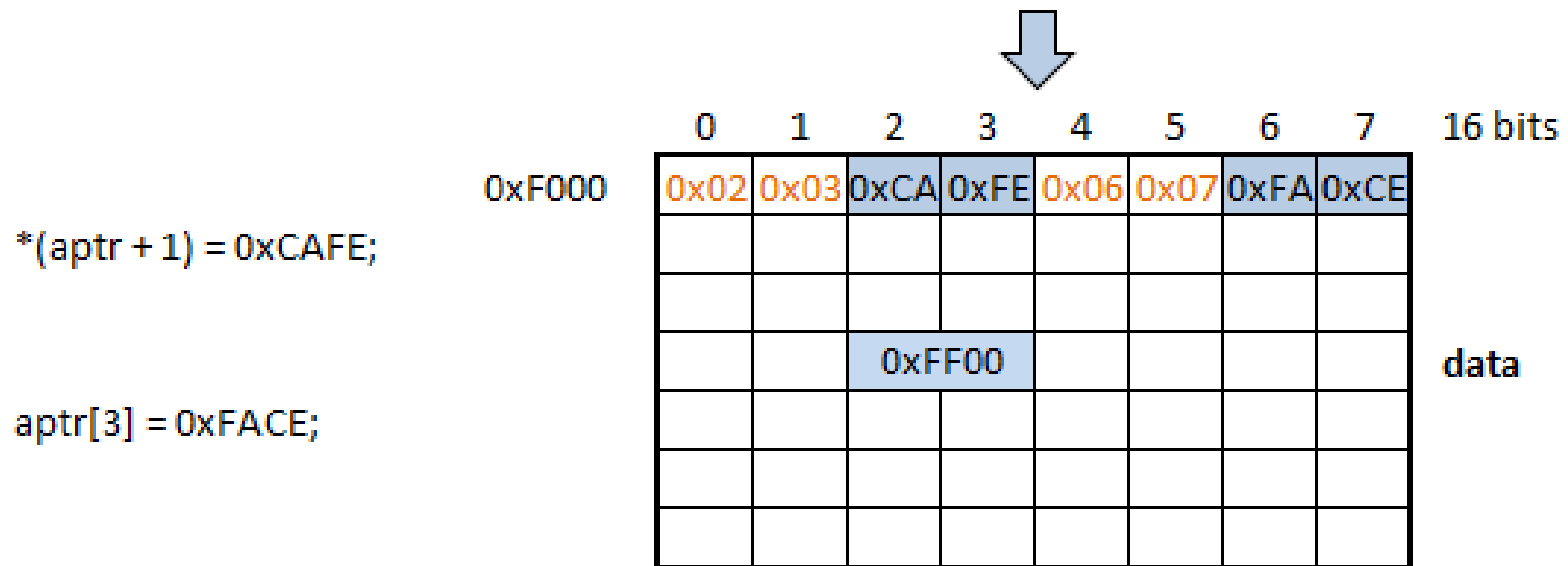
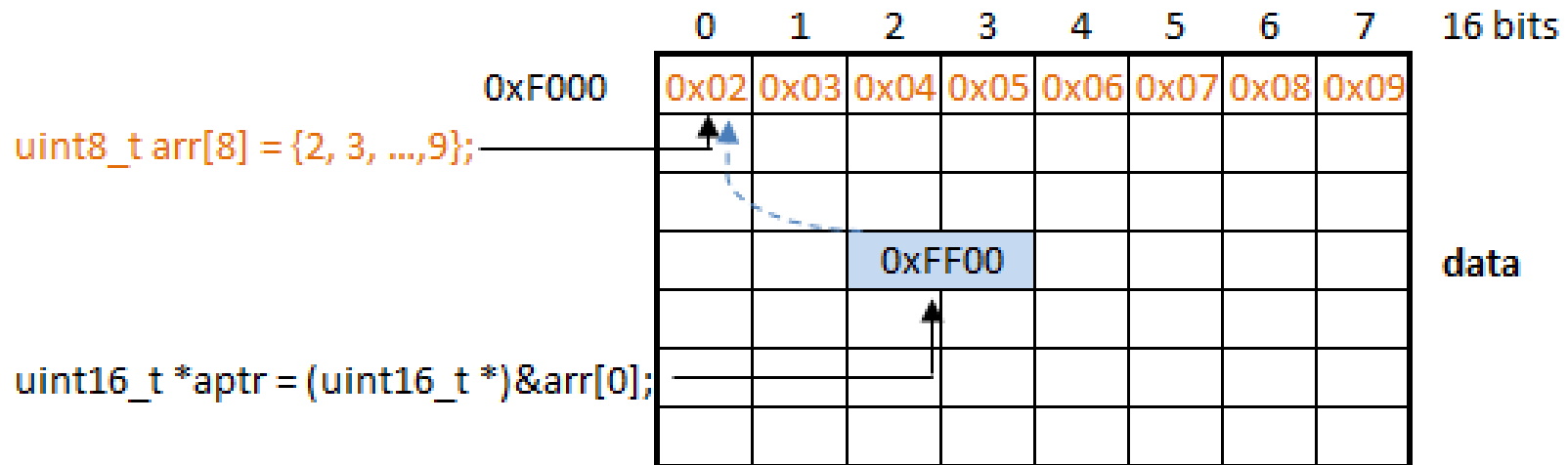


## Pointer Advance

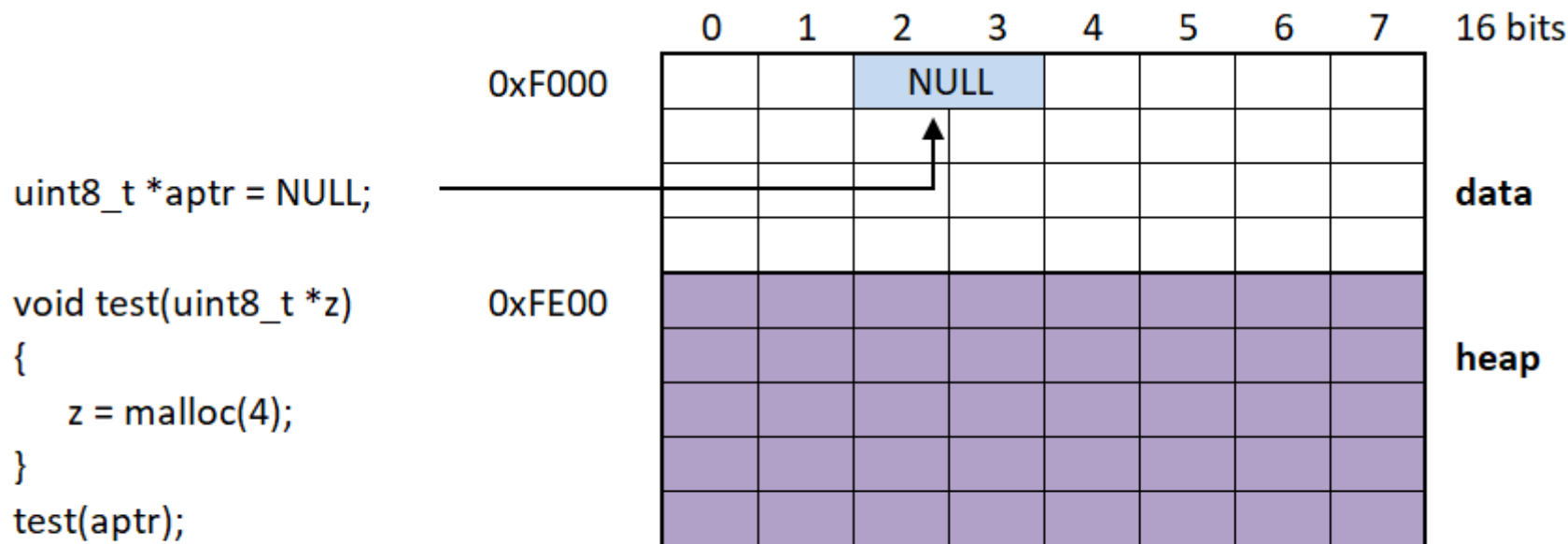
HaiND1

- Assigning pointer to address
- Wrong using pointer
- Pointer essence
- Function pointer
- Callback function

