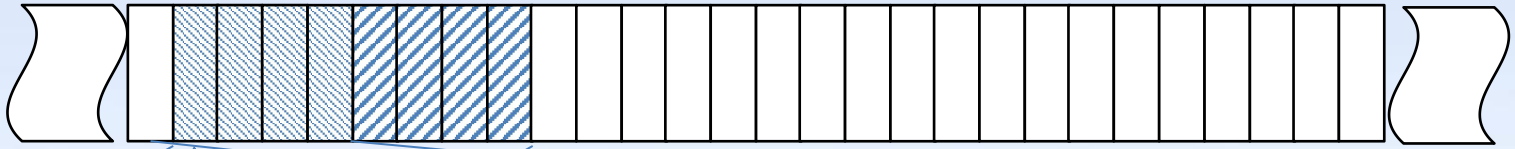
e+0 <0x0076AB11, int*>

*定位

A: 0x0076AB11
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

*e的内存六元组

e[0]到底是什么？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

e[0]=*(e+0)
e的value+sizeof(*e)*0
e的type保持不变

取值

<0x0076AB11, int*>

e+0 <0x0076AB11, int*>

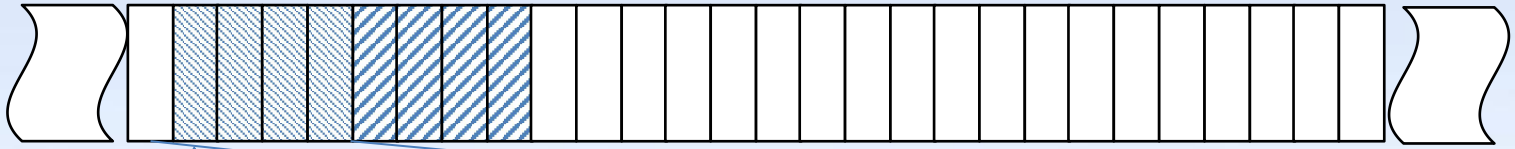
*定位

*定位

A: 0x0076AB11
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

e/(e+0)的内存六元组

&(e[0])、sizeof(e[0])和e[0]



0x0076AB11

&(e[0]): <0x0076AB11, int*>

sizeof(e[0]): <4, size_t>

e[0]: <?, int>
因为int e[2]; 没有初始化这块内存
如果int e[2] = {1, 2};
e[0]: <1, int>

A: 0x0076AB11
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

e/(e+0)的内存六元组

e=e+1和e++为什么会报错

```
int main()
{
    int e[2];

    e = e + 1;

    return 0;
}
```

```
=== Build file: "no target" in "no project" (compiler: unknown) ===
In function 'main':
error: assignment to expression with array type
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

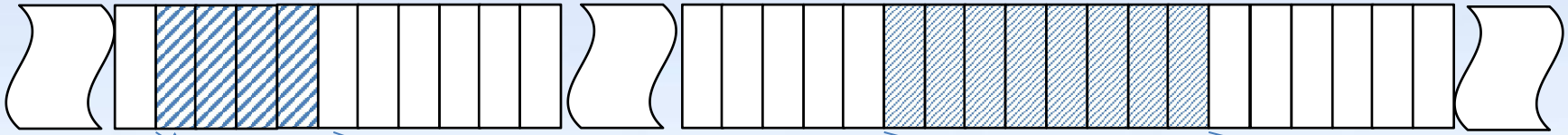
数组的Value一定是指向数组第一个元素的地址编号

对于数组e来说，V的值必须和A相等

因此，e并不是一个常量，它的值不能更改是语法限制的

对常量修改的错误应该是：assignment of read-only variable 'e'

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

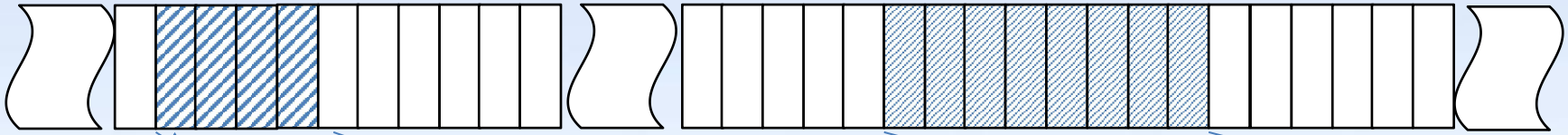
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: ?
V_T: int*

p的内存六元组

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0076AB11
V_T: int*

p的内存六元组

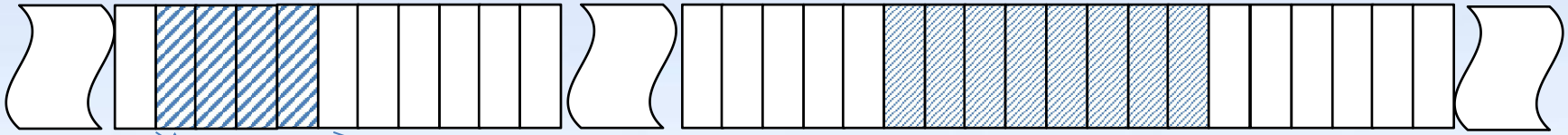
p=e;

<0x0076AB11, int*>

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

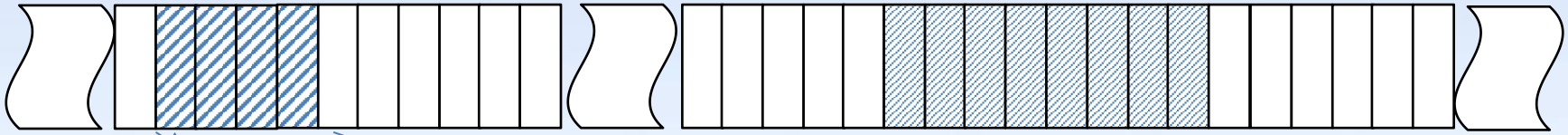
```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0076AB11
V_T: int*

p的内存六元组

p取值 <0x0076AB11, int*>

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

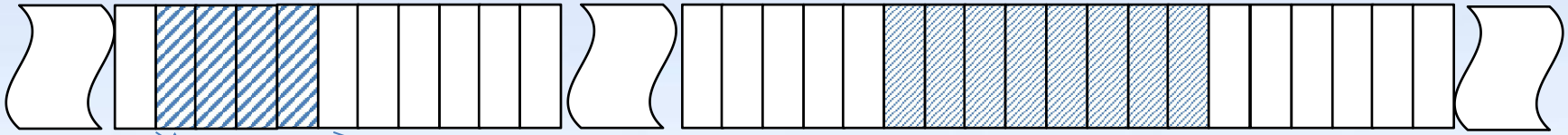
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0076AB11
V_T: int*

p的内存六元组

p取值 <0x0076AB11, int*>

p+1 <0x0076AB15, int*>

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0076AB15
V_T: int*

p的内存六元组

<0x0076AB15, int*>

p=p+1

p+1

p取值

<0x0076AB11, int*>

修改的是p的值
没有修改e的值

特殊的数组：字符串

```
int main()
{
    char* p = "hello";
    char* q = "hello";
    char r[] = "hello";
    char s[] = {'h', 'e', 'l', 'l', 'o', '\0'};

    printf("p=%s, q=%s, r=%s, s=%s\n", p, q, r, s);
    printf("p=%x, q=%x, r=%x, s=%x\n", p, q, r, s);

    return 0;
}
```

```
选择C:\Users\wahaha\Desktop\test\a.exe
p=hello, q=hello, r=hello, s=hello
p=404000, q=404000, r=61fe0a, s=61fe04

Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

1、char s[]={‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};是标准的数组变量初始化

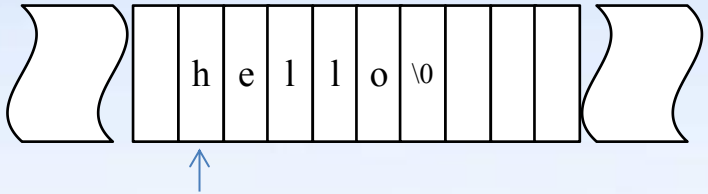
2、char r[]=“hello”;也是数组变量初始化，此时等于{‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};

3、char* p=“hello”;是将字符串”hello”的表示值赋给ptr，字符串是一个特殊的字符数组，其表示值是这个数组的首地址

注意：p和q的值是一样的　　思考：strcpy(p, “world”);会出错吗？

特殊的数组：字符串（续）

