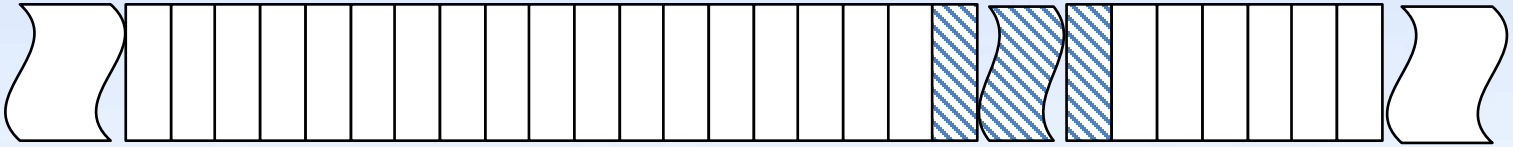
刻画对象的属性：对象名字



对象有名字吗？ 了解 lvalue的重要性

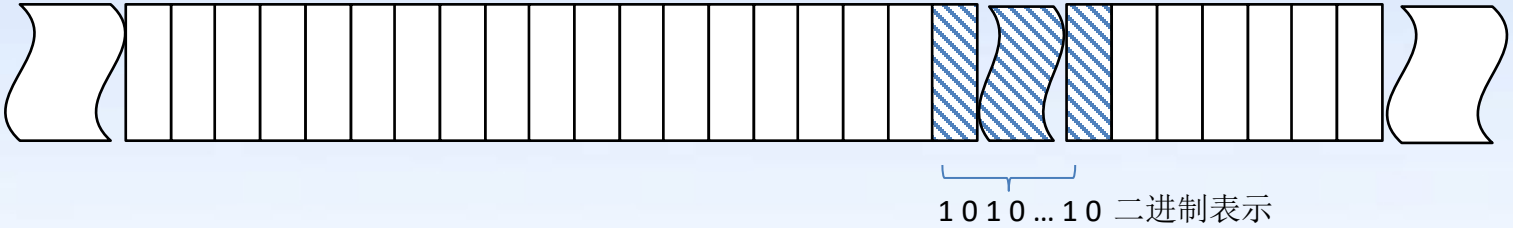
1、对象声明，例如*T O*;

这块对象的标识符就是*O*（我们将这个*O*认为是这块内存的名字）

2、动态分配，例如malloc(4);

这块对象没有标识符（没有名字）

刻画对象的属性：对象表示

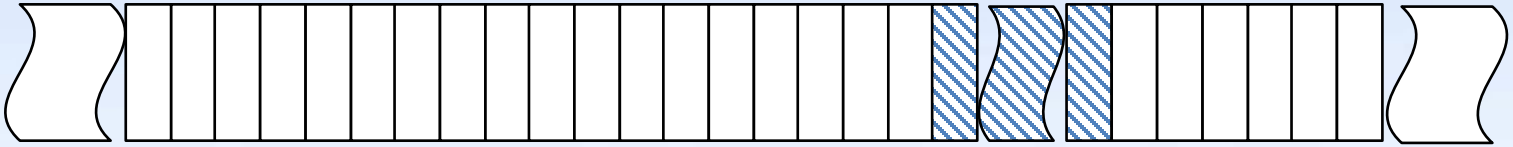


这个对象各个bit组成的二进制串就是这个对象的对象表示(Object Representation)

Object Type + Object Representation → Object Value

了解Trap Representation这个概念

刻画对象的属性：值



<Value, Value Type>

对象都有值（Value），且这个值和值的类型（Value Type）和对象类型有关

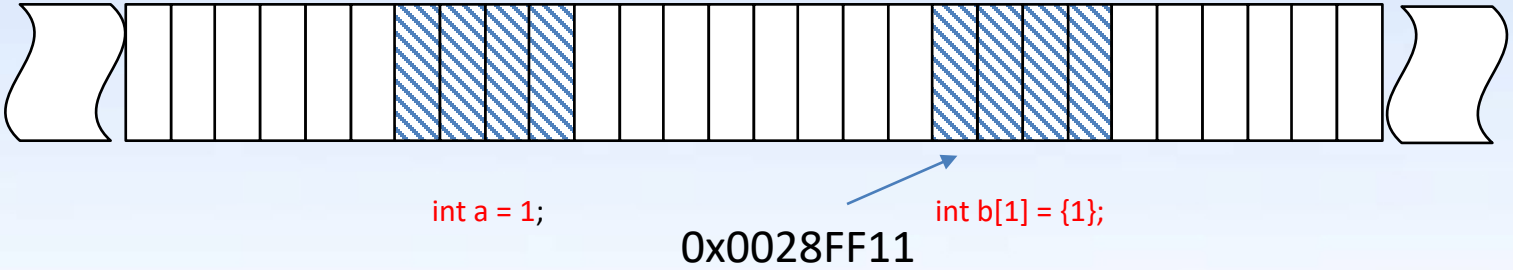
1、非数组对象类型，值类型=对象类型，对象表示按规则转换成Value lvalue conversion

2、数组对象类型，<第一个元素的首地址，元素类型对应的pointer type>

int a[2]，该对象类型是int[2]，所以这块内存的值就是<第一个元素的首地址，int*>

对象取值可以参考本系列视频的4和12

刻画对象的属性：值的示例



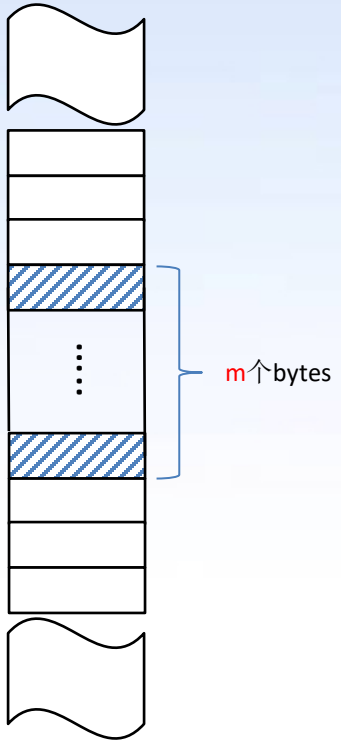
变量a对应的对象和变量b对应的对象的32为0/1串是一样的

变量a对应的对象的值: <1, int>

变量b对应的对象的值: <0x0028FF11, int*>

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;