# Assigning pointer to address

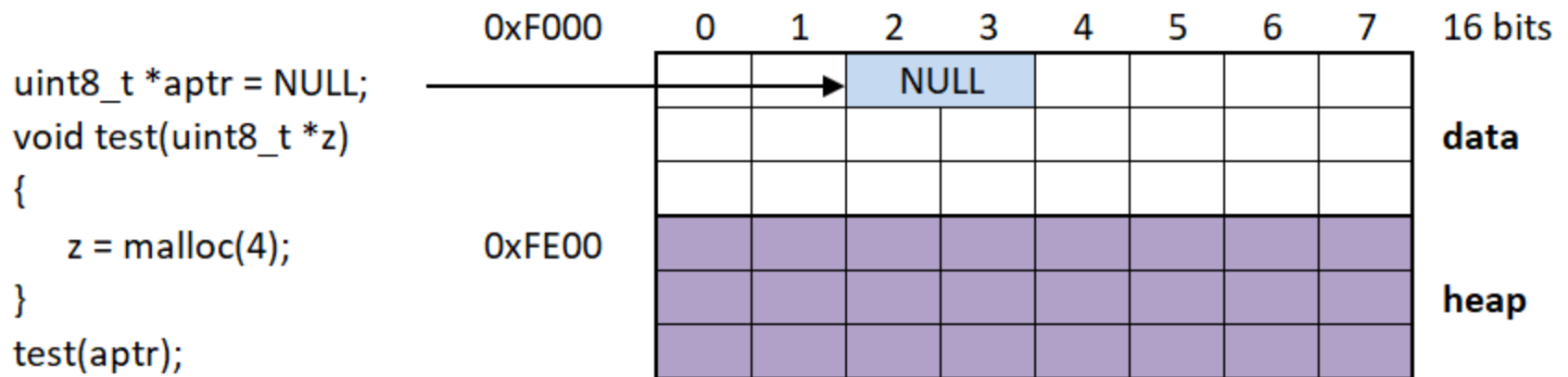


# Wrong using pointer

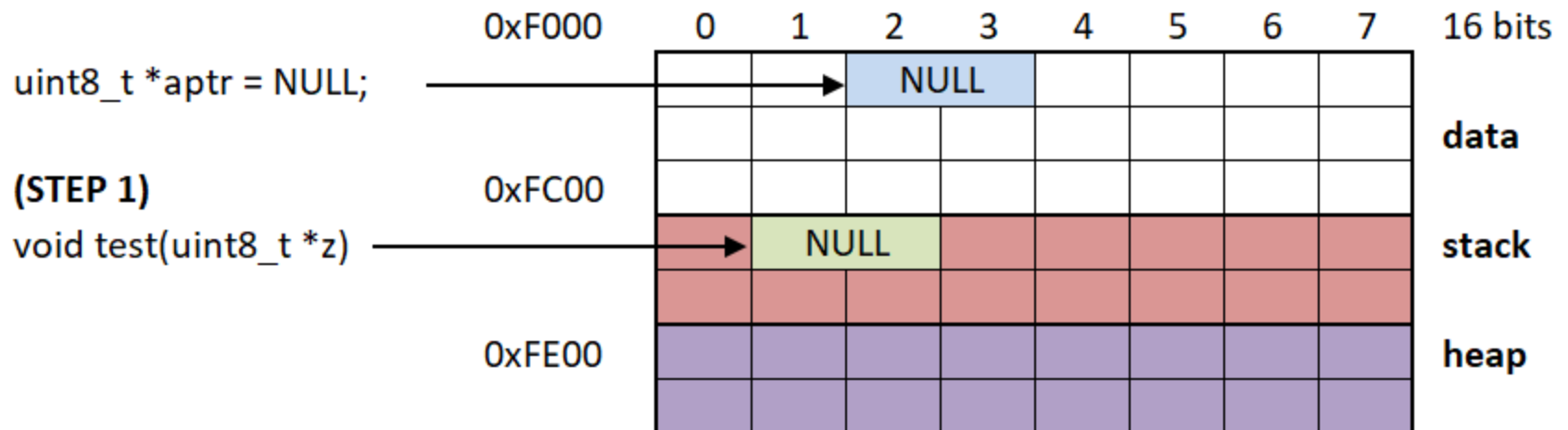


whether aptr pointer is allocated to 4 bytes of memory in heap section??