```
int main()
{
    char* p = "hello";
    char* q = "hello";
    char r[] = "hello";
    char s[] = {'h', 'e', 'l', 'l', 'o', '\0'};

    printf("p=%s, q=%s, r=%s, s=%s\n", p, q, r, s);
    printf("p=%x, q=%x, r=%x, s=%x\n", p, q, r, s);

    strcpy(p, "world");

    return 0;
}
```



0x00404000

p指向的这块内存是文本常量区（只读）

```
C:\Users\wahaha\Desktop\test\a.exe
p=hello, q=hello, r=hello, s=hello
p=404000, q=404000, r=61fe0a, s=61fe04

Process returned -1073741819 (0xC0000005)   execution time : 0.261 s
Press any key to continue.
```

再思考：赋值的对象到底是什么？

```
int main()
{
    int a = 10;

    a++ = 11;
}
```

赋值是针对一块内存操作，内存定位的方法只有
1）用变量名；2）用*+地址的方法
不能跟任何运算符结合，否则就变成了取值运算操作

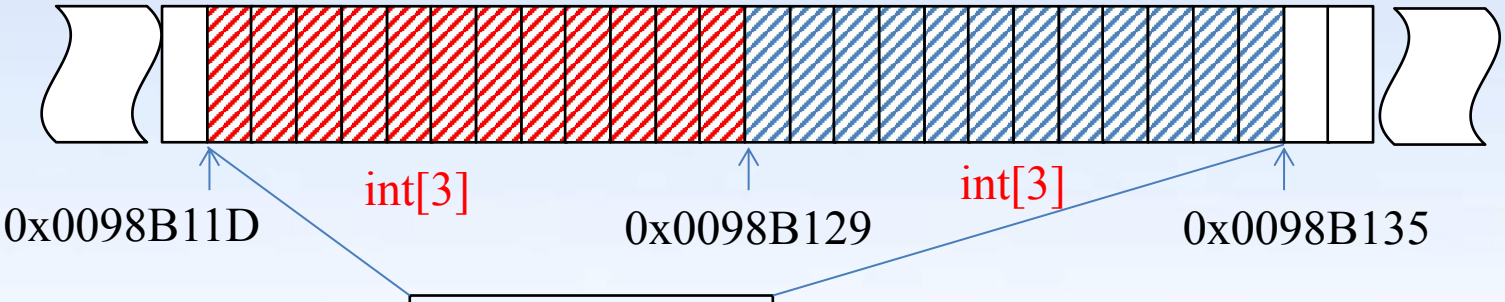
a = 10; a定位了内存，√
a++ = 11;结合运算符，a++变成了<10, int> ╳

char str2[6]; str2="hello"; str2确实定位了内存，为什么不行？
数组类型不可以不是lvalue错误，而是assignment错误

```
error: lvalue required as left operand of assignment
=== Build failed: 1 error(s), 0 warning(s) (0 minute(...
```

```
error: assignment to expression with array type
=== Build failed: 1 error(s), 0 warning(s) (0 minute(...
```

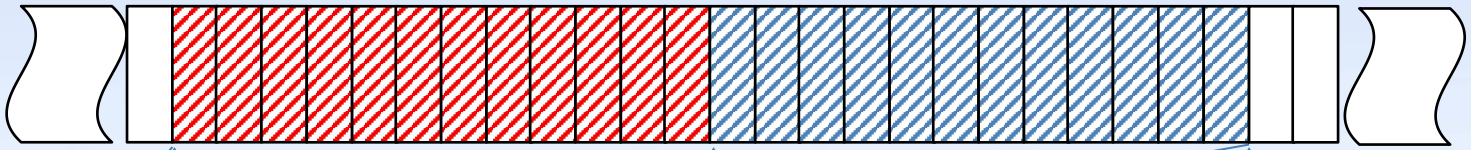
再来观察：int g[2][3]



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

思考1：size为什么是24？
思考2：Variable_Type是int[2][3]
思考3：Value为什么是和Address一样？
思考4：Value Type为什么是int(*)[3]

&g的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

&

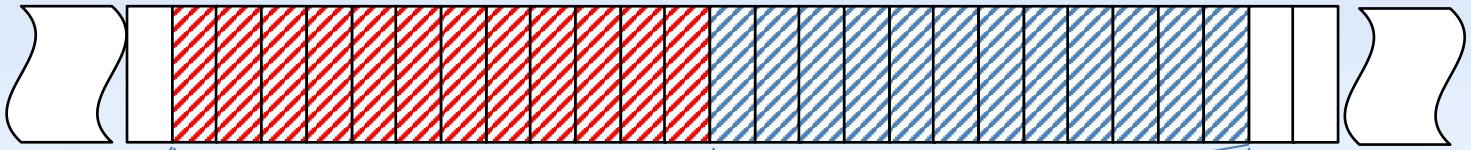
<0x0098B11D, int(*)[2][3]>

如何定义一个变量y=&g

y的类型必须是int(*)[2][3]

int (*y)[2][3] = &g

sizeof(g)的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

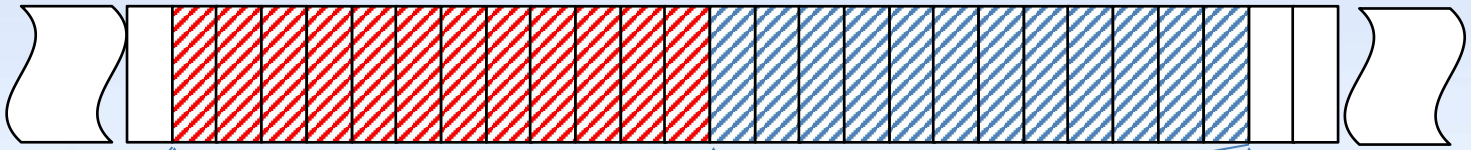
sizeof → <24, size_t>

如何定义一个变量y=sizeof(g)

y的类型必须是size_t

size_t y = sizeof(g)

g的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

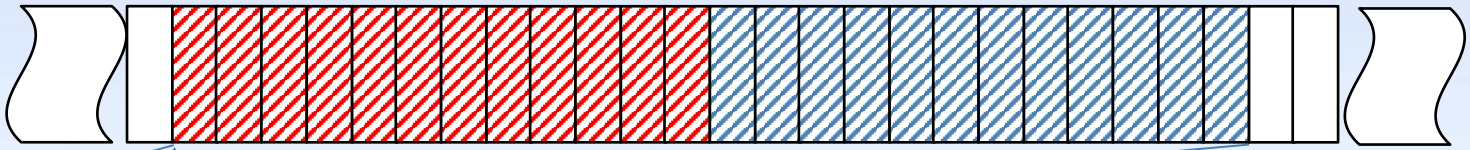
取值 <0x0098B11D, int(*)[3]>

如何定义一个变量y=g

y的类型必须是int(*)[3]

int (*y)[3] = g

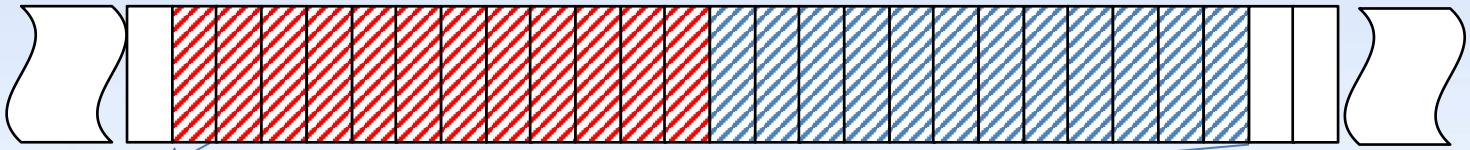
g[1]到底是什么？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

g的内存六元组

g[1]到底是什么？



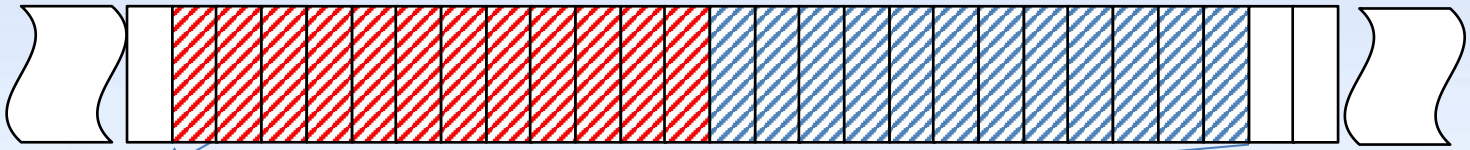
A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

取值

<0x0098B11D, int(*)[3]>

g的内存六元组

g[1]到底是什么？



0x0098B11D　　int[3]　　int[3]

A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

g的内存六元组

取值

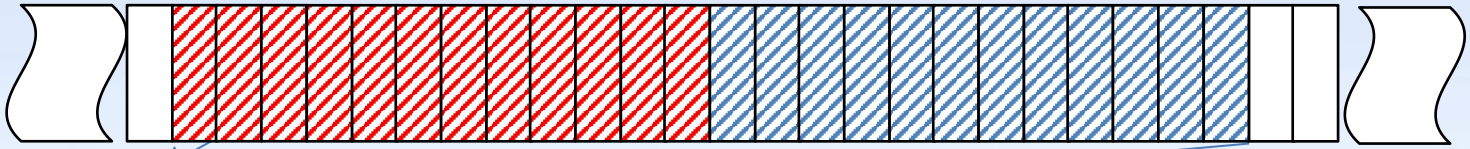
<0x0098B11D, int(*)[3]>

*定位

A: 0x0098B11D
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B11D
V_T: int*

*g的内存六元组

g[1]到底是什么？



0x0098B11D int[3] int[3]

A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

g的内存六元组

g[1]=*(g+1)
g的value+sizeof(*g)*1
g+1的type保持不变

取值

<0x0098B11D, int(*)[3]>

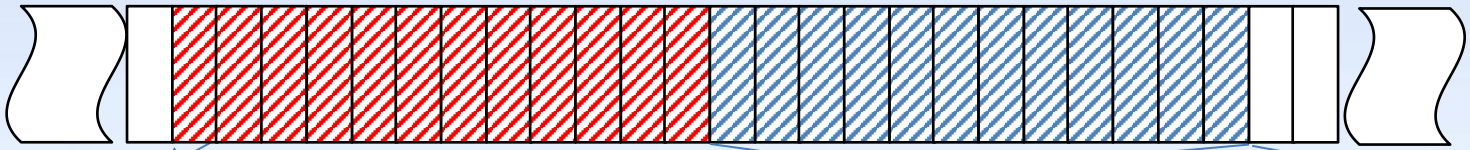
*定位

g+1<0x0098B129, int(*)[3]>

A: 0x0098B11D
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B11D
V_T: int*

*g的内存六元组

g[1]到底是什么？



0x0098B11D int[3] int[3]

A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

g的内存六元组

g[1]=*(g+1)
g的value+sizeof(*g)*1
g+1的type保持不变

取值

<0x0098B11D, int(*)[3]> *定位

*定位

g+1<0x0098B129, int(*)[3]>

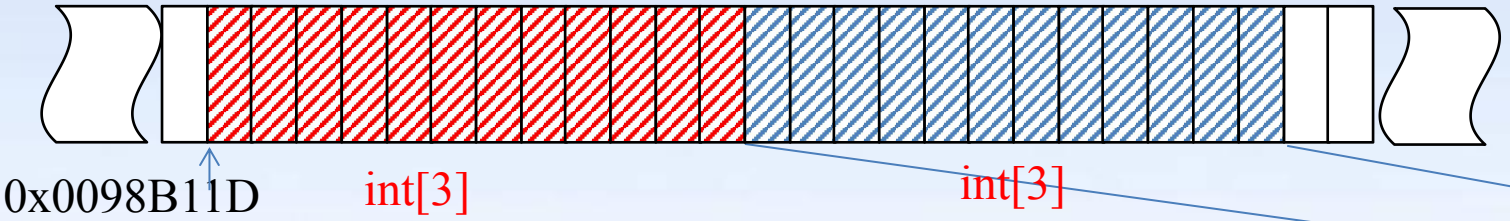
A: 0x0098B11D
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B11D
V_T: int*

*g的内存六元组

A: 0x0098B129
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B129
V_T: int*

*(g+1)的内存六元组

&(g[1])、sizeof(g[1])和g[1]



&(g[1]): < 0x0098B12F, int(*)[3]>

sizeof(g[1]): <12, size_t>

g[1]: <0x0098B12F, int*>

A: 0x0098B129
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B129
V_T: int*

*(g+1)/g[1]的内存六元组

数组变量作为函数参数