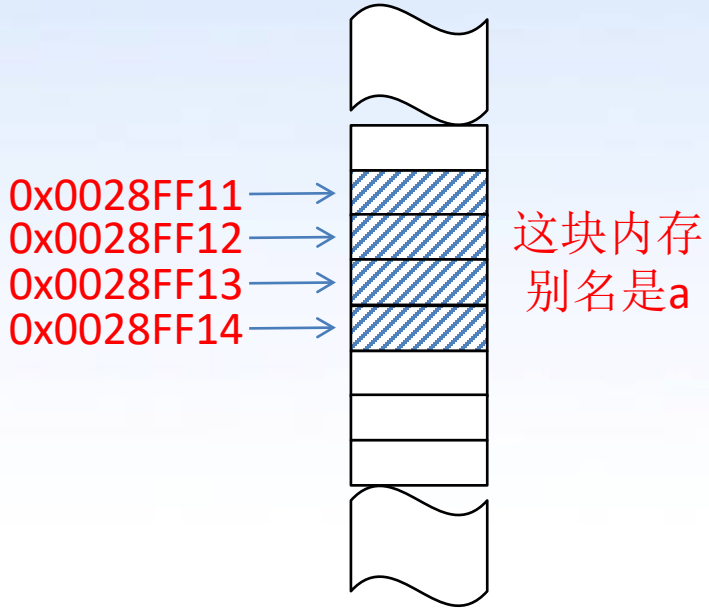
1、分配4个连续byte（为什么是4？）

2、这4个byte组成的块的别名为a

思考：0x0028FF11~0x0028FF14怎么来的？

0x0028FF11这个地址有什么问题呢？

对齐的知识点后面会说明



如何描述变量a对应的内存

int a；

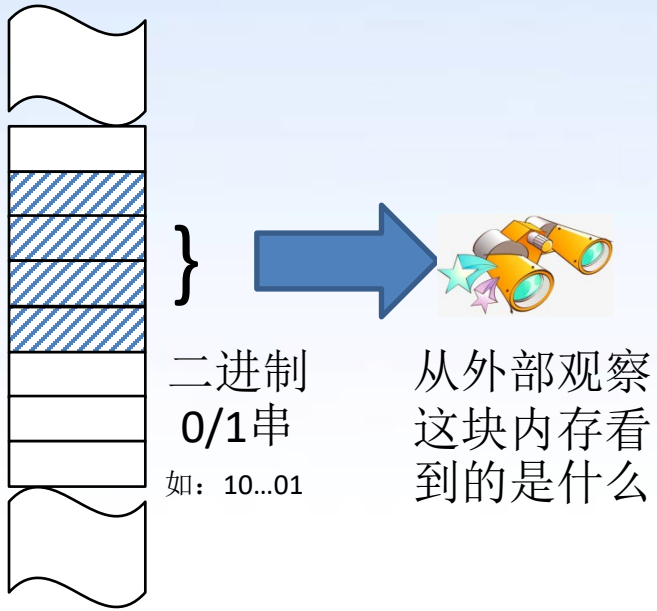
1、这块连续内存第一个byte编号是0x0028FF11

2、这块内存的对象类型是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？

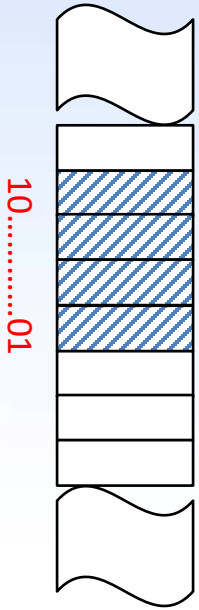
在C语言中，值包含了两个层面的语义：1）值；2）值的类型

Value (V); Value_Type (V_T)记为<V, V_T>

任何一个表达式(exp)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）

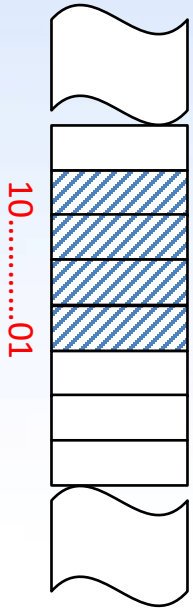
变量a所对应的那块内存也有表示值，也记为<Value, Value_Type>

进一步理解内存的表示值



<V, V_T>

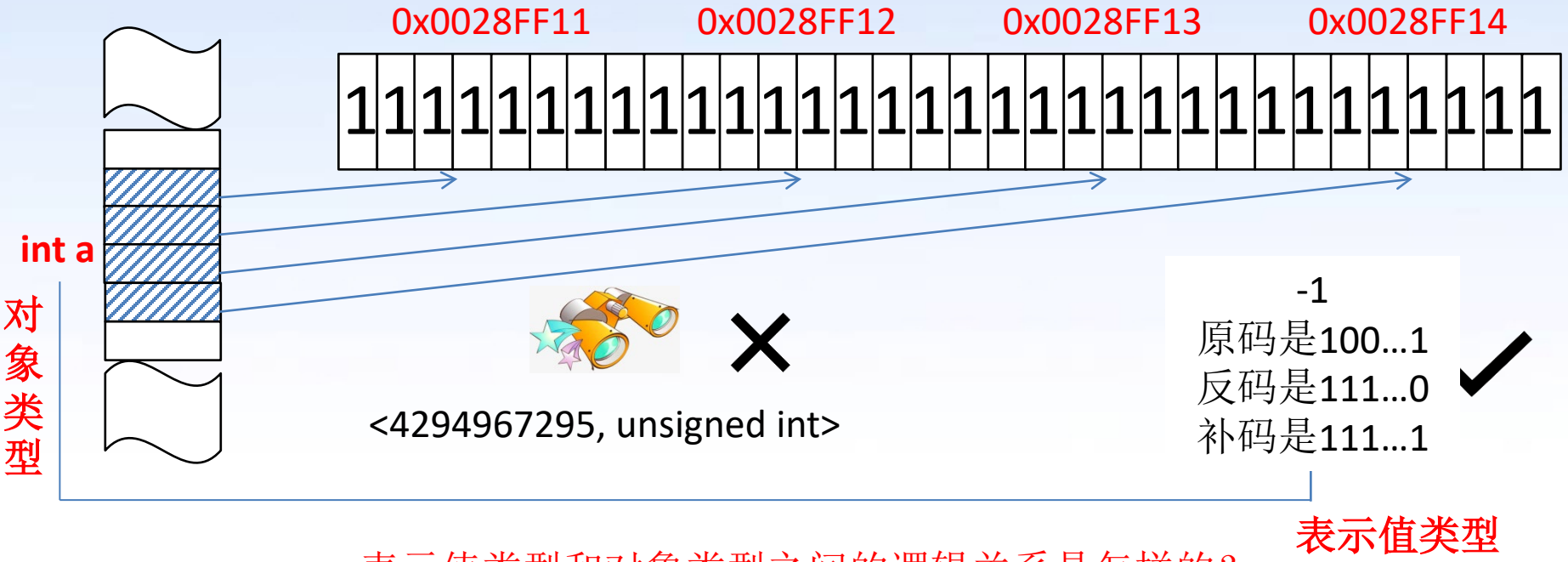
观察整块内存
得到的一个值



<V, V_T>

将一个值按照
写入整块内存

进一步理解内存的表示值

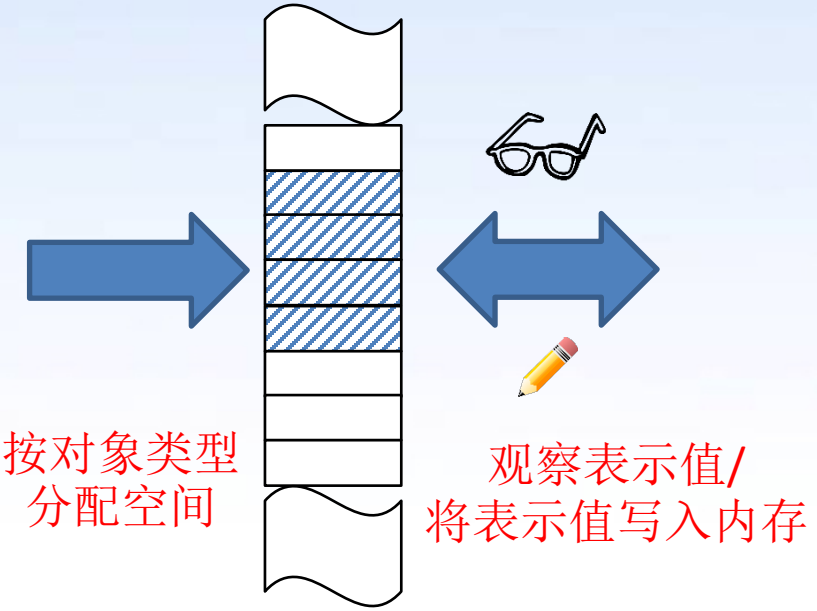


表示值类型和对象类型之间的逻辑关系是怎样的？

内存对应的对象类型 vs. 表示值类型

1、内存的对象类型是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. 对象类型

内存表示值与内存对应的对象类型关系

假设内存表示值记为<V, V_T>，内存对应的对象类型记为Obj_T

1、如果Obj_T是非数组类型

V_T= Obj_T，V则是通过Obj_T去观察这段内存获得的值

例如：int a， Obj_T是int， V_T也是int

2、如果Obj_T是数组类型

V_T是该数组类型中元素对象类型对应的指针类型，V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]， Obj_T是int[10]，元素类型是int， V_T是int*

数组对象取值

。。。an expression that has type “array of type” is converted to an expression with type “pointer to type” that points to the initial element of the array object and is not an lvalue.

先有大致印象，后续我们会专门介绍lvalue这个核心概念

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; 对象类型Object_Type为char，非数组类型，
Value_Type=char

2、double d; 对象类型Object_Type为double，非数组类型，
Value_Type=double

3、float f[3][4]; 对象类型Object_Type为float[3][4]，数组类型，元素类型为float[4]，
Value_Type=float(*)[4]

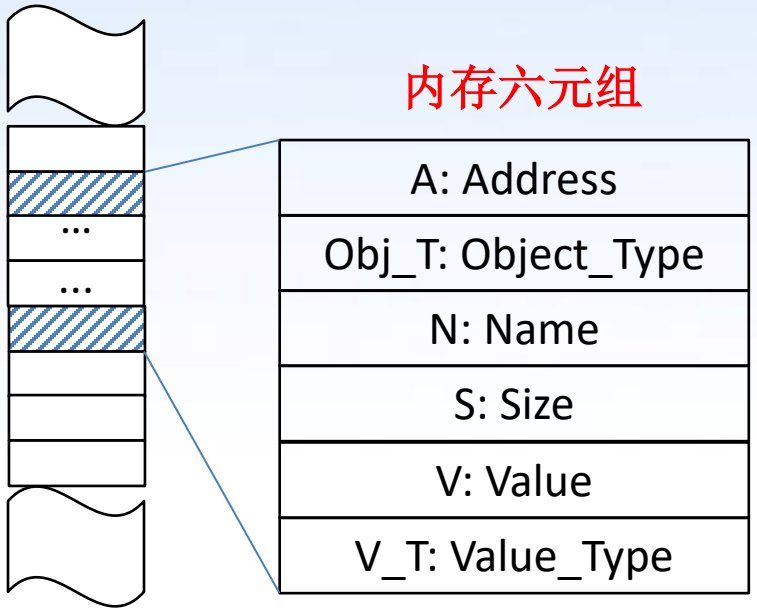
4、int* p[3]; 对象类型Object_Type为int*[3]，数组类型，元素类型为int*，
Value_Type=int**

内存六元组模型

Object = {Address, Object_Type, Name, Size, Value, Value_Type}

Object：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
- Object_Type: 对象类型
- 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, Object_Type, Name, Size, Value, Value_Type}

1、Address由系统分配，一旦确定无法修改

2、Object_Type和Name是变量声明对应的对象类型和变量名

3、Size是这块内存的大小（字节数）

4、Value和Value_Type的取值根据Object_Type来确定

Object_Type是非数组对象类型 vs. 数组对象类型

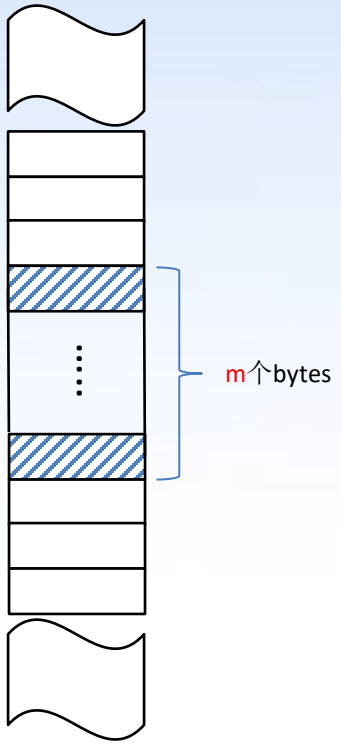
内存六元组是为了让大家更好的了解分配的内存而提出的一个逻辑模型

内存的相关操作

分配内存

1、通过变量声明

2、通过malloc

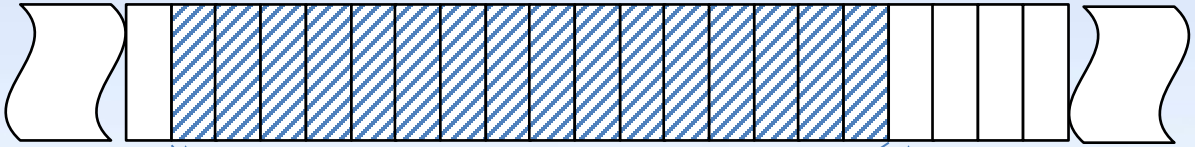


内存赋值

释放内存

读取内存相关信息

观察：malloc(16)