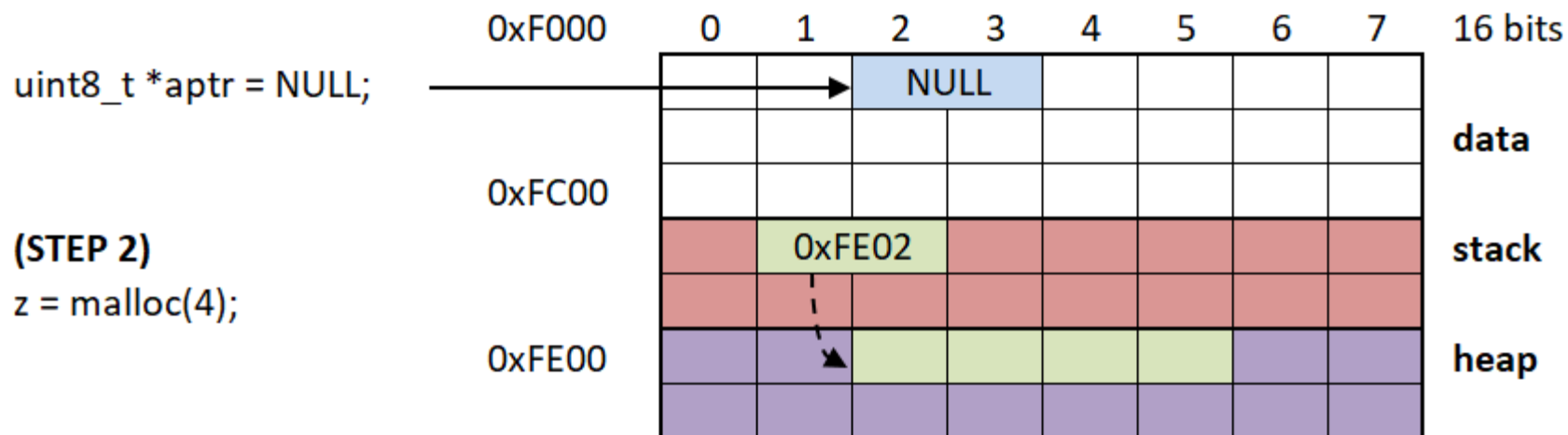
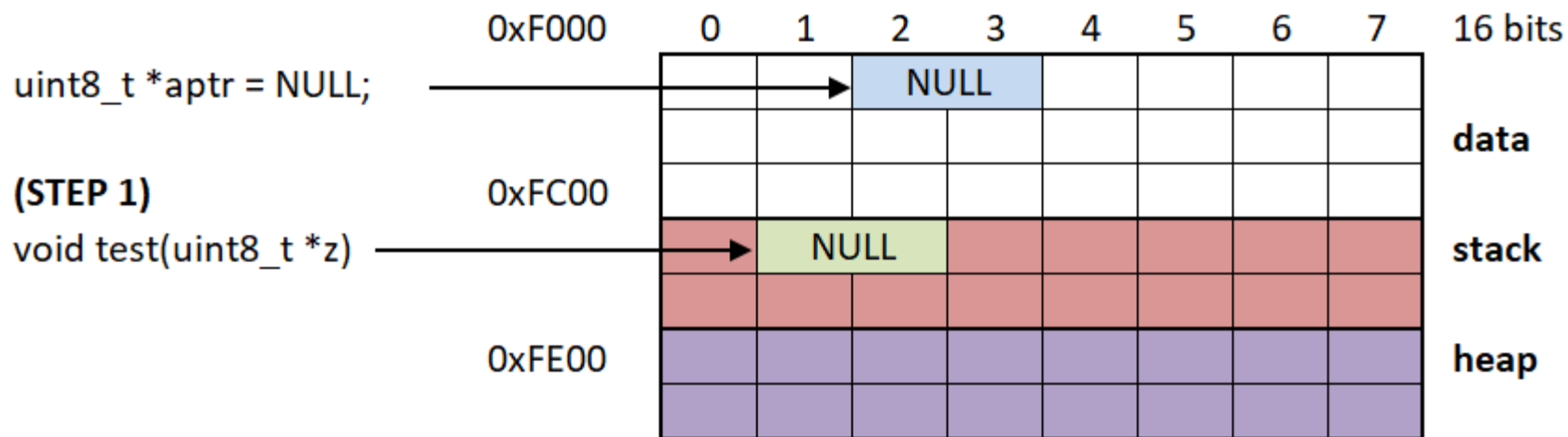
# Wrong using pointer



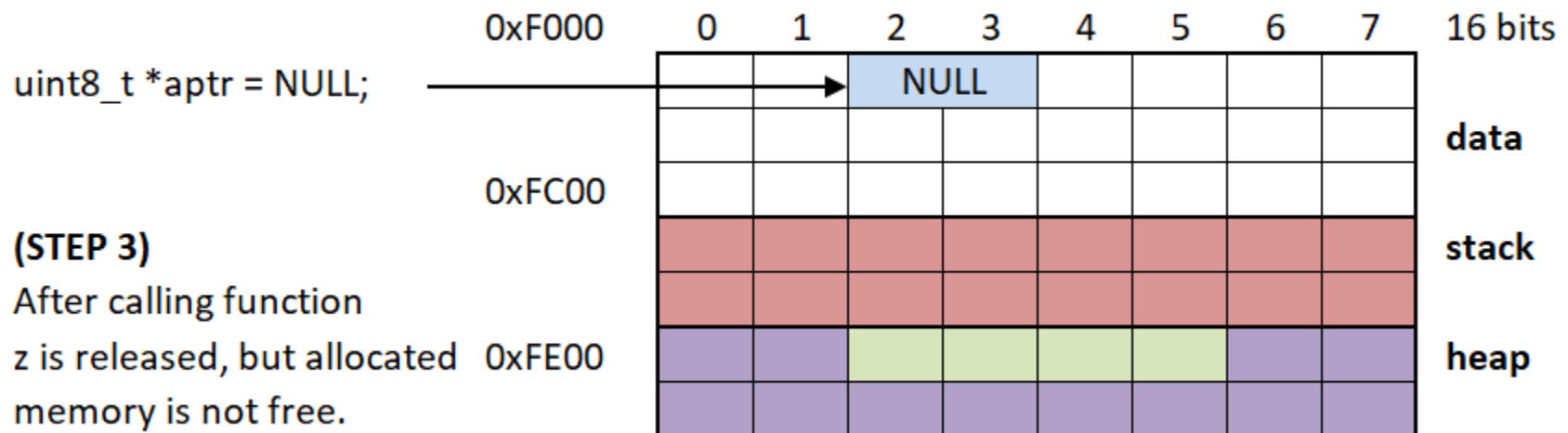
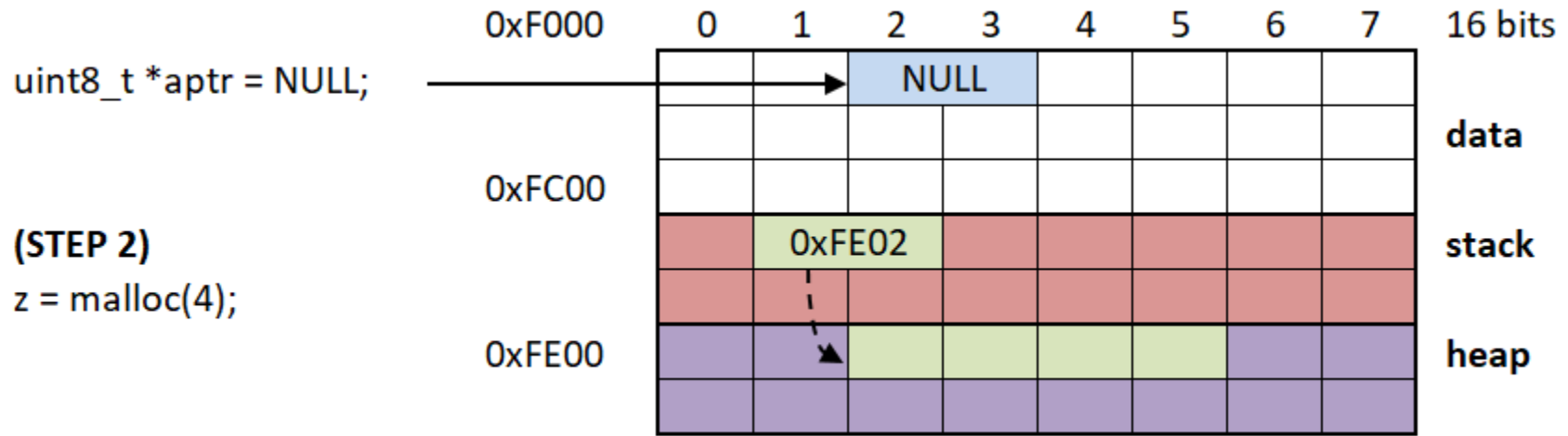
Apter calling test(aptr); -> aptr is still pointed to NULL. Lets explain how test function called.