```
void func1(int* pe)
{
    //do something
}

void func2(int (*pg)[3])
{
    //do something
}

int main()
{
    int e[2];
    int g[2][3];

    func1(e);
    func2(g);

    return 0;
}
```

main函数里面：

e: <0x0076AB11, int*>

g: <0x0098B11D, int(*)[3]>

func1和func2函数里面：

pe: <0x0076AB11, int*>

pg: <0x0098B11D, int(*)[3]>

C语言参数传递的机制是：Pass By Value

形参中：int[]= int*，int[][3]= int(*)[3]

数组中第一维信息的丢失是C语言
所有数组类型变量内存的取值机制造成的

进一步思考二维数组的形参形式

```
void func(int g[][3])
{
    // do something
}

int main()
{
    int g[2][3] = {0};
    func(g);
    return 0;
}
```

```
void func(int* g, int row, int col)
{
    // do something
}

int main()
{
    int g[2][3] = {0};
    func((int*)g, 2, 3);
    return 0;
}
```

```
void func(int* g, int size)
{
    // do something
}

int main()
{
    int g[2][3] = {0};
    func((int*)g, 6);
    return 0;
}
```

使用int g[][3]作为形参，数字3需要写在参数上，扩展性较弱

int row, int col vs. int size

int** pg当形参能行吗？

```
void func(int** pg)
{
    pg[0][0] = 1;
}

int main()
{
    int g[2][3] = {0};
    func((int**)g);
    return 0;
}
```

注意：g[2][3]={0}, 假设g对应内存首地址为0x0098B11D

pg[0][0] = 1或**pg=1会有什么问题？

int** pg当形参能行吗？

```
void func(int** pg)
{
    pg[0][0] = 1;
}

int main()
{
    int g[2][3] = {0};
    func((int**)g);
    return 0;
}
```

注意g[2][3]={0}
g对应内存首地址
0x0098B11D

pg的内存六元组

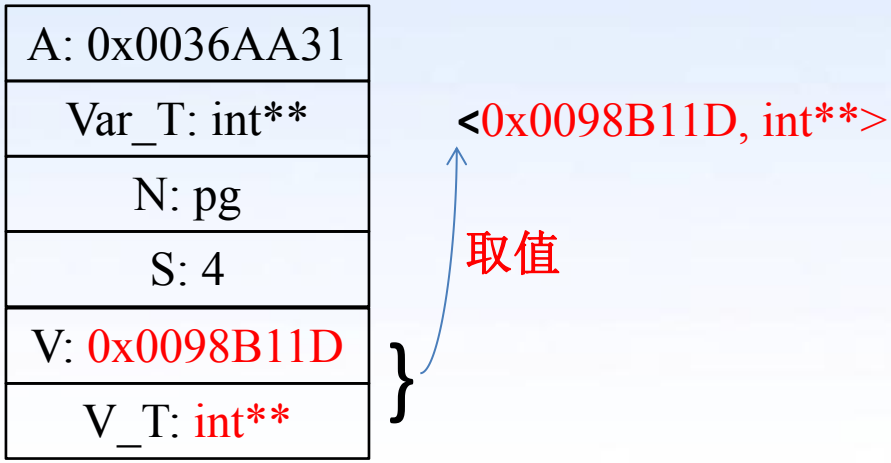
A: 0x0036AA31
Var_T: int**
N: pg
S: 4
V: 0x0098B11D
V_T: int**

int** pg当形参能行吗？