0x00351728　　0x00351738

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

```
malloc(16);
```

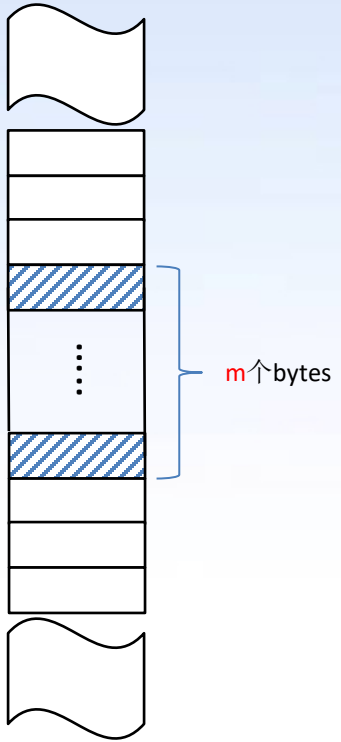
思考1： 为什么有Address和Size

思考2： 这块内存为什么没有Object_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

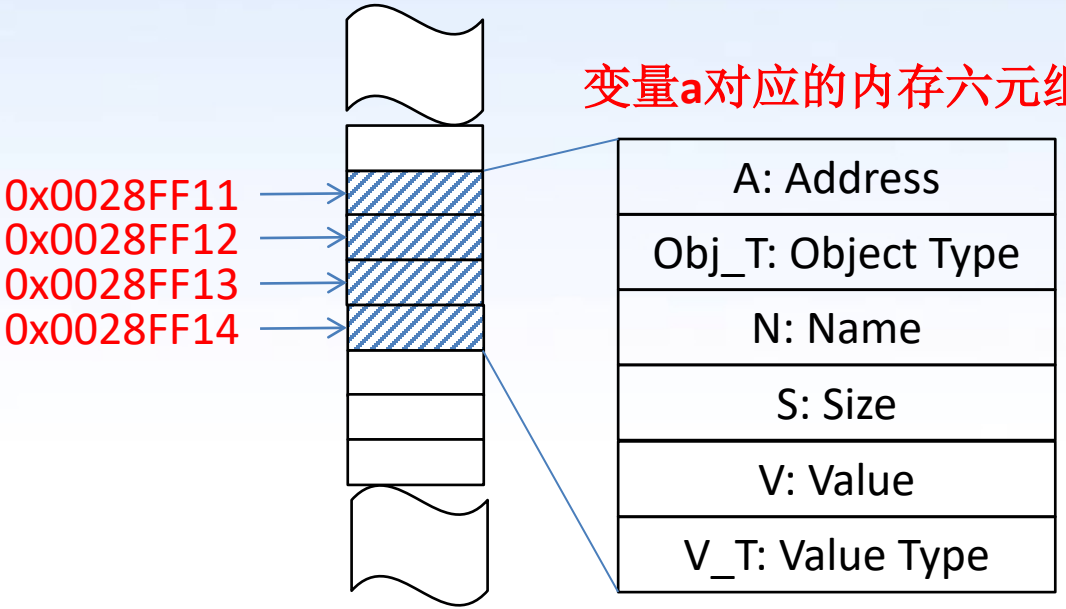
变量赋值示例：int a;a=10;

1) int a; 2) a = 10;

在内存中分配了4个字节
1、对象类型是int
2、变量名称是a

通过int a变量声明出来的内存如何描述

int a;



Address: 0x0028FF11
Object Type: int
Name: a
Size: 4
Value: ？ (Undefined)
Value Type: int

Object_Type是非数组对象类型
Value_Type = Object_Type

思考：这段内存有Value吗？

变量赋值示例：int a;a=10;

1) int a; 2) a = 10;

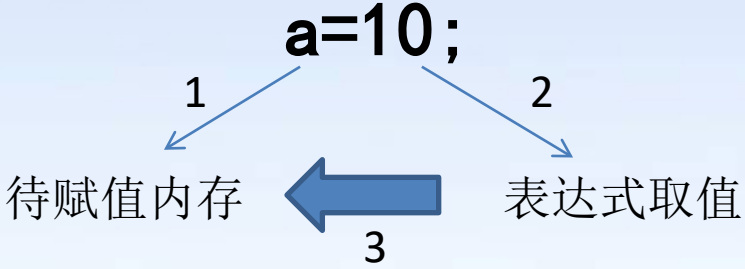
基础表达式　　基础表达式

在C语言中，只能对一块内存进行赋值，因此

等号左边表达式：要求必须能识别出一块有效内存

预告：a和10都是一种基础表达式，但a是lvalue，10不是lvalue很多地方翻译成左值，但l不是left，是locator

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？

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：

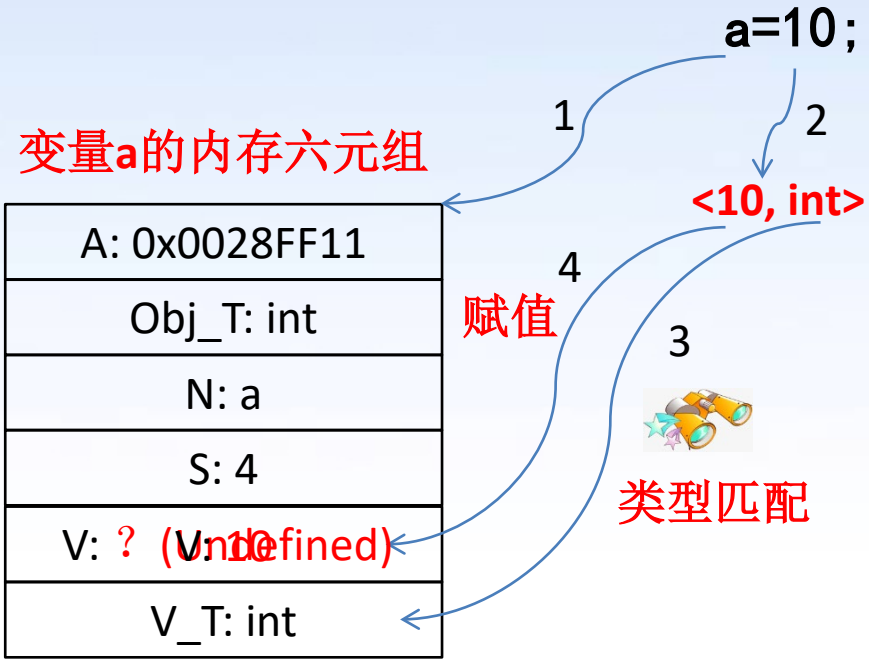
int a; float b; double c;