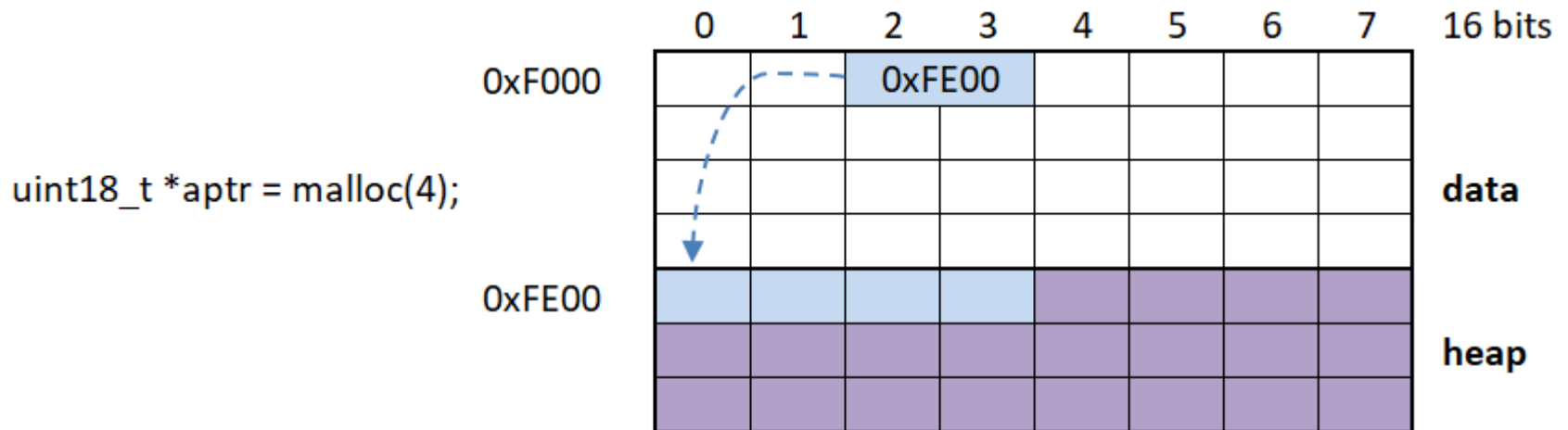
# Wrong using pointer



# Wrong using pointer



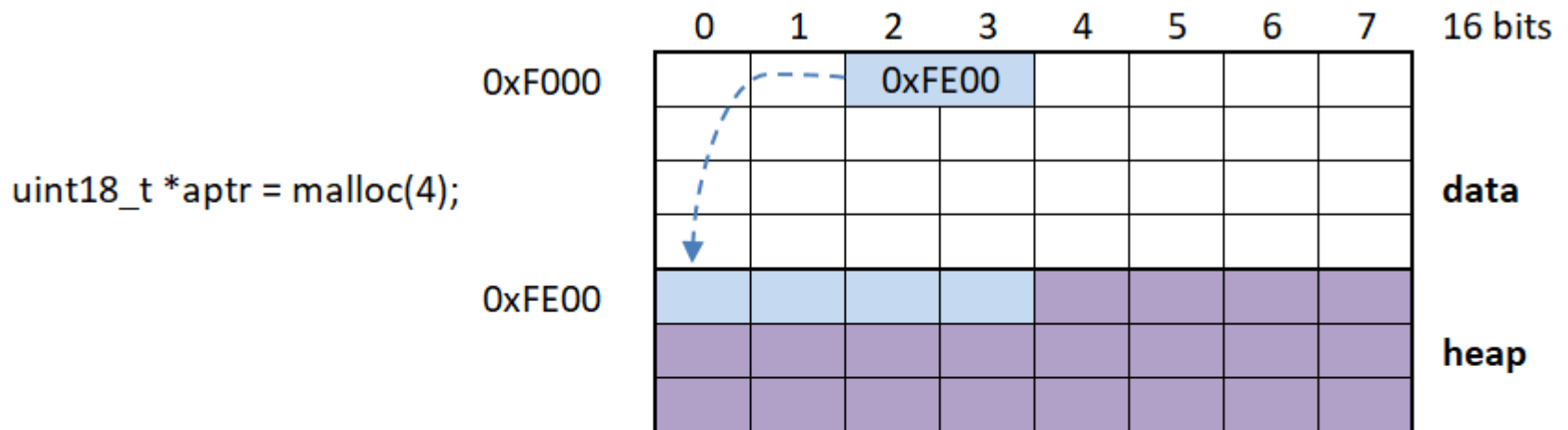
# Pointer essence



malloc function will return address of allocated memory to pointer, here is 0xFE00.  
 this address will be updated to content of memory cell where aptr variable stored.  
 --> **aptr = malloc(4);** <=> **aptr = 0xFE00** <=> **\*(0xF002) = 0xFE00.**



# Pointer essence



Apply this essence of memory allocation to test function.

```
void test(uint8_t *z)
```

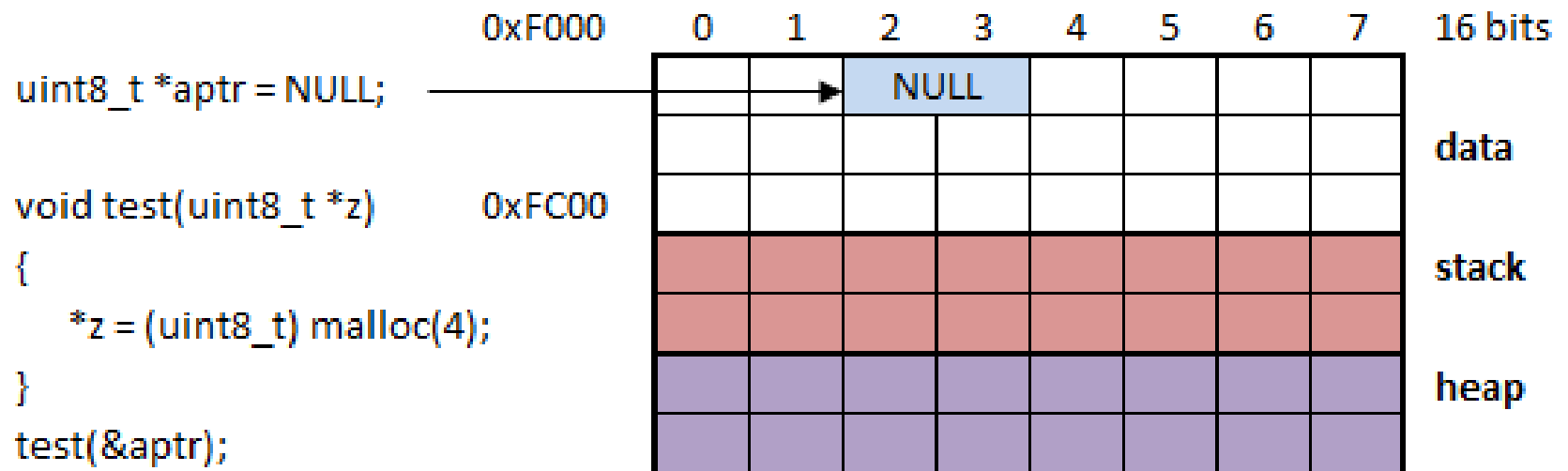
```
{
```

```
    ...
```

in body of test, we must to have coressponding code with  
 \*(0xF002) = malloc(4); -> we must pass 0xF002 to z parameter  
 of test function -> test(0xF002) <=> **test(&aptr)**