```
void func(int** pg)
{
    pg[0][0] = 1;
}

int main()
{
    int g[2][3] = {0};
    func((int**)g);
    return 0;
}
```

注意g[2][3]={0}
g对应内存首地址
0x0098B11D

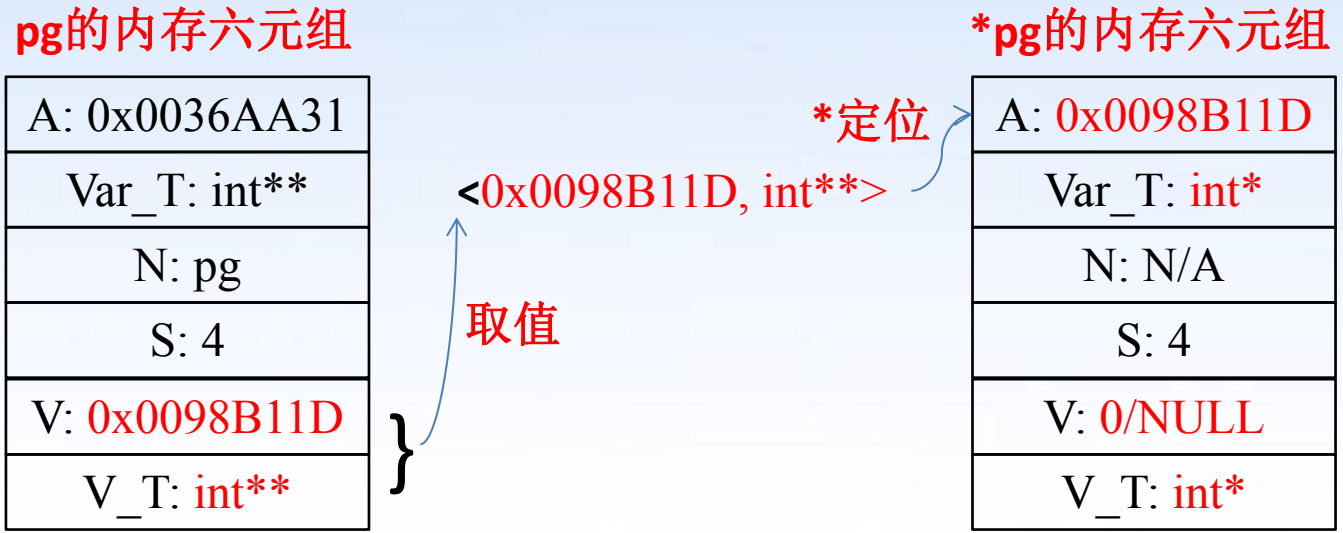
pg的内存六元组



int** pg当形参能行吗？

```
void func(int** pg)
{
    pg[0][0] = 1;
}

int main()
{
    int g[2][3] = {0};
    func((int**)g);
    return 0;
}
```

注意g[2][3]={0}
g对应内存首地址
0x0098B11D



int** pg当形参能行吗？

```
void func(int** pg)
{
    pg[0][0] = 1;
}

int main()
{
    int g[2][3] = {0};
    func((int**)g);
    return 0;
}
```

注意g[2][3]={0}
g对应内存首地址
0x0098B11D

pg的内存六元组

A: 0x0036AA31
Var_T: int**
N: pg
S: 4
V: 0x0098B11D
V_T: int**

取值

<0x0098B11D, int**>

*定位

*pg的内存六元组

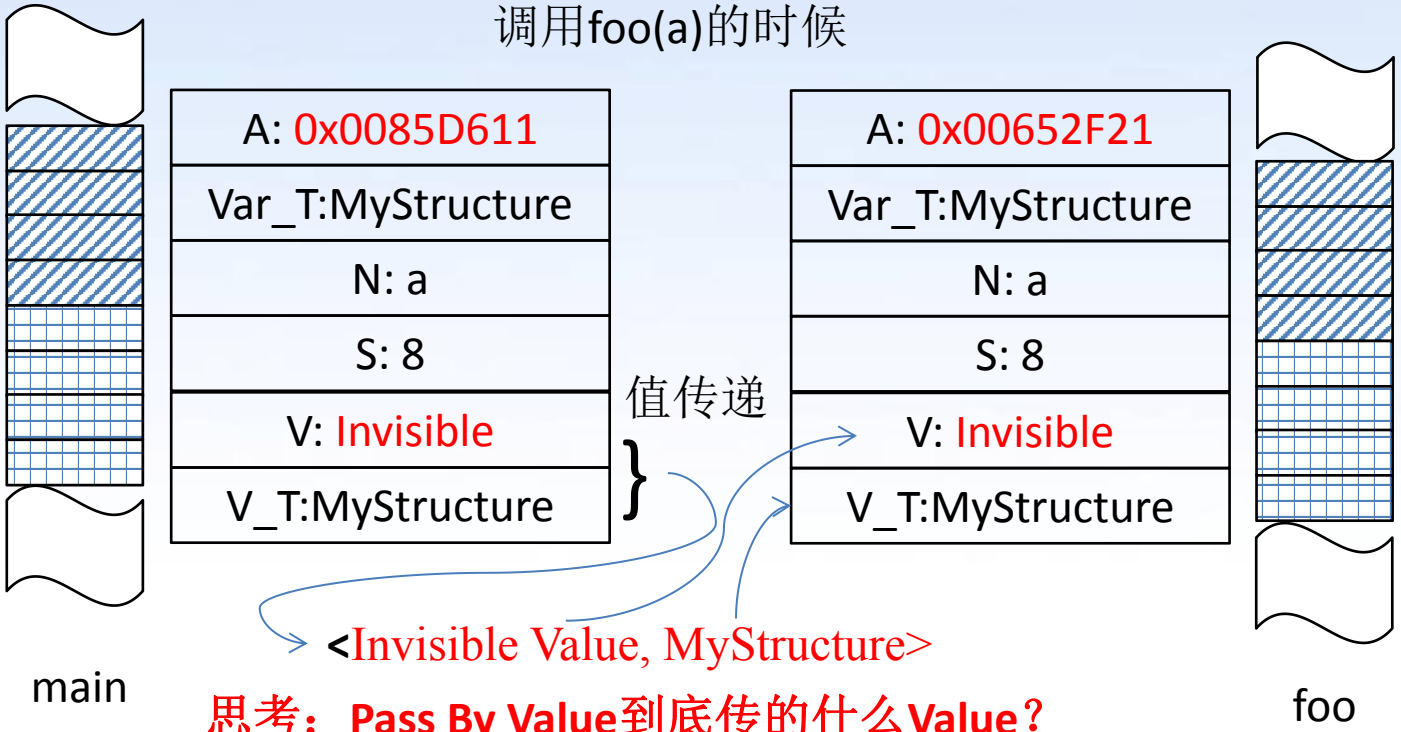
A: 0x0098B11D
Var_T: int*
N: N/A
S: 4
V: 0/NULL
V_T: int*

**pg

*pg的Value=0怎么来的？
会导致后续什么问题？

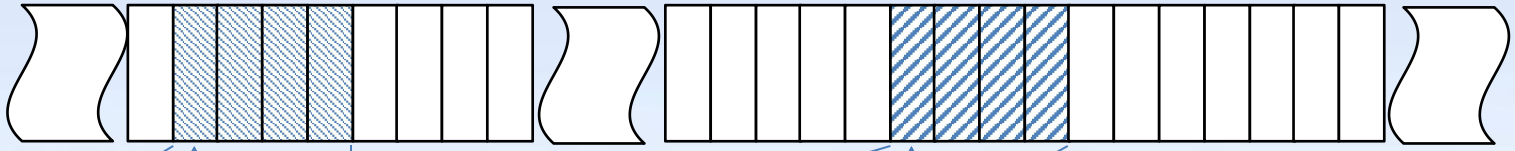
结构体作为函数参数