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



1、根据变量名a定位内存（用六元组描述）

2、获得等号右边表达式的值

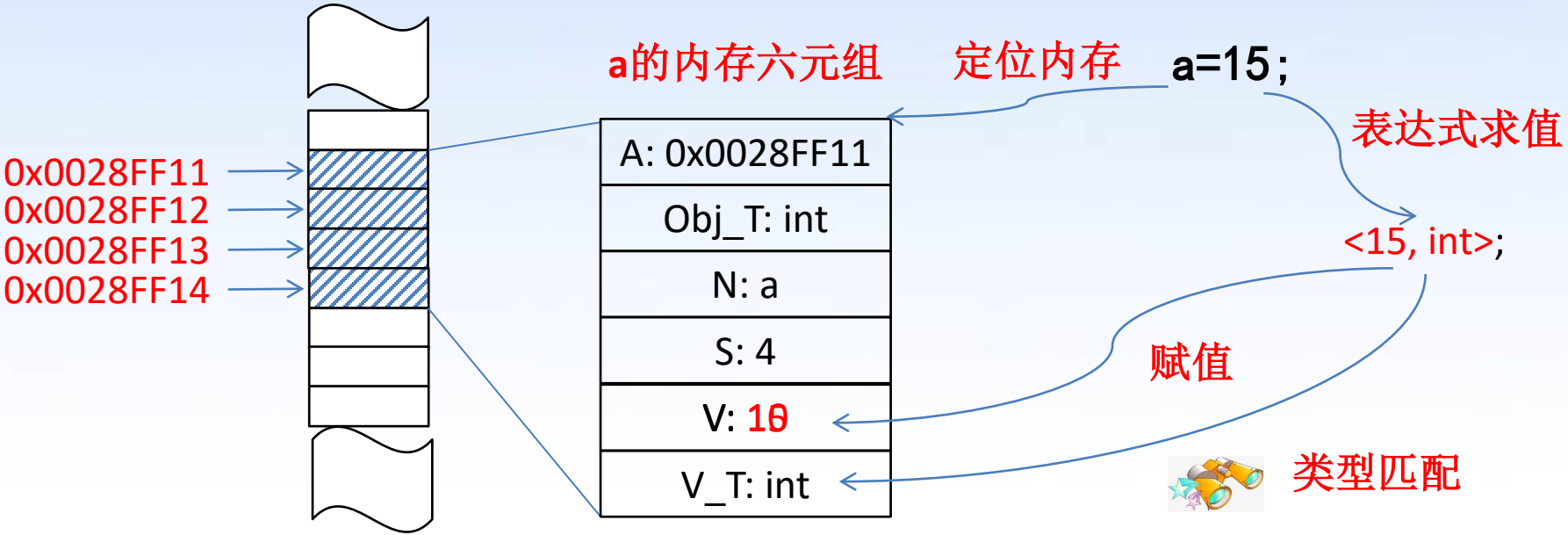
<Value, Value_Type>（<10, int>）

3、检查Value_Type是否与内存表示值 的Value_Type匹配 （潜在的Warning/Error）

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？



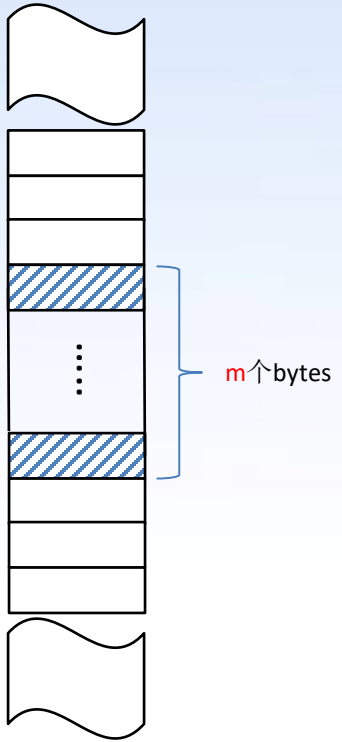
思考：现在Value为什么是10

其他对象类型声明和赋值示例

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

其他对象类型声明和赋值示例

```
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*

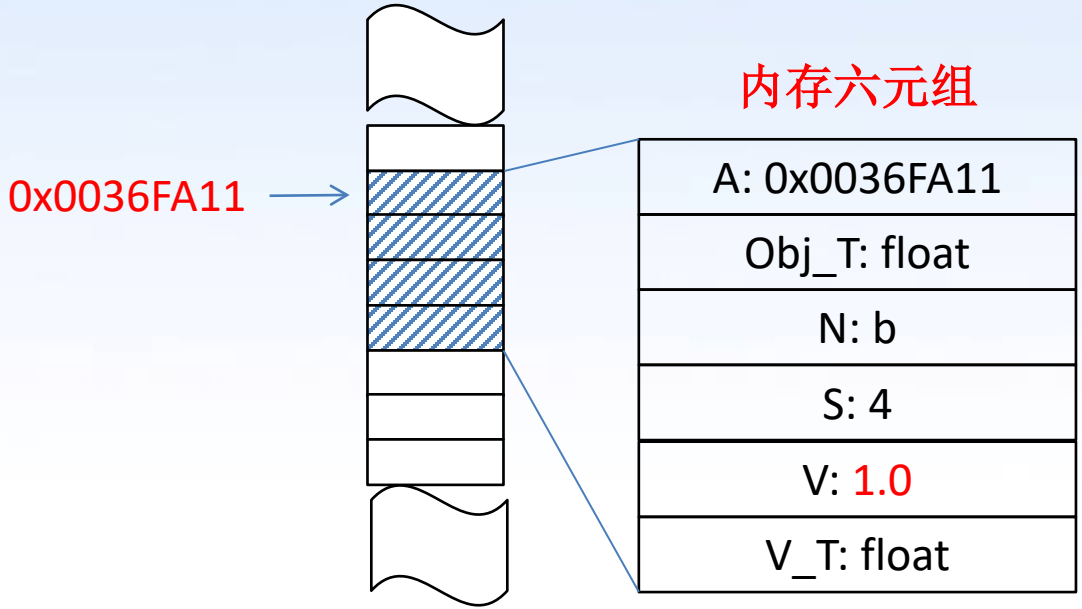
一维数组，例如：int[2], char[8]

“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：float b=1.0;

这是变量初始化，分配内存的同时对内存赋值

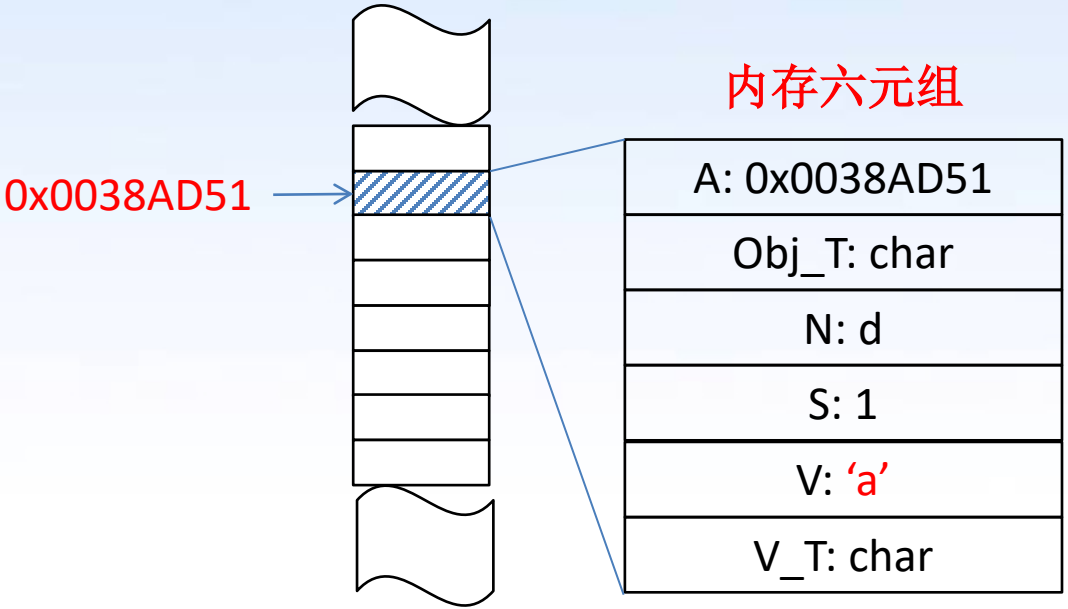


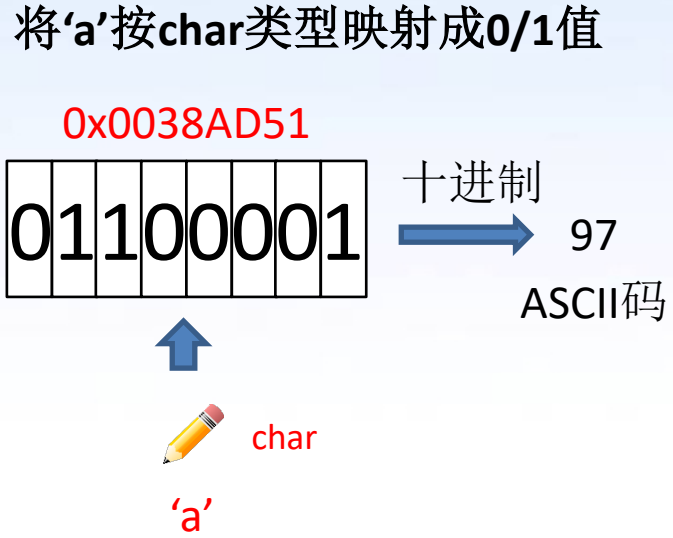
将‘1.0’按float类型映射成0/1值

32位0/1串是实际存储的值
1.0是用float观察出来的值

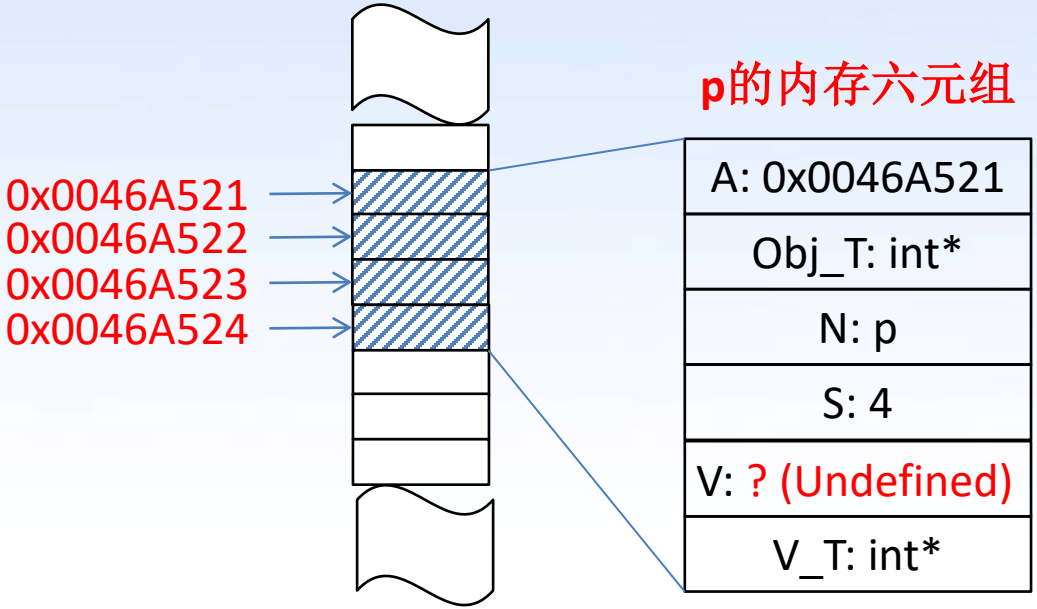
观察：char d='a';

这是变量初始化，分配内存的同时对内存赋值





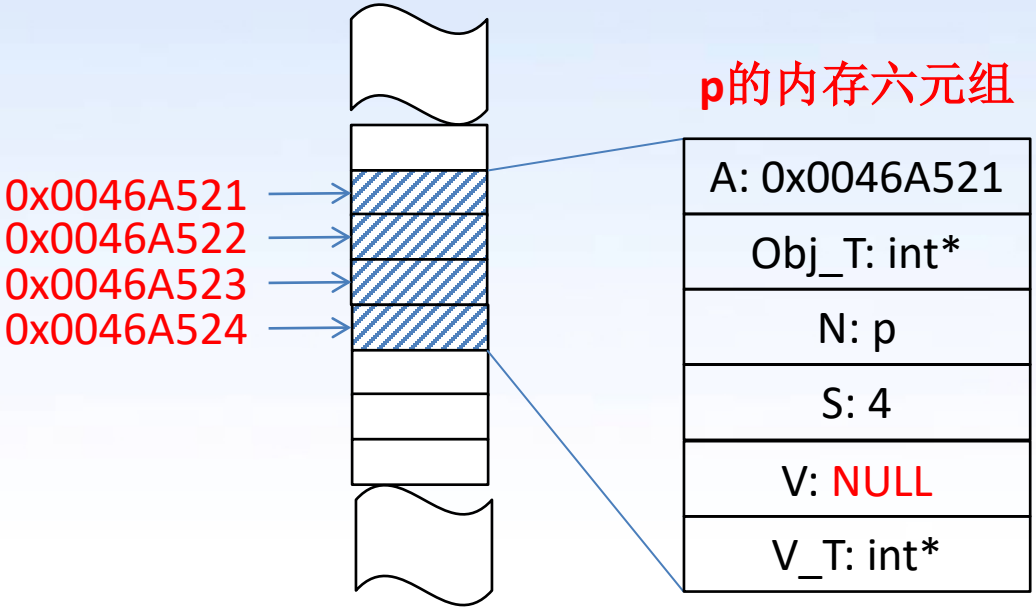
观察：int* p;



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;



变量合理的初始化是
良好的防错性编程习惯

复习一下：数组对象类型的表示值

假设内存表示值记为<V, V_T>，内存对应的对象类型记为Obj_T

1、如果Obj_Type是**非数组类型**

V_T=Obj_T，V则是通过Obj_T去观察这段内存获得的值

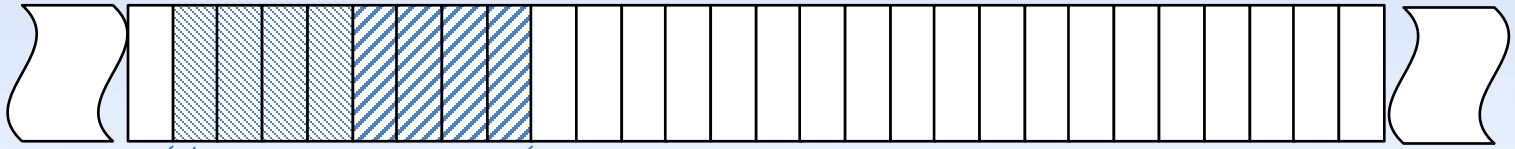
例如：int a，Obj_T是int，V_T也是int

2、如果Obj_T是**数组类型**

V_T是该数组类型中元素对象类型对应的指针类型，V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]，Obj_T是int[10]，元素类型是int，V_T是int*

观察：int e[2];



0x0076AB11

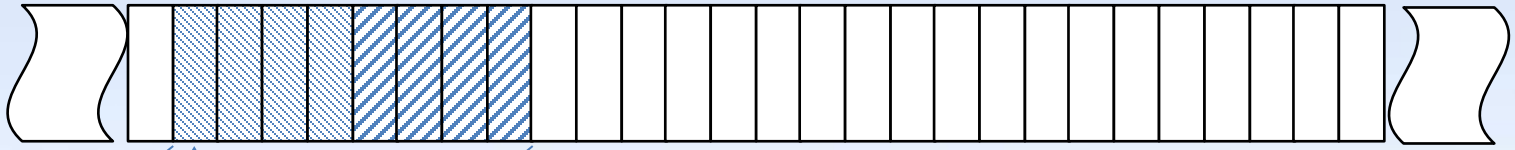
A: 0x0076AB11
Obj_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

1、数组类型：Object_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、第一个元素对象类型是int，所占4个字节编号
[0x0076AB11, 0x0076AB14]

观察：int e[2]={1,2};



0x0076AB11

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

<1, int> <2, int>

1、数组对应的内存整体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];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
Obj_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 ✓

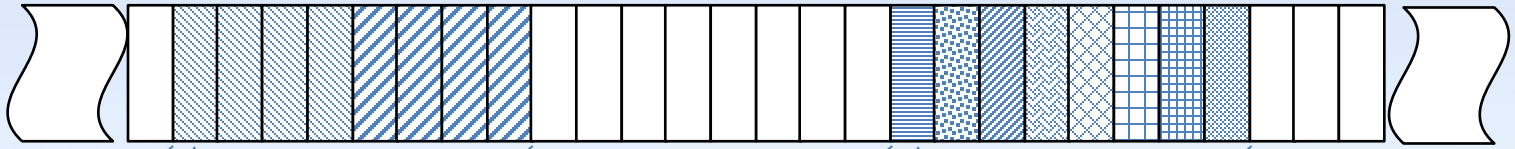
数组变量内存的Value一定是指向数组第一个元素的地址编号

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

真实原因：e不是一个modifiable lvalue

观察：int e[2];和char f[8];



0x0076AB11

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

0x0076AB21

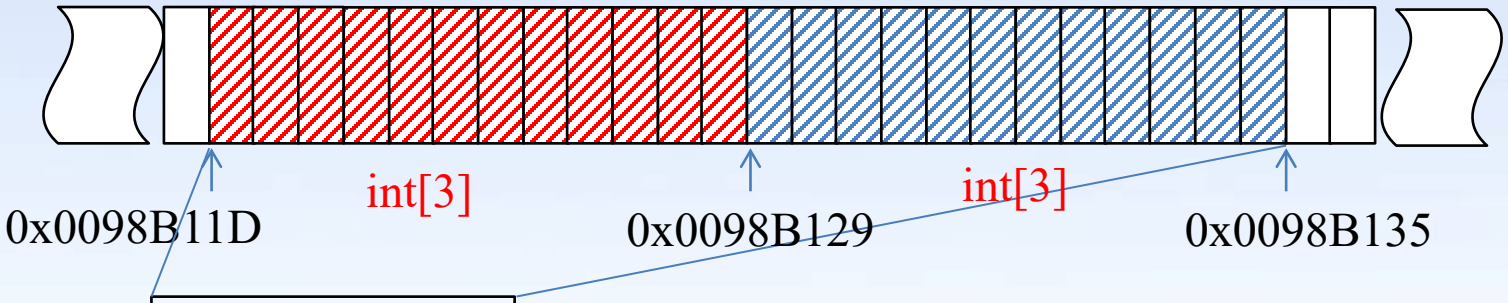
A: 0x0076AB21
Obj_T: char[8]
N: f
S: 8
V: 0x0076AB21
V_T: char*

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

1、数组类型：Object_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、第一个元素对象类型是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Object_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对象类型是什么？

3、变量m对应的内存的表示值类型Value_Type是什么？

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

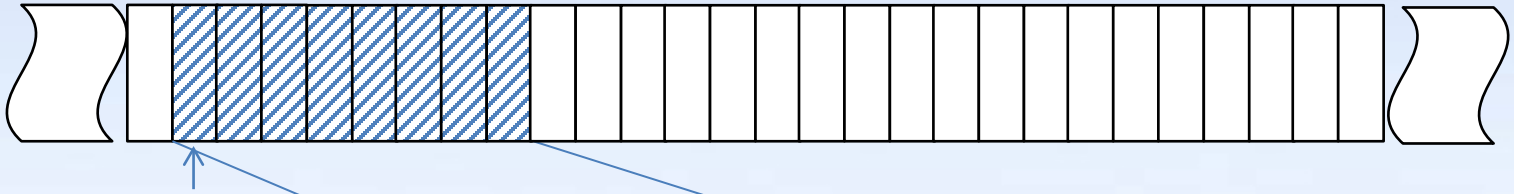
答案：

1、Object_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;
```