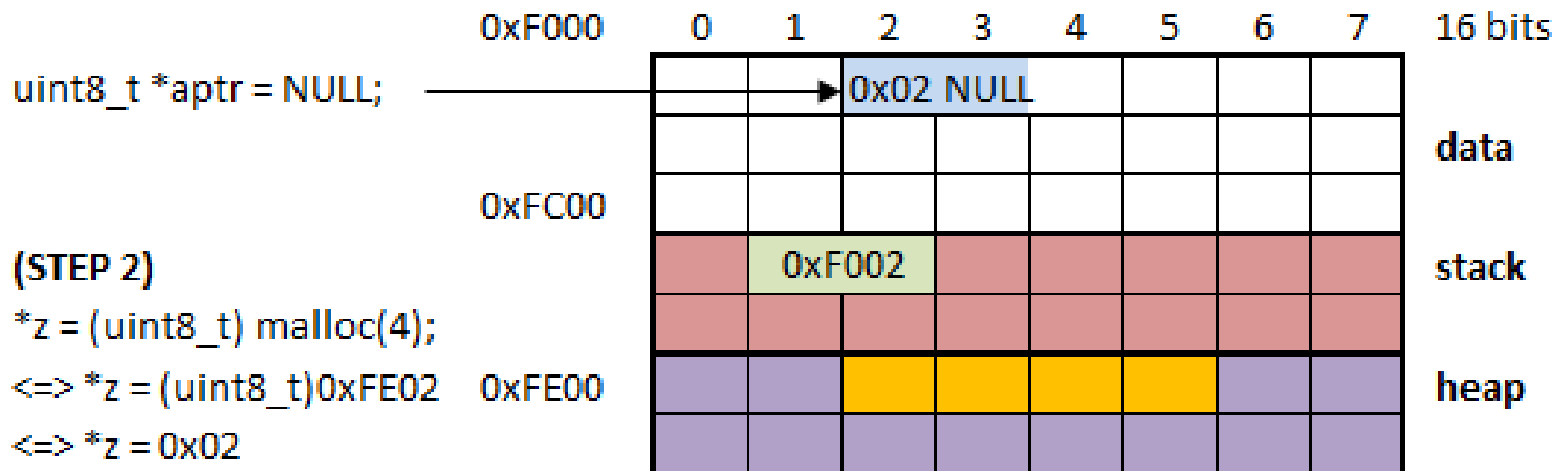
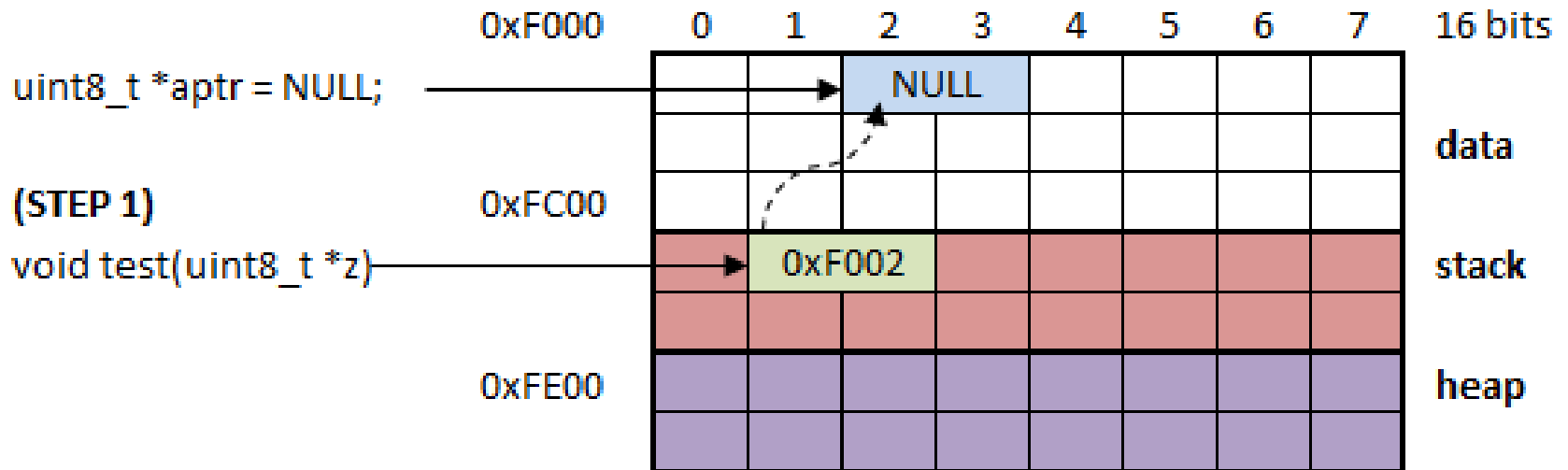
```
}
```

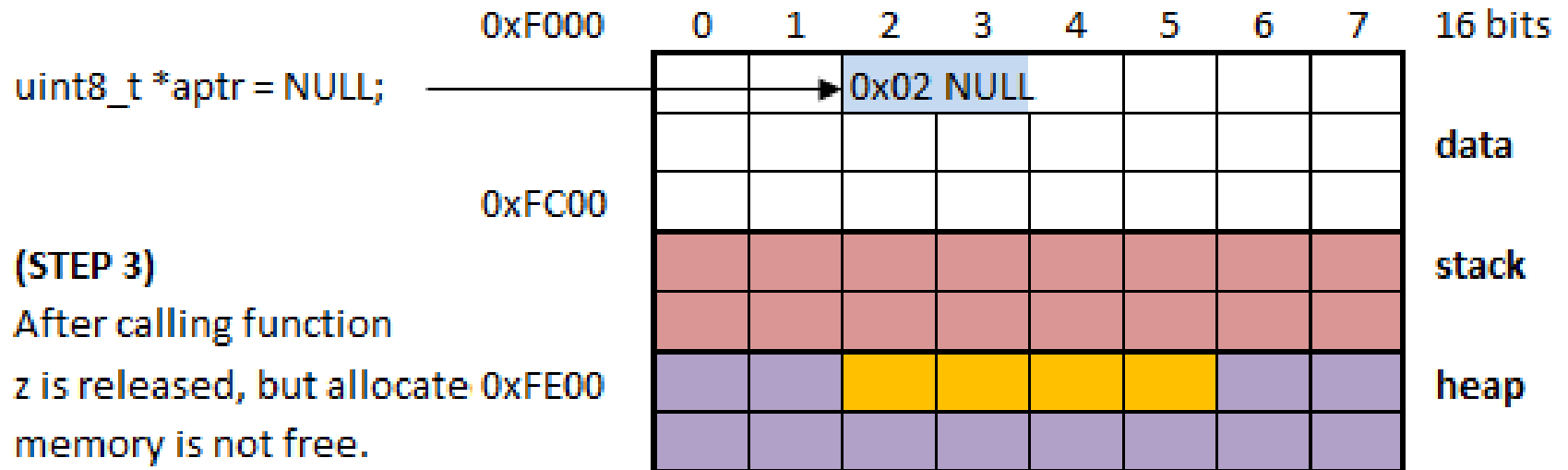
# Pointer essence



There is still something wrong? Lets see how test function called again.