```
typedef struct _MyStructure{
    int a;
    int b;
} MyStructure;

int foo(MyStructure a)
{
    a.a++;
}

int main()
{
    MyStructure a;

    a.a = a.b = 0;

    foo(a);

    printf("a.a=%d\n", a.a);

    return 0;
}
```



思考：**Pass By Value到底传的什么Value？**

再回顾：变量声明 vs. malloc



0x0028FF11

A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int

a的内存六元组

0x00351728

A: 0x00351728
Var_T: N/A
N: N/A
S: 4
V: N/A
V_T: N/A

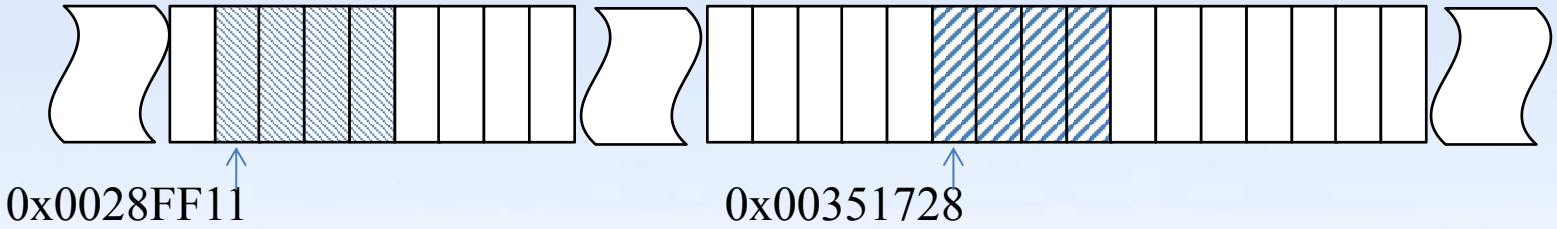
malloc(4)分配的内存六元组

int a vs. malloc(4)

访问a对应内存
1、通过变量名a
2、通过首地址0x0028FF11

访问malloc(4)对应内存
1、变量名？
2、通过首地址0x00351728

首地址访问：变量声明 vs. malloc



int a=36; int* p = &a;
p: <0x0028FF11, int*>

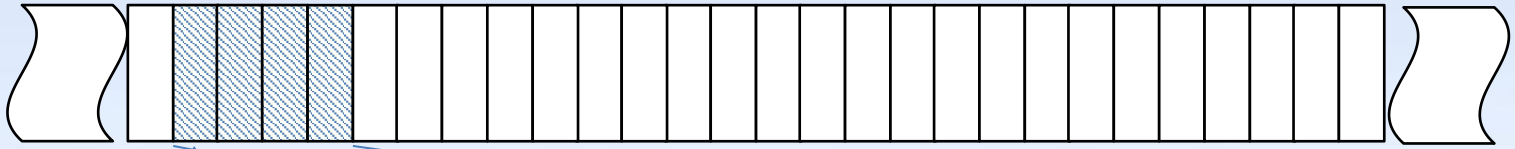
可以通过*p访问这块内存

void* q = malloc(4);
q: <0x00351728, void*>

不能通过*q访问这块内存

q无法还原0x00351728开始的内存，因为q的类型是void，*q还原的Variable_Type是void，无法确定size

通过首地址访问变量所在内存



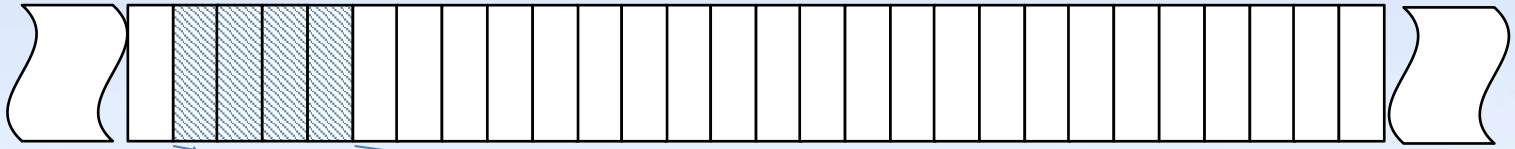
0x0028FF11

```
int* p = &a;
p: <0x0028FF11, int*>
```

A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 36
V_T: int

*p的内存六元组

通过首地址访问变量所在内存（续）



0x0028FF11

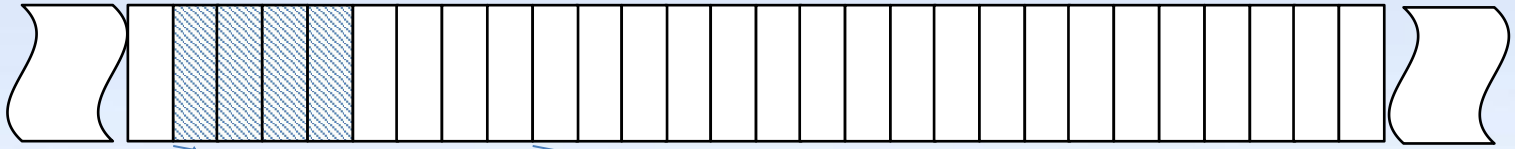
```
int* p = &a;
p: <0x0028FF11, int*>

float* f = (float*)&a;
f: <0x0028FF11, float*>

printf("*f = %f\n", *f);
```

A: 0x0028FF11
Var_T: float
N: N/A
S: 4
V: 0.000000
V_T: float

*f的内存六元组

这样访问合法吗？
有什么问题？

通过首地址访问变量所在内存（续）



0x0028FF11