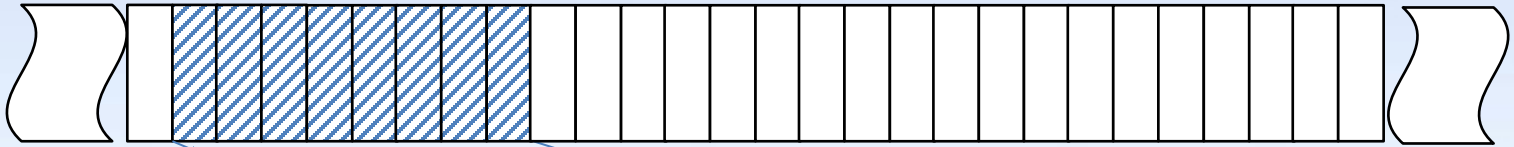
A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Undefined
V_T: struct m_struct

struct m_struct是非数组对象类型

使用sizeof获得结构体变量大小
sizeof(struct m_struct)

思考题：Value现在是什么？

观察：结构体变量h



0x0085D611

```
struct m_struct {
    int a;
    float b;
} h = {10, 1.0};
```

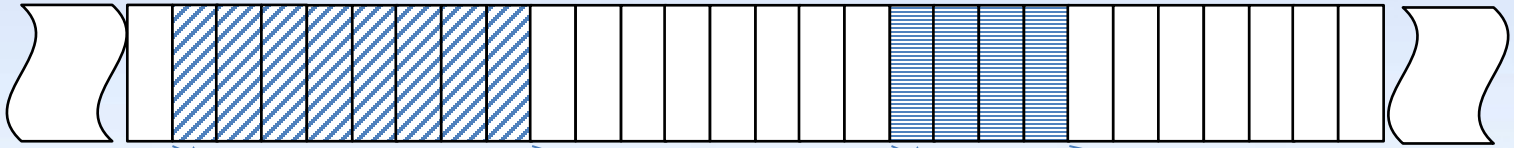
A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Invisible
V_T: struct m_struct

printf("h= ?\n", h);