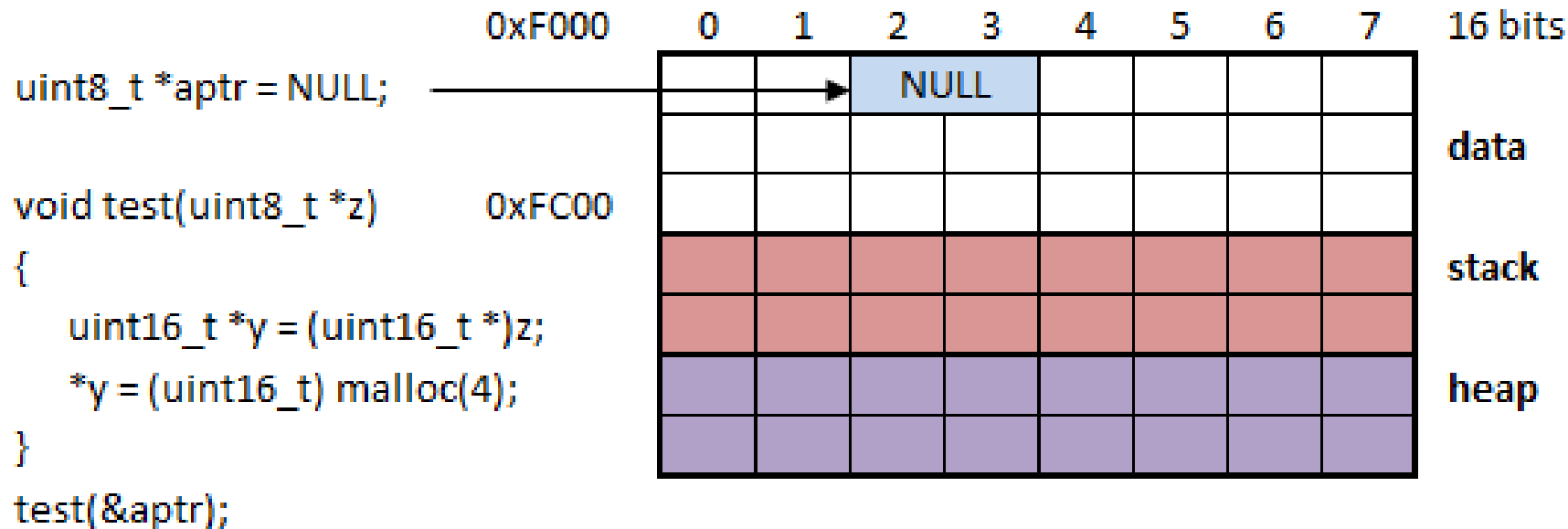
# Pointer essence





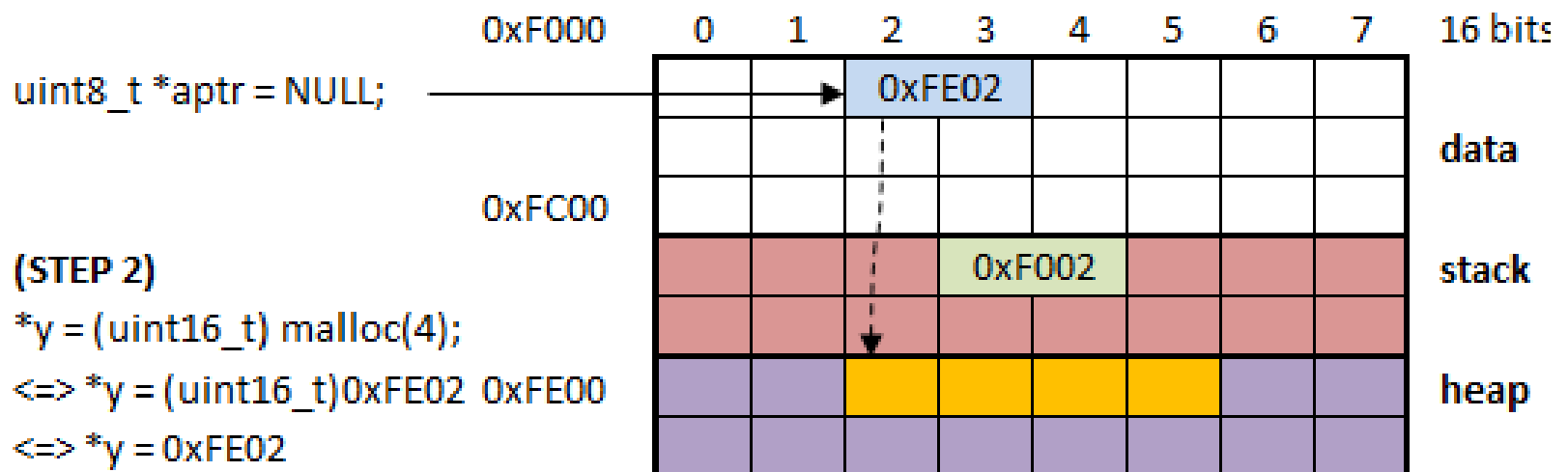
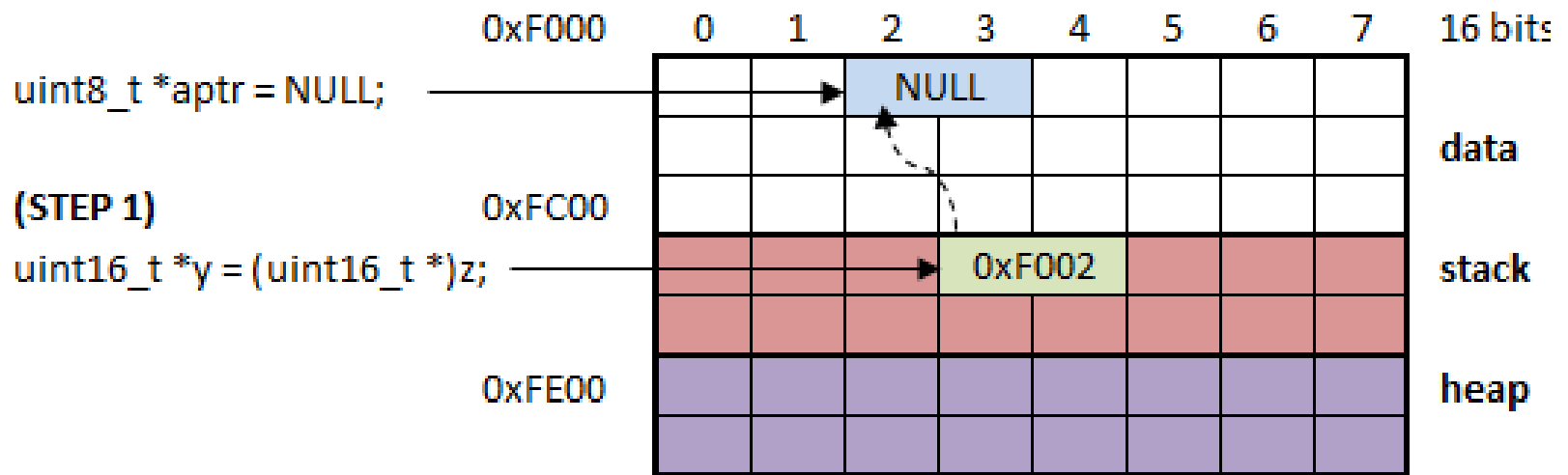
The pointer `aptr` is different from 0 now, but it still not pointed to 0xFE02 as expected. Now the memory cell 0xF002 is updated to 0x02 value, but it's not expectation, it require cell 0xF002 is updated to 0xFE, cell 0xF003 is updated to 0x02.