```
int* p = &a;
p: <0x0028FF11, int*>
```

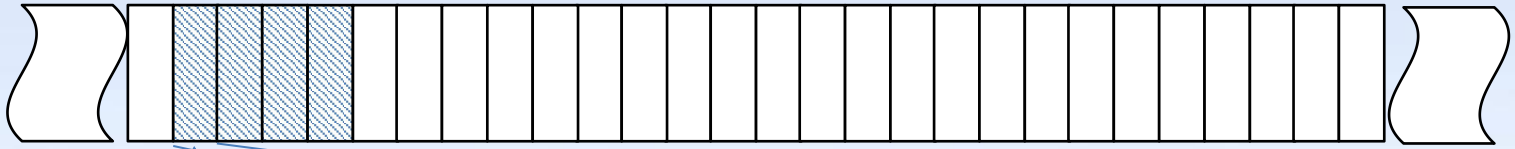
```
double* d = (double*)&a;
d: <0x0028FF11, double*>
```

A: 0x0028FF11
Var_T: double
N: N/A
S: 8
V: ?
V_T: double

*d的内存六元组

这样访问合法吗？
有什么问题？

通过首地址访问变量所在内存（续）



0x0028FF11

int* p = &a; (注意a=36)
p: <0x0028FF11, int*>

char* c = (char*)&a;
c: <0x0028FF11, char*>

A: 0x0028FF11
Var_T: char
N: N/A
S: 1
V:'$'
V_T: char

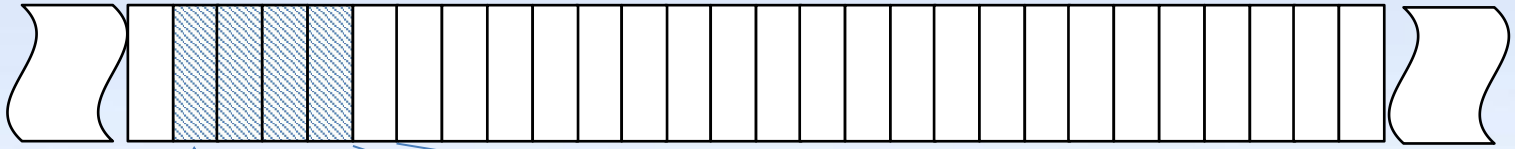
*c/c[0]的内存六元组

'$'的ASCII码是36

变量c的类型为char*，蕴含着c的值指向的那块内存为一个数组，元素为char

c[0], c[1], c[2], c[3]

通过首地址访问变量所在内存（续）



0x0028FF11

int* p = &a;
p: <0x0028FF11, int*>

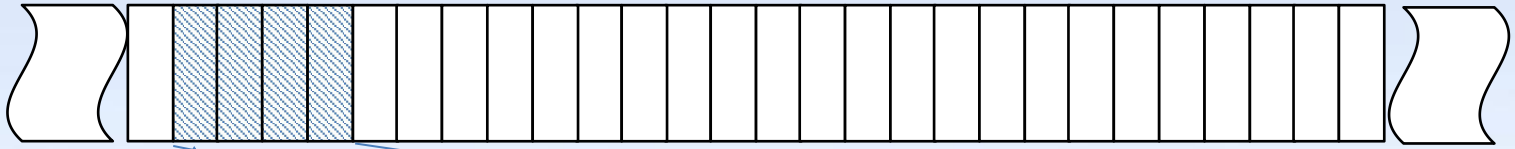
char* c = (char*)&a;
c: <0x0028FF11, char*>

A: 0x0028FF11
Var_T: char
N: N/A
S: 1
V: ?
V_T: char

*(c+4)/c[4]的内存六元组

c[4]语法上可以吗？
会带来什么问题？

通过首地址访问变量所在内存（续）



0x0028FF11

int* p = &a;
p: <0x0028FF11, int*>

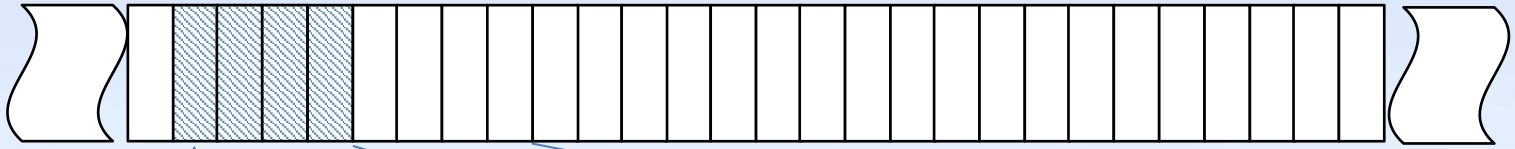
char (*r)[4] = (char(*)[4])&a;
r: <0x0028FF11, char(*)[4]>

A: 0x0028FF11
Var_T: char[4]
N: N/A
S: 4
V: 0x0028FF11
V_T: char*

*r/r[0]的内存六元组

r的类型为char(*)[4]，蕴含着指向的那块内存为一个数组，元素为char[4]

通过首地址访问变量所在内存（续）



0x0028FF11

int* p = &a;
p: <0x0028FF11, int*>

char (*r)[4] = (char(*)[4])&a;
r: <0x0028FF11, char(*)[4]>

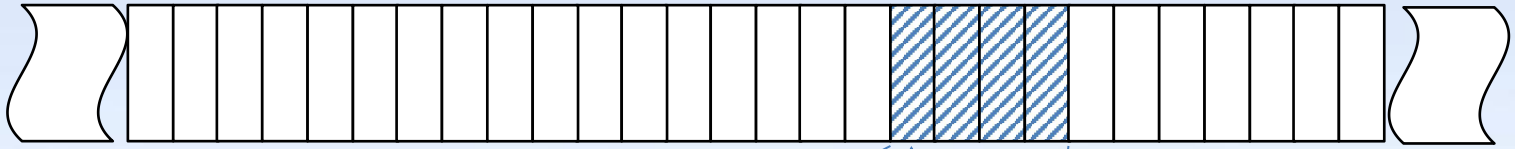
A: 0x0028FF15
Var_T: char[4]
N: N/A
S: 4
V: 0x0028FF15
V_T: char*

*(r+1)/r[1]的内存六元组

r[1]语法上可以吗？
会带来什么问题？

char* c = (char*)&a;
char (*r)[4] = (char(*)[4])&a;
区别在哪？

访问malloc分配的内存



0x00351728

A: 0x00351728
Var_T: N/A
N: N/A
S: 4
V: N/A
V_T: N/A

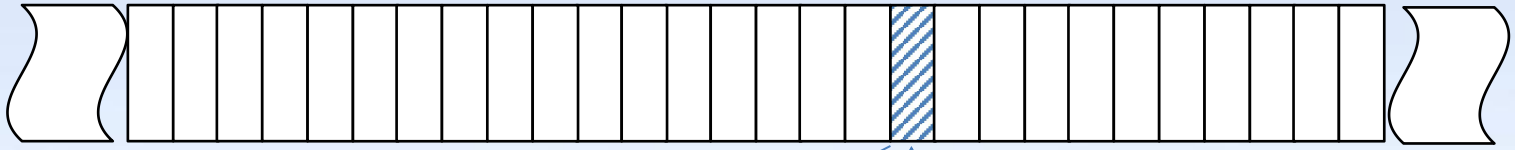
malloc(4)分配的的内存六元组

malloc(4);

这块内存没有变量名称

无法通过变量名进行定位使用

访问malloc分配的内存



0x00351728

void* p = malloc(4);

p: <0x00351728, void*>

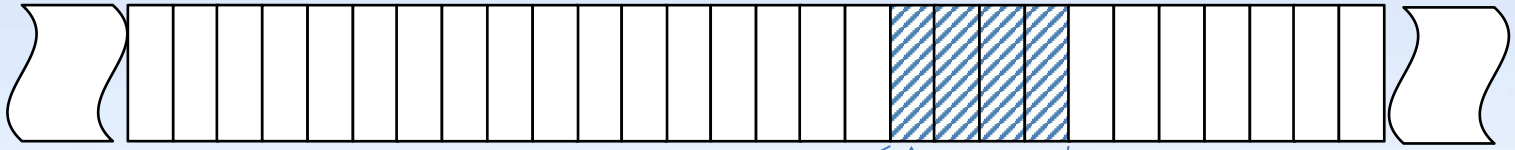
*p定位

A: 0x00351728
Var_T: N/A
N: N/A
S: N/A
V: N/A
V_T: N/A

<0x00351728, void*>不能有效识别

*p无法定位有效内存

访问malloc分配的内存



0x00351728

int* p = (int*)malloc(4);

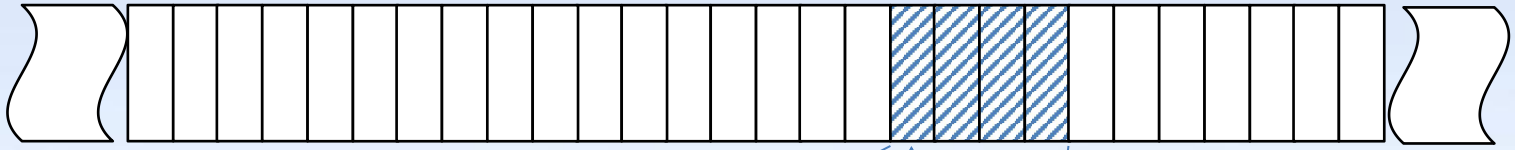
p: <0x00351728, int*>

*p定位

A: 0x00351728
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

*p的内存六元组

访问malloc分配的内存



0x00351728

float* f = (float*)malloc(4);

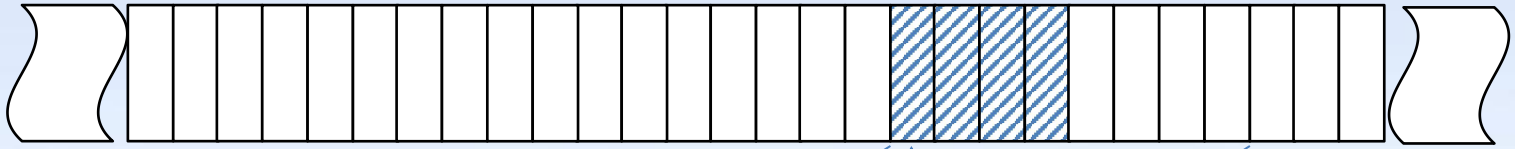
f: < 0x00352A18, float*>

*f定位

A: 0x00351728
Var_T: float
N: N/A
S: 4
V: ?
V_T: float

*f的内存六元组

访问malloc分配的内存（续）



0x00351728

double* d = (double*)malloc(4);

d: < 0x0039AA21, double*>

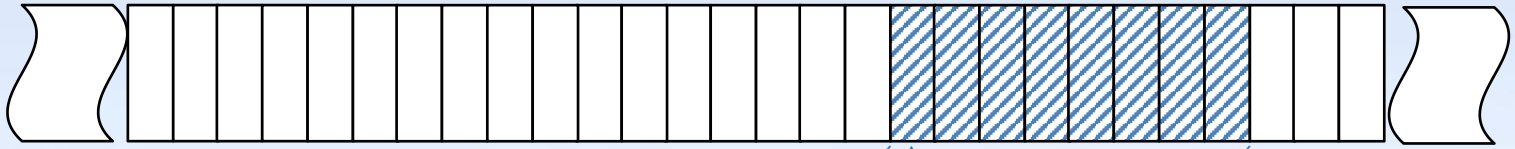
*d定位

A: 0x00351728
Var_T: double
N: N/A
S: 8
V: ?
V_T: double

*d的内存六元组

有什么问题？

访问malloc分配的内存（续）



0x00351728

double* d = (double*)malloc(sizeof(double));

d: <0x0039AA21, double*>

*d定位

利用sizeof计算待分配空间大小

A: 0x00351728
Var_T: double
N: N/A
S: 8
V: ?
V_T: double

*d的内存六元组

malloc在工程中最常见的形式

给定一个int a[10]; a对应内存的表示值类型是int*
换句话说，10个int组成的内存表示值类型为int*，即

```
int* p = a;
```

假设我们用malloc申请10个连续int空间，即malloc(sizeof(int)*10)
其语义可视为申请一个int[10]空间，则该空间表示值类型应该为int*