内存的值观察的是整个内存

h对应的内存有值，我们只是看不懂

观察：对比结构体变量h和联合体变量i



0x0085D611

0x0085D621

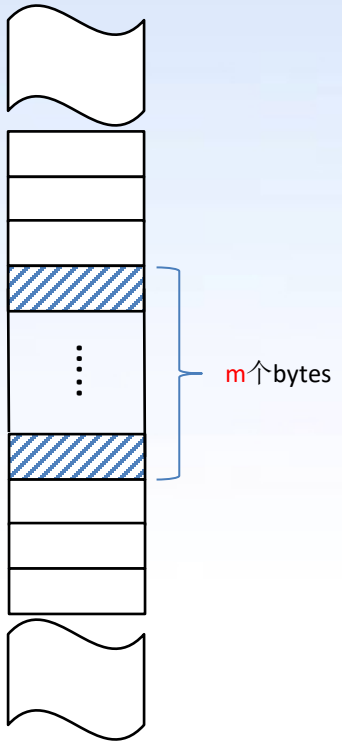
```
struct m_struct {
    int a;
    float b;
}h;

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

A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Undefined
V_T: struct m_struct

A: 0x0085D621
Obj_T: union m_union
N: i
S: 4
V: Undefined
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

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;

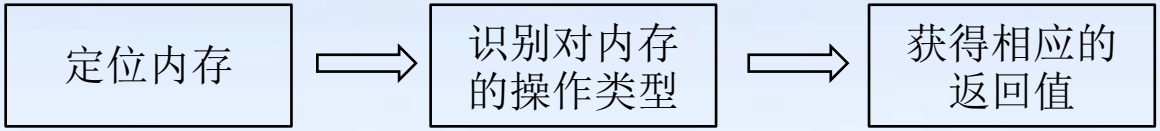
通过a，b，c就可以定位到对应的内存

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

int* p; int (*q)[10];

通过*p, *q来定位对应的内存

内存相关信息读取的三种操作



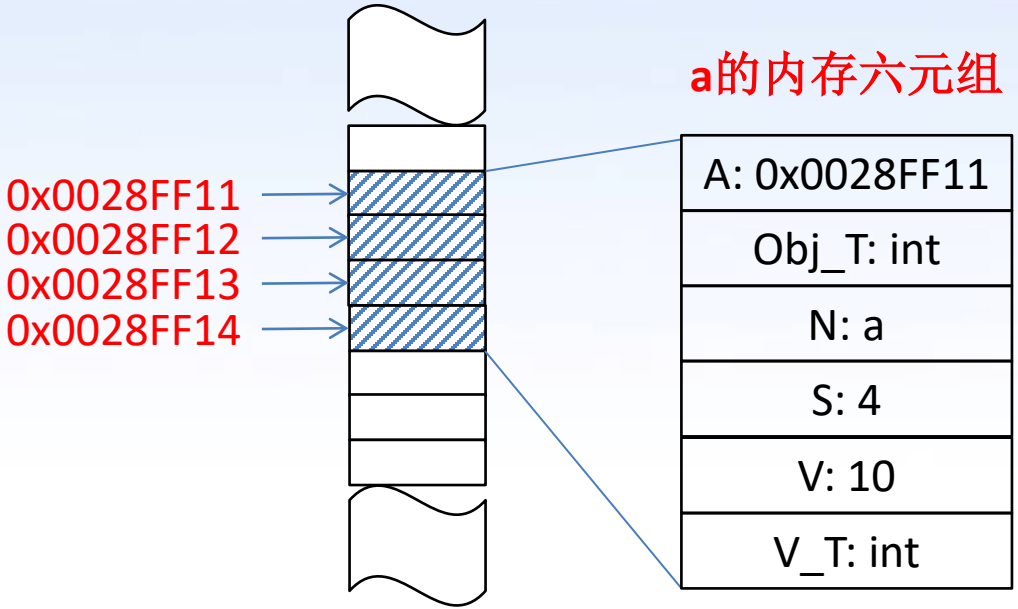
通过表达式定位内存：
1、变量名表达式
例如： a, b, c
2、*+指针类型表达式
例如: *p, *(p+1)

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>