# Pointer essence

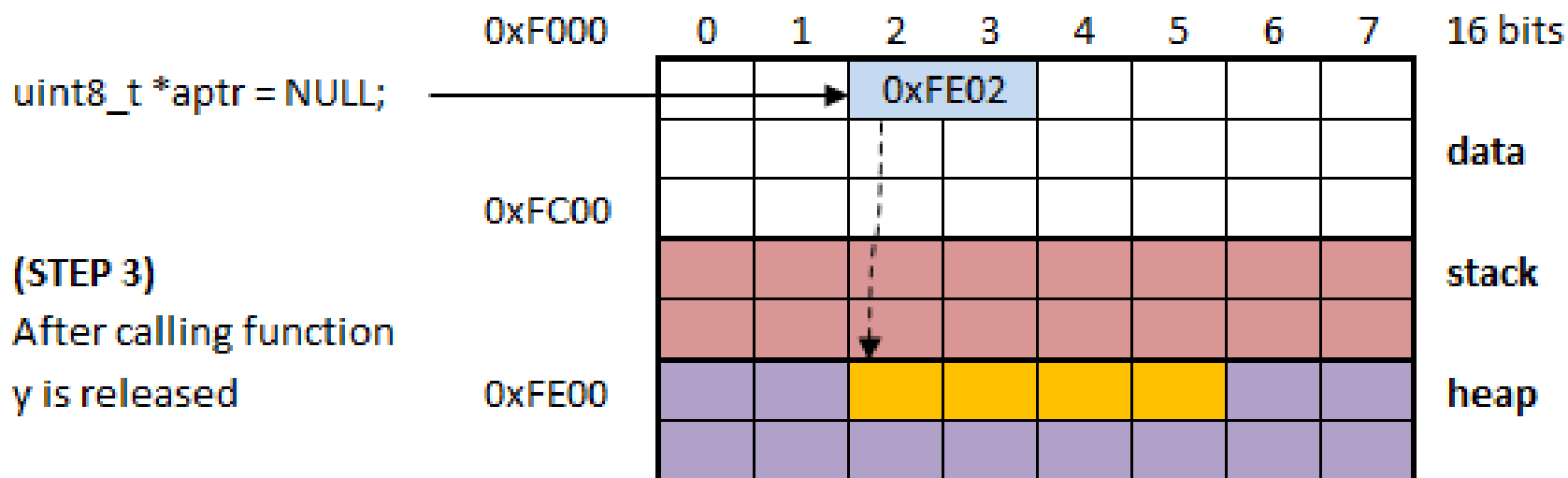


Local variable `uint16_t *y` will help to resolve current issue. Lets see how test function called again.

# Pointer essence



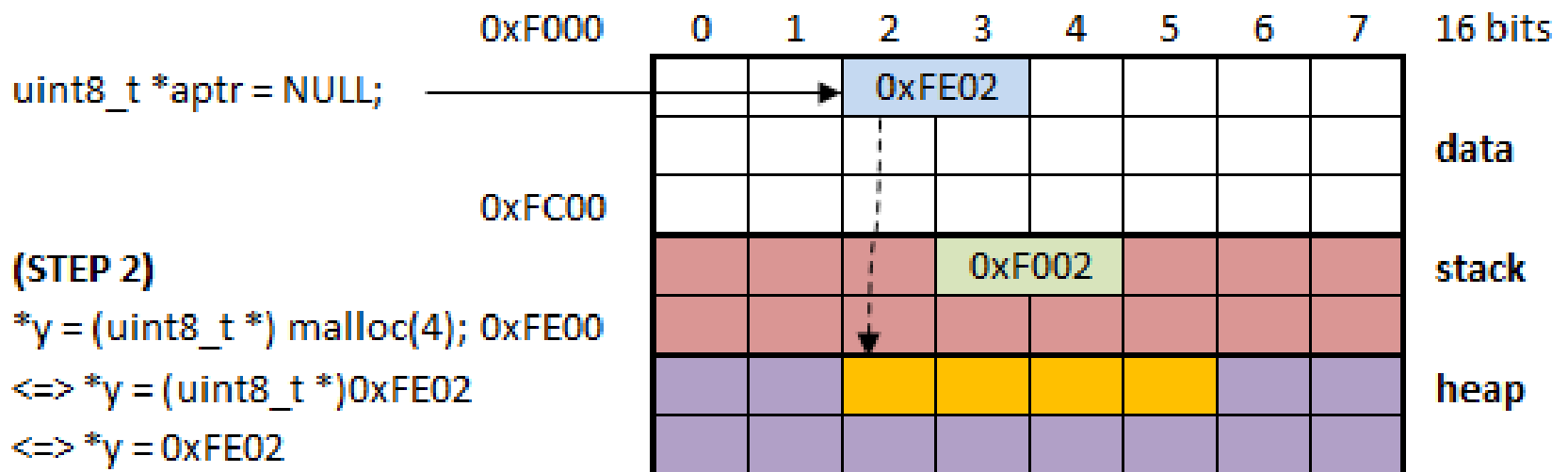
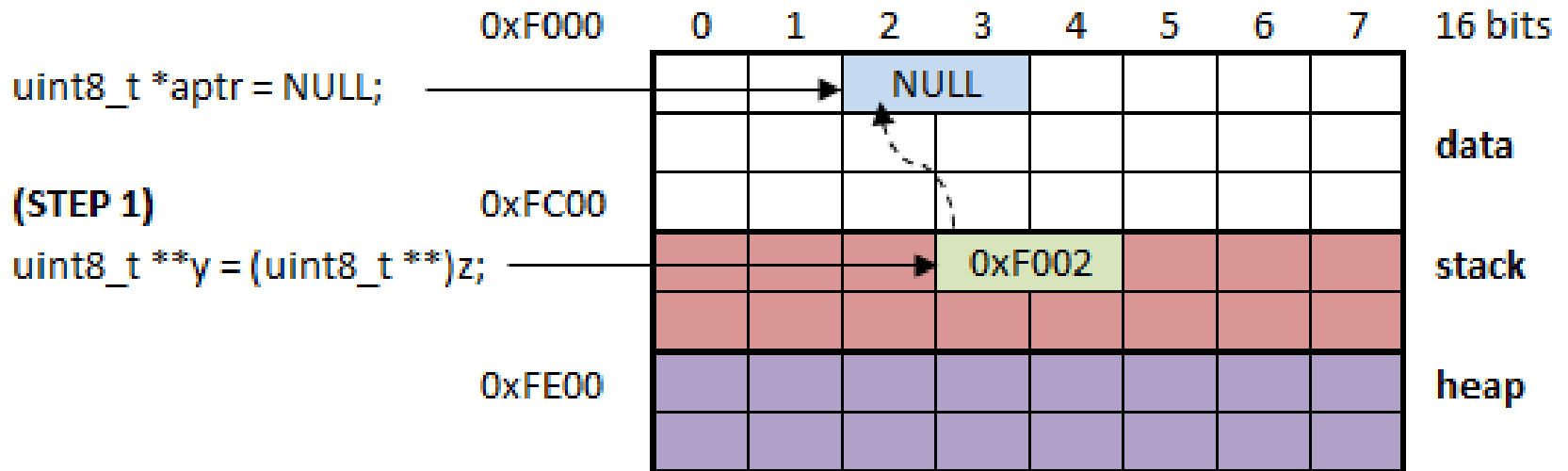
# Pointer essence



Now, aptr pointer pointed to 0xFE02 as expected (0xFE02 is assumed as return of malloc function)