```
int* q = (int *)malloc(sizeof(int)*10);
```

malloc在工程中最常见的形式

假设一个变量类型Var_T，申请N个Var_T大小的内存语法为

Var_T* p = (Var_T*)malloc(sizeof(Var_T)*N)

int* p = (int*)malloc(sizeof(int)*1)
int* p = (int*)malloc(sizeof(int)*10)

这是最常见的分配一维数组的方法

可视为分配了一个Var_T[N]变量类型空间

用malloc分配内存工程中最常见的形式

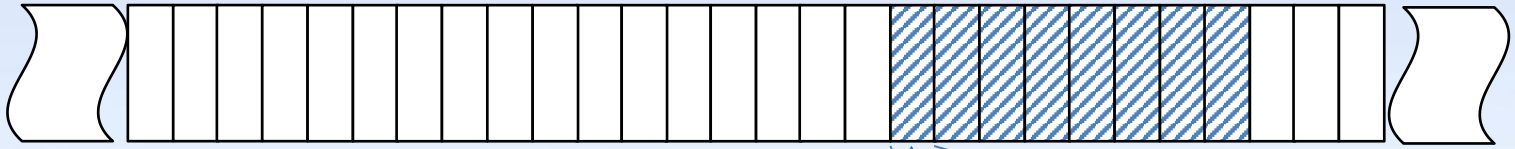
思考：

int* p = (int*)malloc(sizeof(int))

int* p = (int*)malloc(sizeof(int)*1)

malloc的返回值是一个指针类型，指针类型隐含对数组的访问

访问malloc分配的内存（续）



0x00392B11

char* c = (char*)malloc(sizeof(char)*8);

c: < 0x00392B11, char*>

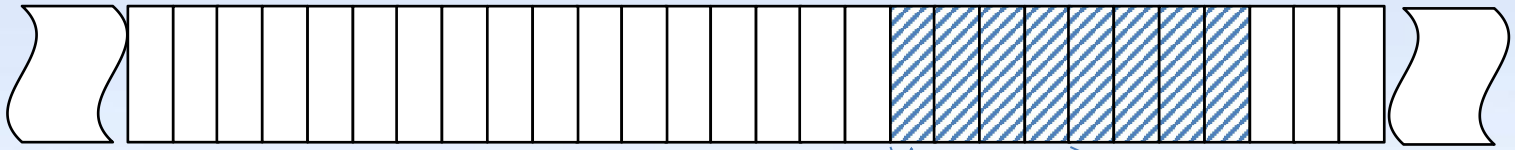
*定位

A: 0x00392B11
Var_T: char
N: N/A
S: 1
V: ?
V_T: char

*c的内存六元组

可视为分配一个char[8]类型变量的内存

访问malloc分配的内存（续）



0x00392B11

char (*r)[4] = (char(*)[4])malloc(sizeof(char[4])*2);

r: < 0x00392B11, char(*)[4]>

*定位

A: 0x00392B11
Var_T: char[4]
N: N/A
S: 4
V: 0x00392B11
V_T: char*

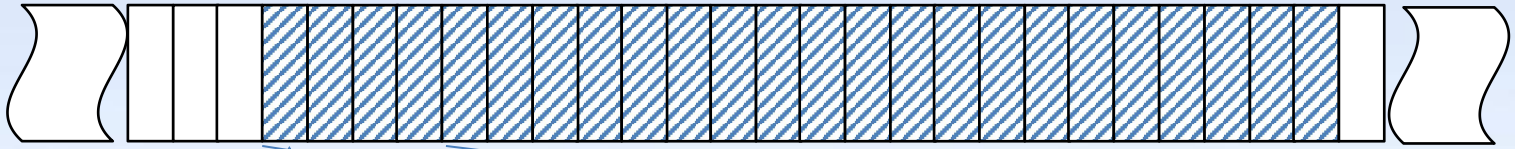
*r的内存六元组

可视为分配一个char[2][4]类型的空间

同样申请8个字节，区分指针类型的差异

char* vs. char(*)[4]

利用malloc分配一维数组



0x0029DF55

```
int* p= (int*)malloc(24);
或
int* p =(int*)malloc(sieof(int)*6);
```

p: <0x0029DF55, int*>

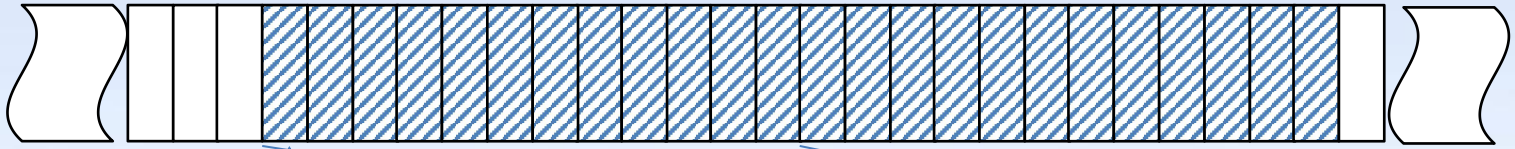
*定位

A: 0x0029DF55
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

*p的内存六元组

用sizeof(int)*6而不是24，语义更明确
这块内存的类型可视为int[6]

利用malloc分配二维数组



0x0029DF55

int (*q)[3] =(int(*)[3]) malloc(24);
或
int (*q)[3] =(int(*)[3])malloc(sieof(int[3])*2);

q: <0x0029DF55, int(*)[3]>

*定位

A: 0x0029DF55
Var_T: int[3]
N: N/A
S: 12
V: 0x0029DF55
V_T: int*

*q的内存六元组

用sizeof(int[3])*2而不是24，语义更明确
这块内存的类型可视为int[2][3]

利用malloc直接分配二维数组的缺点

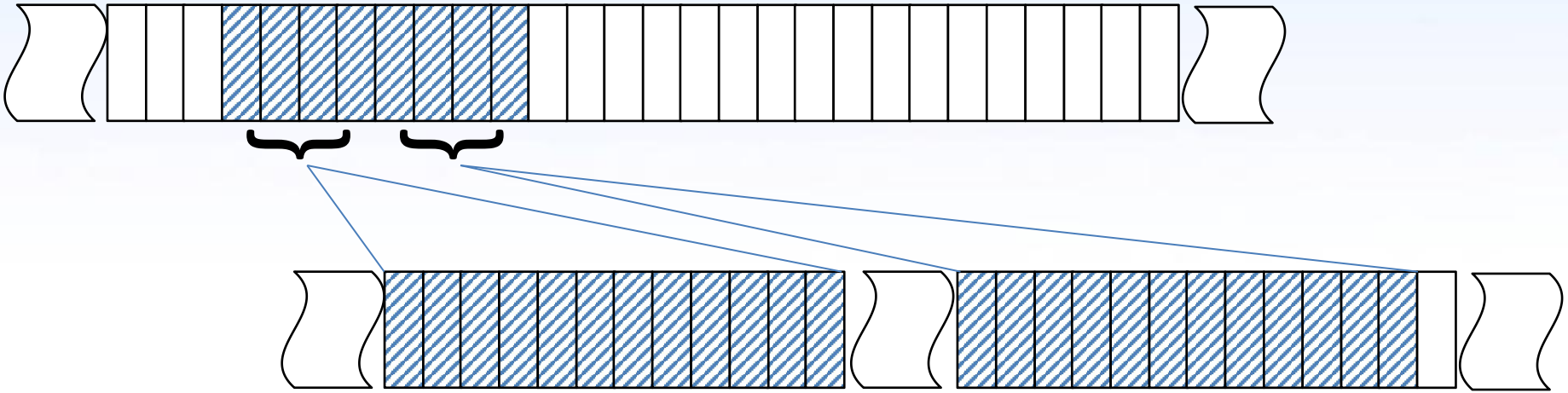
int (*q)[3] =(int(*)[3])malloc(sieof(int[3])*2);

指针类型为int(*)[3]，数字3需要写死在程序中，工程扩展性较差

malloc更常用于分配一维数组

更常见malloc分配二维数组