对内存的三种取值操作

给定int a; a=10;



观察下面三个语句

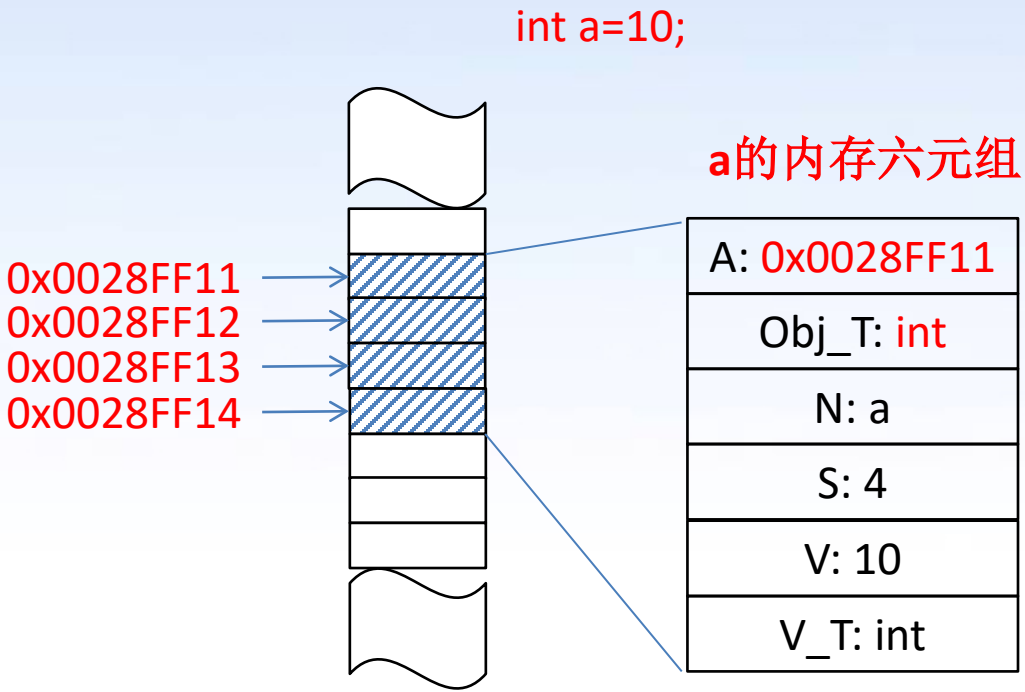
首先通过变量名定位a所在内存，然后三个内存相关信息获取操作按优先级：

1、&a

2、sizeof(a)

3、a

观察：int a=10；&a返回值是什么？



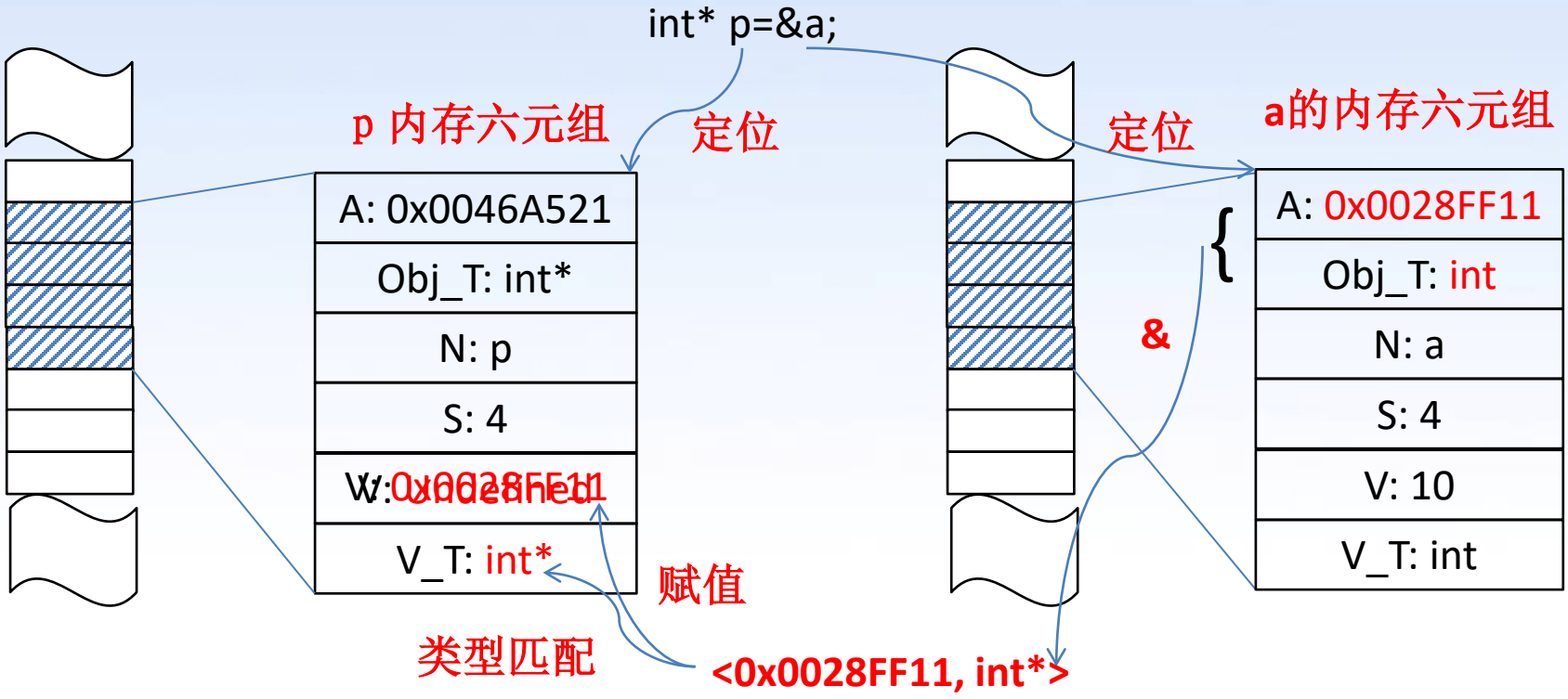
对于&a：
1、识别a所在内存
2、发现前面有&
3、返回值规则如下：

返回值：<Address, Obj_T*>

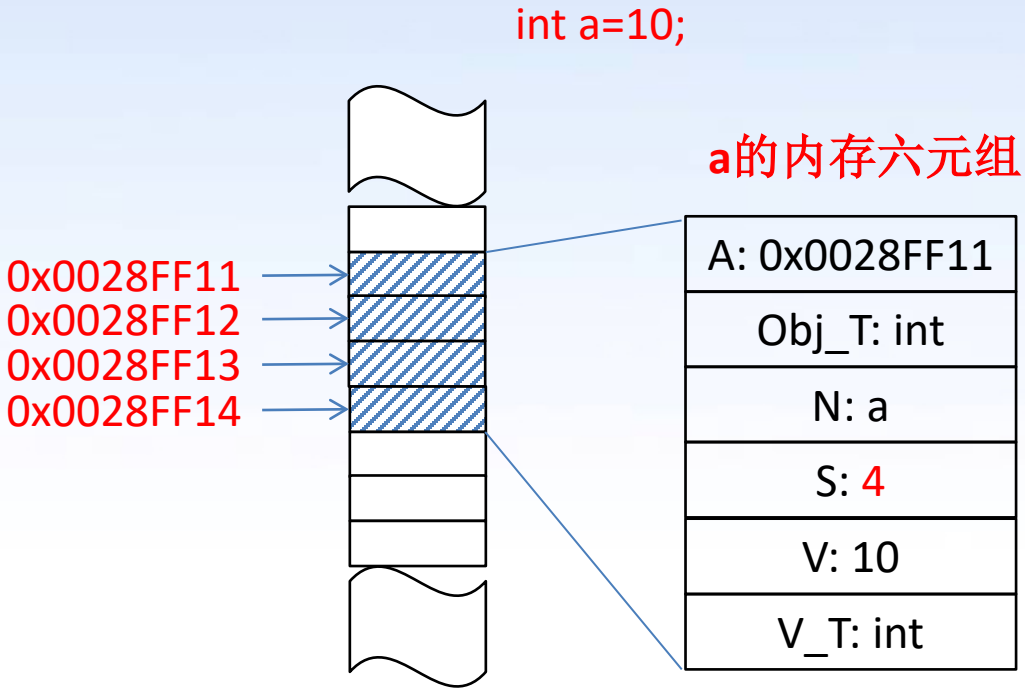
Obj_T*是指向Obj_T的指针类型
*的具体位置依语法而定

&a：<0x0028FF11, int*>

观察：int* p=&a发生了什么？



观察：int a=10; sizeof(a)返回值是什么？

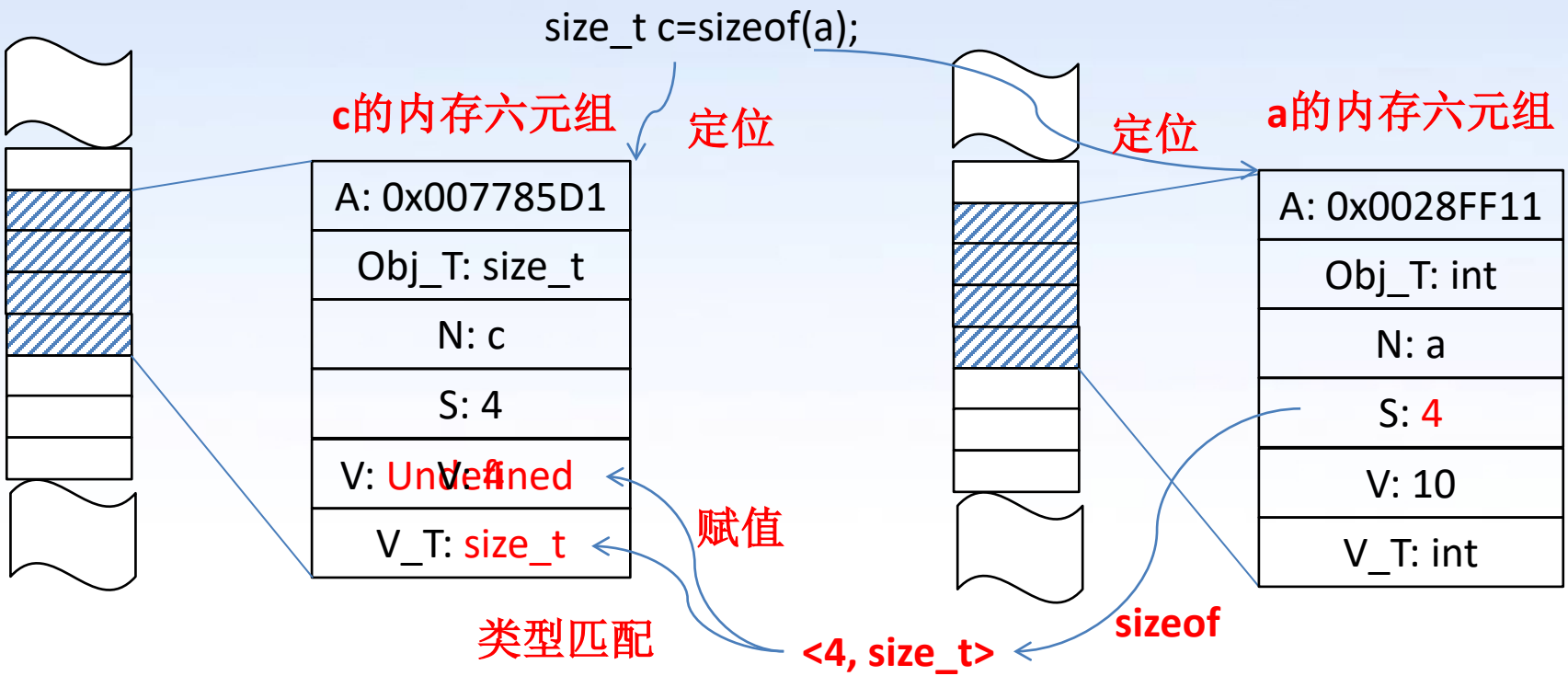


对于sizeof(a)：
1、识别a所在内存
2、发现前面没有&
3、发现和sizeof结合一起
3、返回值规则如下：

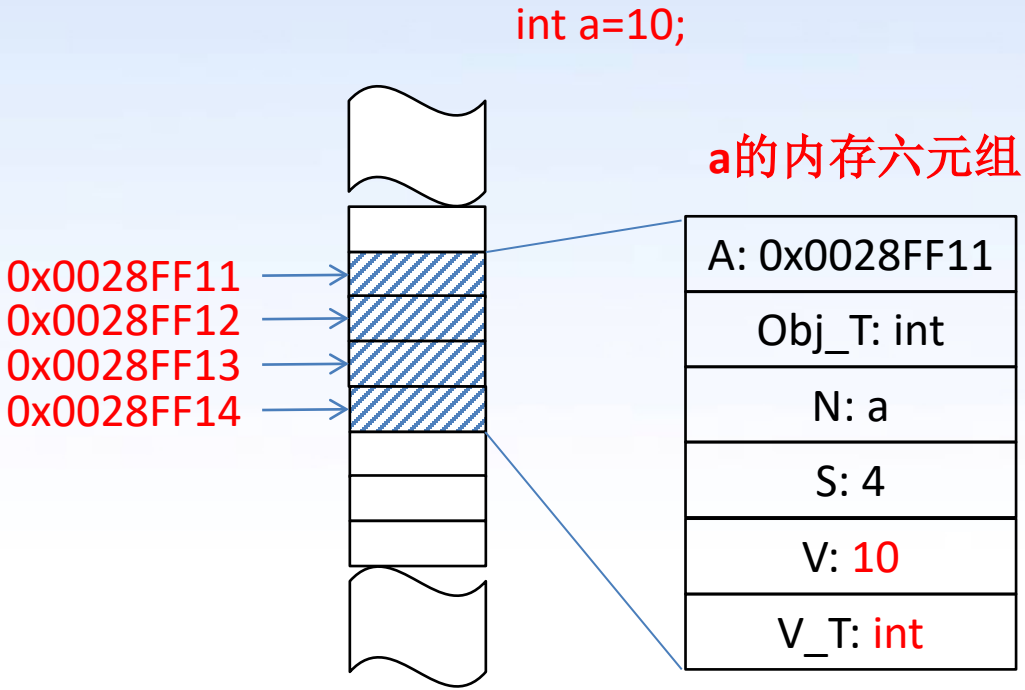
返回值：<Size, size_t>
size_t是内存大小的数据类型

sizeof(a)：<4, size_t>

size_t c=sizeof(a)发生了什么？



观察：int a=10；a返回值是什么？



对于a（例如a++）：

1、识别a所在内存

2、发现前面没有&

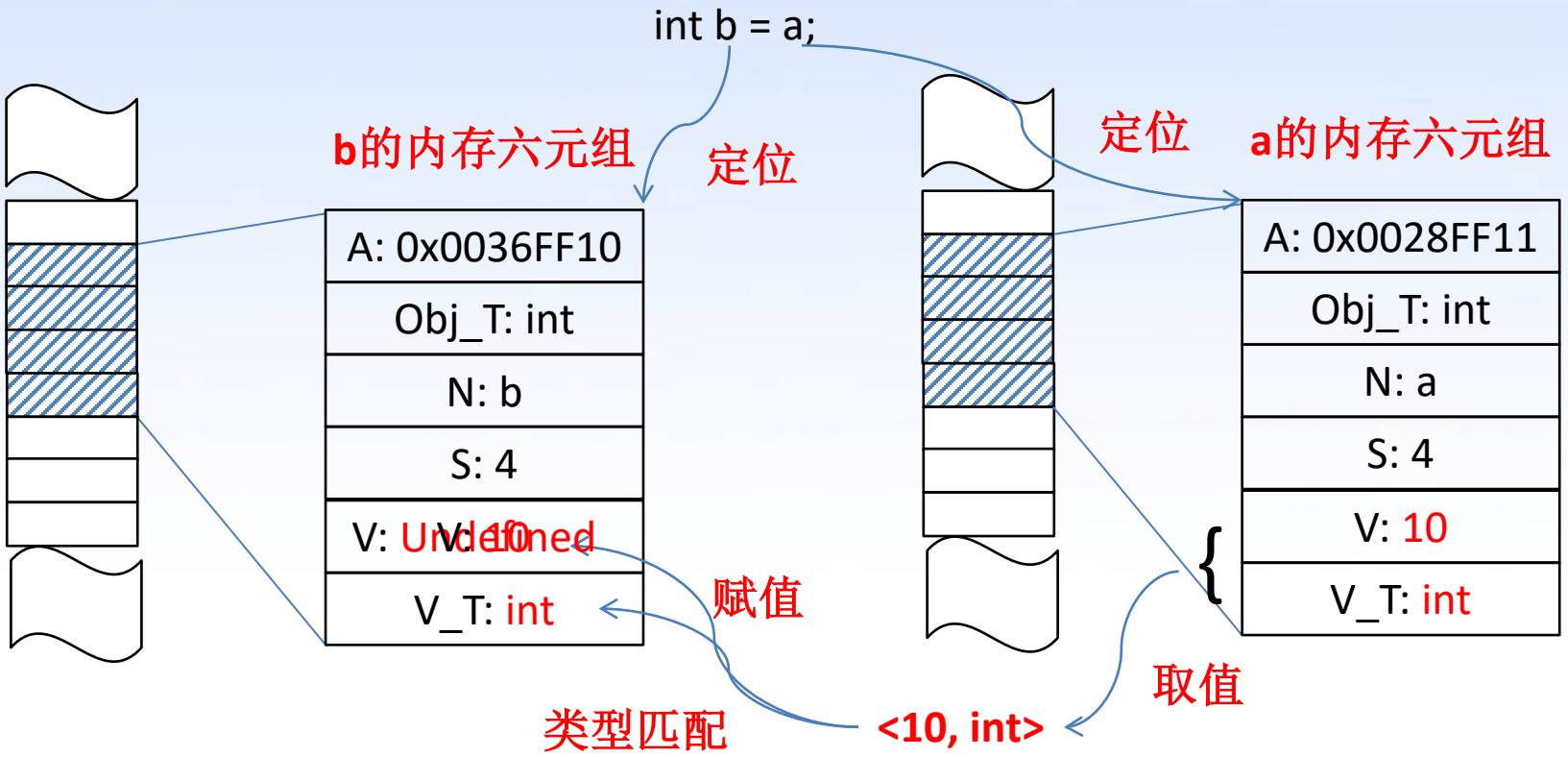
3、发现没有和sizeof结合一起

3、返回值规则如下：

返回值：<Value, Value_Type>

a: <10, int>