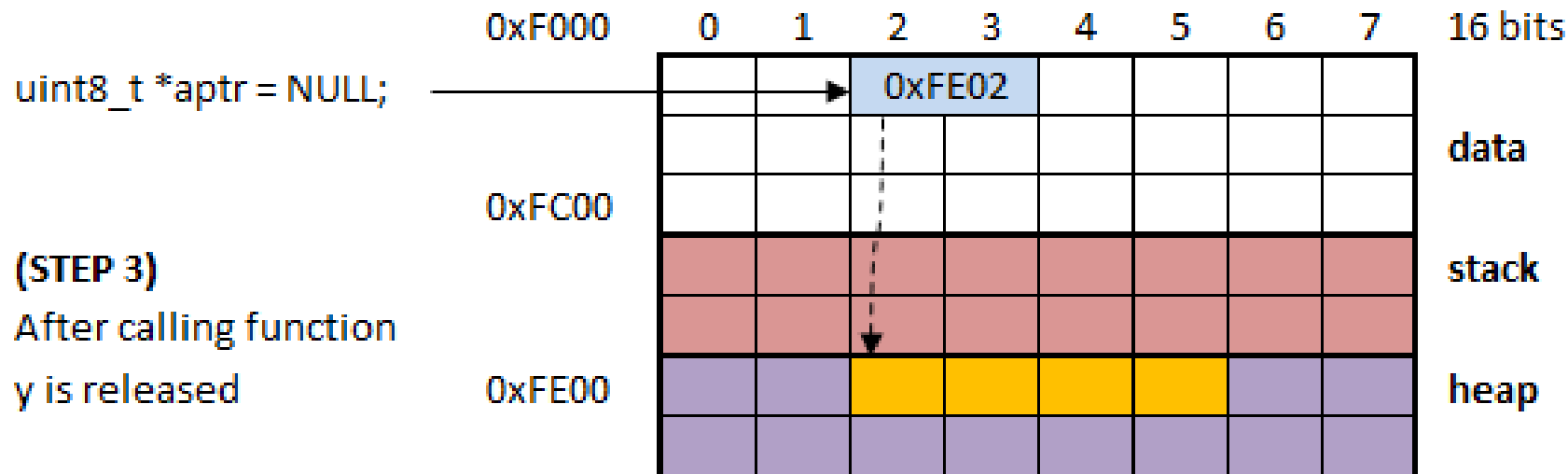
But this solution is applied to 16 bits chip only, it will be failed for 32 bits chip.  
Is there any common solution applied to both 16 bits and 32 bits?

# Pointer essence





# Pointer essence

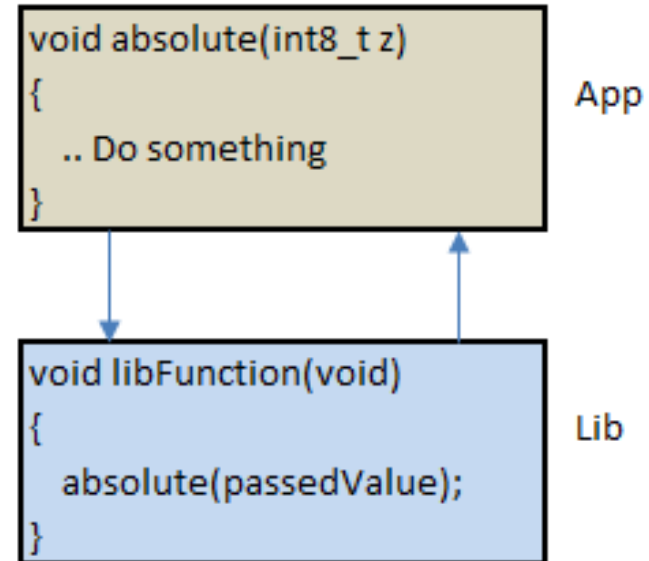


Now, aptr pointer pointed to 0xFE02 as expected (0xFE02 is assumed as return of malloc function). And this solution is applied to both 16 bits and 32bits of chip.

**Finally, to optimize the code without declaration of \*y variable, the allocate function is**

```
void test(uint8_t **z)
{
    *z = (uint8_t *)malloc(4);
}
```

Lets say about the situation that library need to call to a function in application to do something. Here lib is calling absolute function directly, what is the problem?



# What is function pointer

- ❖ Function pointer is a data type defined by user.
- ❖ A function pointer variable is a variable that stores address of a function.
- ❖ Function pointer is a pointer but cannot be allocated/de-allocated.

# How to define function pointer

```
void test(void)
```

```
→ typedef void (*fPointer)(void);
```

```
void test1(uint8_t x)
```

```
→ typedef void (*fPointer1)(uint8_t x);
```

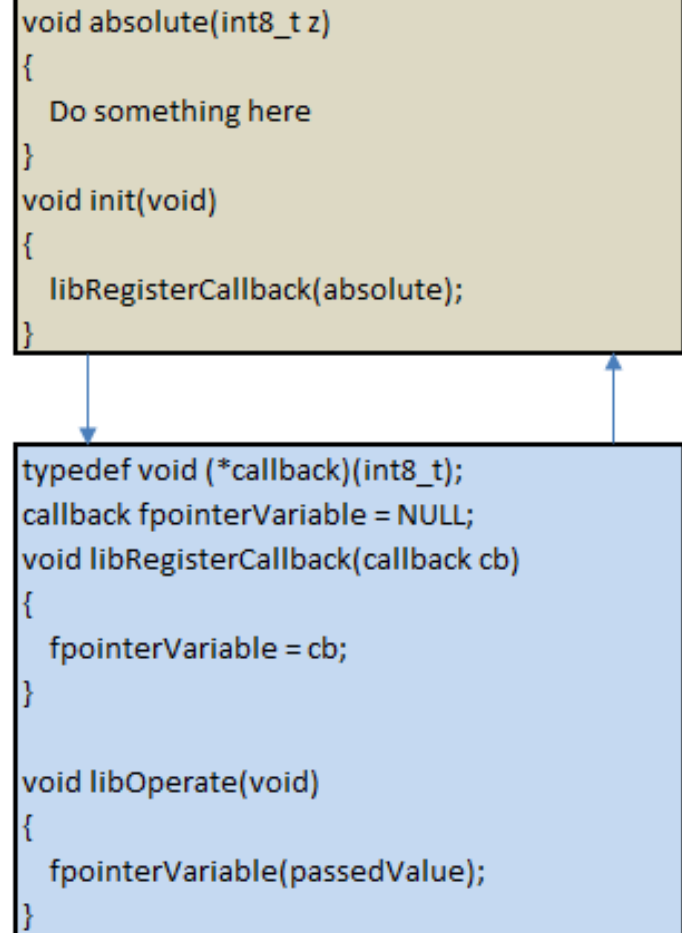
```
uint16_t test2(uint8_t x, uint32_t y)
```

```
→ typedef uint16_t (*fPointer2)(uint8_t x, uint32_t y);
```

```
...
```

- ❖ Initializes a callback function by calling libRegisterCallback function provided by lib. The address of absolute function will be passed as input parameter to register to fpointerVariable in the lib layer.
- ❖ Lib uses registered fpointerVariable to call to absolute instead of calling it directly.
- ❖ This helps lib source code is independently from application.

```
void absolute(int8_t z)
{
    Do something here
}
void init(void)
{
    libRegisterCallback(absolute);
}
```



```
typedef void (*callback)(int8_t);
callback fpointerVariable = NULL;
void libRegisterCallback(callback cb)
{
    fpointerVariable = cb;
}

void libOperate(void)
{
    fpointerVariable(passedValue);
}
```

# Q & A