```
int** pp = (int**)malloc(sizeof(int*)*2);

for(int i=0; i<2; i++) {
    pp[i] = (int*)malloc(sizeof(int)*3);
}
```



思考题

利用malloc分配240个字节，如何定义指针变量p，让该240字节类型为

1、char[240]

2、int[6][10]

3、int[3][4][5]

思考题

利用malloc分配240个字节，如何定义指针变量p，让该240字节类型视为

1、char[240]

2、int[6][10]

3、int[3][4][5]

答案

1、char* p = (char*)malloc(sizeof(char)*240);

char (*p)[240] = (char(*)[240])malloc(sizeof(char[240])); → char[1][240]

2、int (*p)[10] = (int(*)[10])malloc(sizeof(int[10])*6);

int (*p)[6][10] = (int(*)[6][10])malloc(sizeof(int[6][10])); → int[1][6][10]

3、int (*p)[4][5] = (int(*)[4][5])malloc(sizeof(int[4][5])*3);

int (*p)[3][4][5] = (int(*)[3][4][5])malloc(sizeof(int[3][4][5])); → int[1][3][4][5]

思考题

void* p = malloc(32);

1、int* q = (int*)p;
2、char* r = (char*)p;
3、int (*s)[4] = (int(*)[4])p;
4、char (*t)[2][4] = (char(*)[2][4])p;

请给出q+1, r+1, s+1, t+1的值，假设P的值是0x006E1410

思考题

void* p = malloc(32);

1、int* q = (int*)p;
2、char* r = (char*)p;
3、int (*s)[4] = (int(*)[4])p;
4、char (*t)[2][4] = (char(*)[2][4])p;

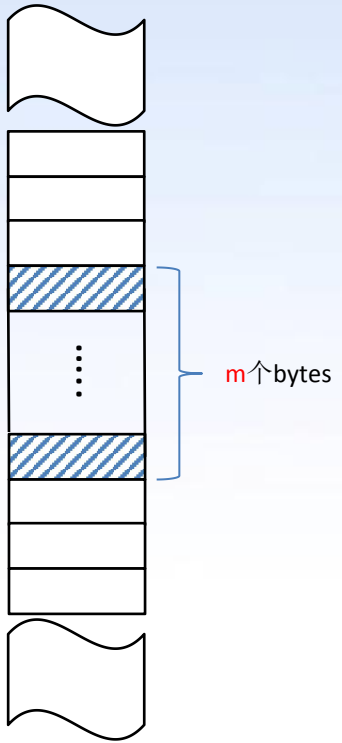
请给出q+1, r+1, s+1, t+1的值，假设P的值是0x006E1410

答案
1、0x006E1414；2、0x006E1411；3、0x006E1420；4、0x006E1418

内存的相关操作

分配内存

内存赋值



释放内存

读取内存相关信息

分配的内存需要释放

1、通过变量声明分配出来的内存不需要释放

2、通过malloc分配出来的内存需要用free进行释放

```
int* p = (int*)malloc(sizeof(int)*4);
free(p);
```

执行了多少次malloc，就需要执行多少次free

Memory Leak是一个恒久的挑战！！！

了解一下typedef

C语言提供了一个typedef的方法来为数据类型提供别名

Original Type	Target Type	Typedef Declaration
char	INT1	typedef char INT1;
short	INT2	typedef short INT2;
int	INT4	typedef int INT4;
int*	PINT	typedef int* PINT;
char*	PCHAR	typedef char* PCHAR;
int(*)[2]	PAINT	typedef int (*PAINT)[2];
int**	PPINT	typedef int** PPINT;
int[2]	AINT	typedef int AINT[2];
int[2][3]	AAINT	typedef int AAINT[2][3];

1、typedef char INT1;
将char定义一个别名INT1
char c; vs. INT1 c;

2、typedef int AINT[2];
注意定义数组别名的语法
int a[2]; vs. AINT a;

int[3]真的是变量类型吗？

Original: int[3], Target: VINT

typedef int VINT[3];

通过typedef新定义出来的VINT是不是变量类型？

```
VINT e;
VINT* p = &e;
int (*q)[3] = &e;
```

思考：VINT是不是一种变量类型?

如何将int[2][3]定义为VVINT类型？

方法1

Original : int[2][3], Target: VVINT

typedef int VVINT[2][3];

```
VVINT g;
VVINT* p = &g;
int (*q)[2][3] = &g;
```

方法2

利用刚才定义的VINT的方法

1、typedef int VINT[3];

2、typedef VINT VVINT[2];

```
VVINT g;
VVINT* p = &g;
int (*q)[2][3] = &g;
```

思考题

1、int* p, q;
p和q分别是什么变量类型？

2、typedef int* PINT;
PINT p, q;
p和q分别是什么变量类型？

思考题

1、int* p, q;
p和q分别是什么变量类型？

2、typedef int* PINT;
PINT p, q;
p和q分别是什么变量类型？

答案
1、p是int*类型，q是int类型

2、p和q都是int*类型

关于const

假设给定一个数据类型Var_T和变量名称N，定义常量变量的语法为

Var_T const N

意味着变量N对应的内存的表示值只能通过初始化赋值，后续不可以修改

例：

```
int const a = 10;

a = 20;
```

关于const

例：
int* const p = &a;

*p = 20; ✓

p = NULL; ✗

提示：int*是一个变量类型

typedef int* PINT;
PINT const p = &a;

关于const

例：

int const* p = &a;

p = NULL; ✓

*p = 20; ✗

提示：int const*是int const的指针类型

typedef int const CINT;

CINT* p = &a;

关于const

例：

int const* const p = &a;

p = NULL; ✗

*p = 20; ✗

int const*是int const的指针类型，将int const*看做一个数据类型

参考typedef int const* PCINT; 或 typedef int const CINT;
typedef CINT* PCINT;

PCINT const p = &a;

Var_T const vs. const Var_T

形式化定义上，Var_T const N = const Var_T N

示例：Var_T为int

int const a = 10;

无歧义

typedef int const CINT;

CINT a;

const int a = 10;

无歧义

typedef const int CINT;

CINT a;

a = 20; ✗

Var_T const vs. const Var_T

Var_T const也是一种数据类型，指向该数据类型的指针为Var_T const*

const Var_T也是一种数据类型，指向该数据类型的指针为const Var_T*

int const* p = &a;

无歧义

typedef int const CINT;

CINT* p = &a;

const int* p = &a;

无歧义

typedef const int CINT;

CINT* p = &a;

p = NULL; ✓

*p = 20; ✗

Var_T const vs. const Var_T

示例：Var_T为int*

int* const p = &a;

无歧义

typedef int* PINT;

PINT const a;

p = NULL; ✗

*p = 20; ✓

const int* p = &a;

如何理解？

把const int看作一个整体？
还是把int*看作一个整体？

Var_T const vs. const Var_T

int* const p = &a;

P = NULL; ✗

*p = 20; ✓

const int* p = &a;

typedef const int CINT;
CINT* p = &a;

P = NULL; ✓

*p = 20; ✗

typedef int* PINT;
const PINT p = &a;

P = NULL; ✗

*p = 20; ✓

const int* p = &a;

P = NULL; ✓

*p = 20; ✗

Var_T const vs. const Var_T

int const* const p = &a;

p = NULL; ✗

*p = 20; ✗

const const int* p = &a; ?

p = NULL; ✓

const int const* p = &a; ?

p = NULL; ✓

const int* const p = &a; ?

p = NULL; ✗

*p = 20; ✗

Var_T const vs. const Var_T

int const* const p = &a;

p = NULL; ✗

*p = 20; ✗

typedef const int CINT;

const CINT* p = &a;

p = NULL; ✓

*p = 20; ✗

typedef const int CINT;
typedef CINT* PCINT;

const PCINT p = &a;

p = NULL; ✗

*p = 20; ✗

C语言标准中对const位置的摆放没有明确语义规范

推荐使用Var_T const N的形式，也更有利于理解Var_T是一种数据类型

测试一下效果

int n[2][3][4][5]={0}，以下表达式返回值是什么？

&n, &n+1
n, n+1
n[0], n[0]+1
n[0][0], n[0][0]+1
n[0][0][0], n[0][0][0]+1
n[0][0][0][0], n[0][0][0][0]+1

假设变量**n**对应的内存首地址为0x00FF3811
返回值表示为**<Value, Value_Type>**形式

测试一下效果

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
printf("%d\n", *(((int*)(&a+1))-1));
```

打印的结果是多少？

提示

1、&a+1

2、(int*)(&a+1)

3、((int*)(&a+1))-1

4、*(((int*)(&a+1))-1)

知之、好之、乐之

谢谢！