int b=a时发生了什么？



变量 vs. 变量对应的内存

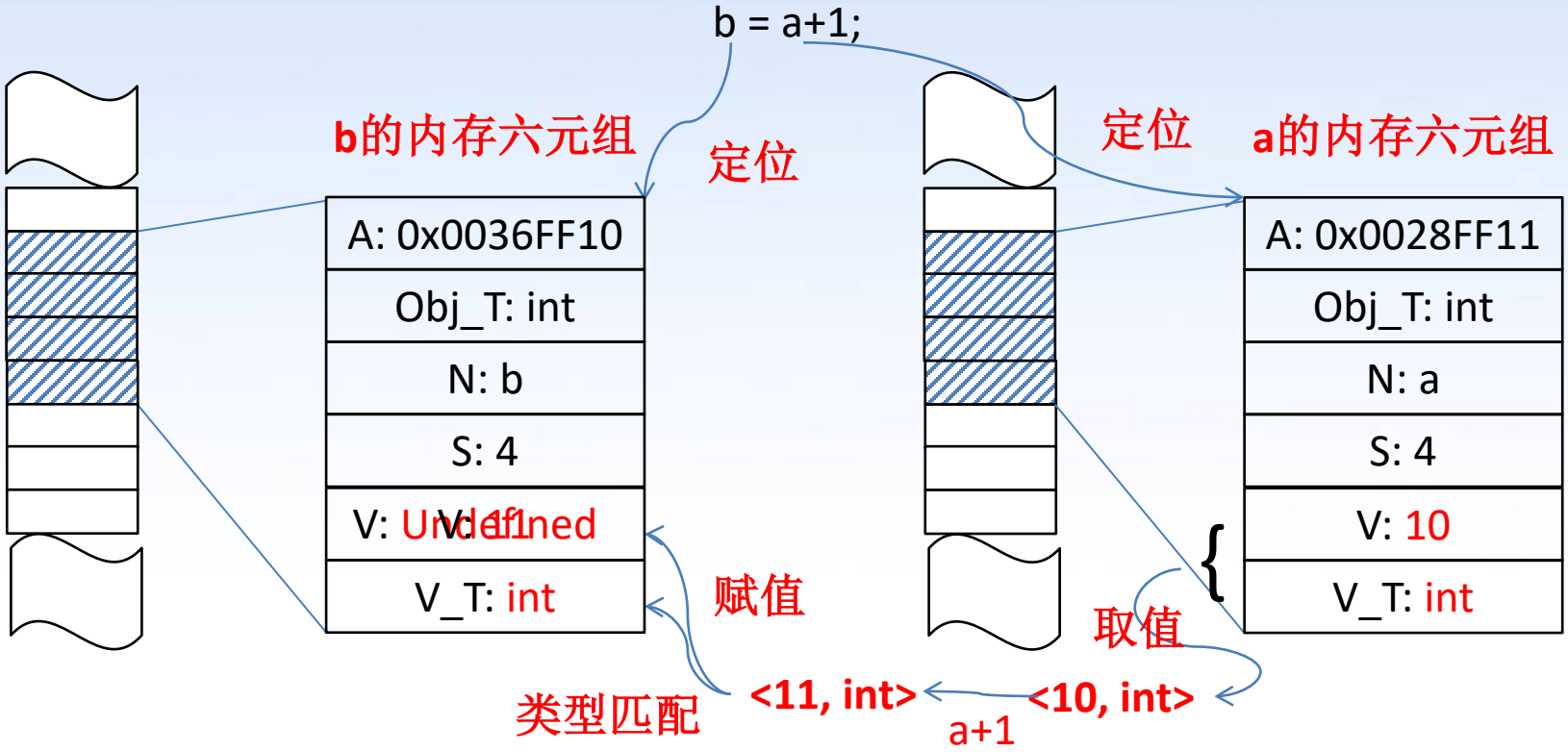
int a = 10;

1、变量a的地址是0x0028FF11 -> 变量a对应的内存的地址是0x0028FF11

2 、变量a的大小是4 -> 变量a对应的内存的大小是4

3、变量a的值是10 -> 变量a对应的内存的表示值是10

int b=a+1的结果是什么？



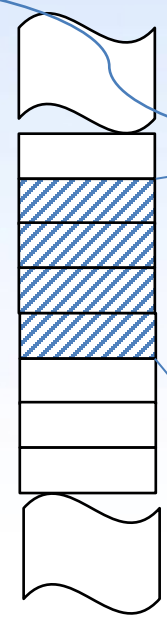
&(a+1)：这个表达式有值吗？

&(a+1);

只能对一块内存进行&操作

×

&



定位

a的内存六元组

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

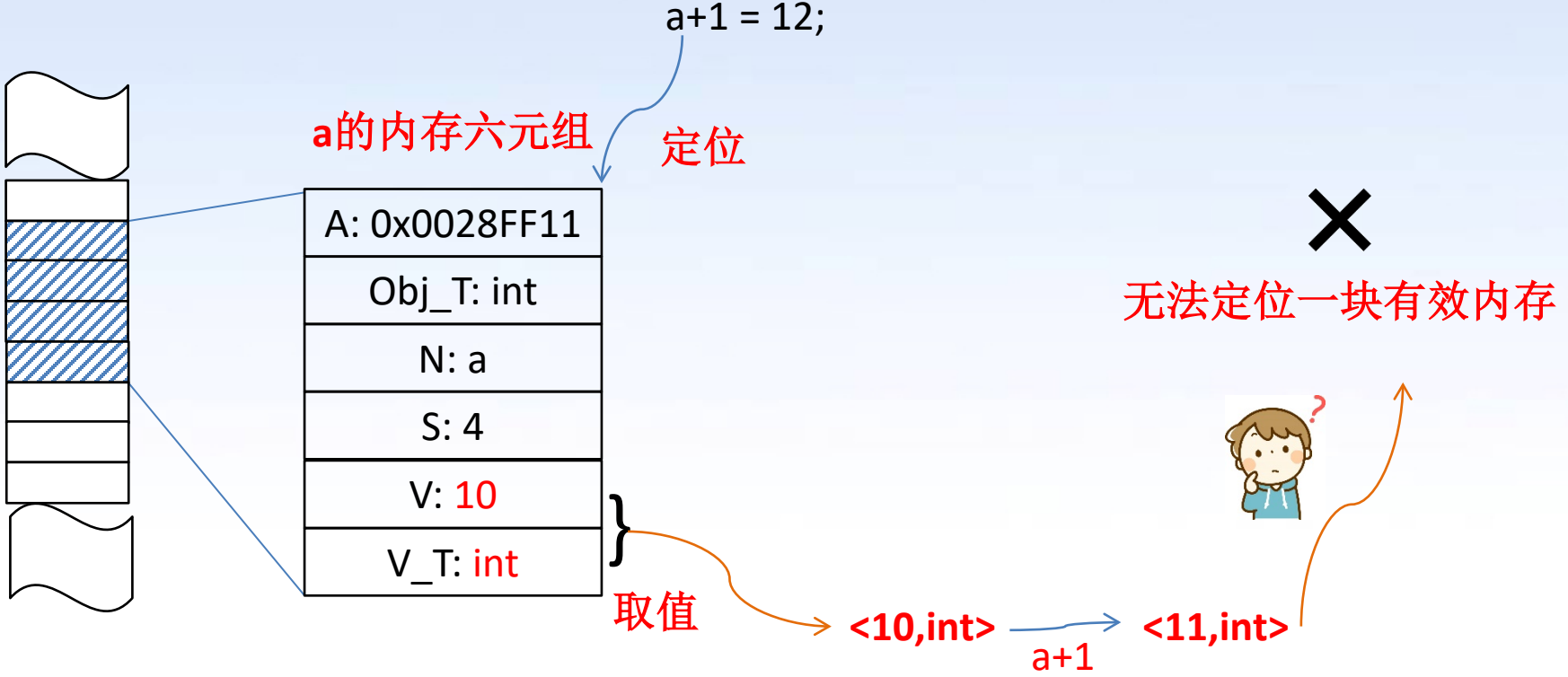
取值

<11, int>

<10, int>

a+1

a+1=12会怎么样


a+1 = 12;
a的内存六元组
定位
A: 0x0028FF11
Obj_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

思考题

1、float b;
float c;
&b=&c是否有错？

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

答案：
1、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？

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

1、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