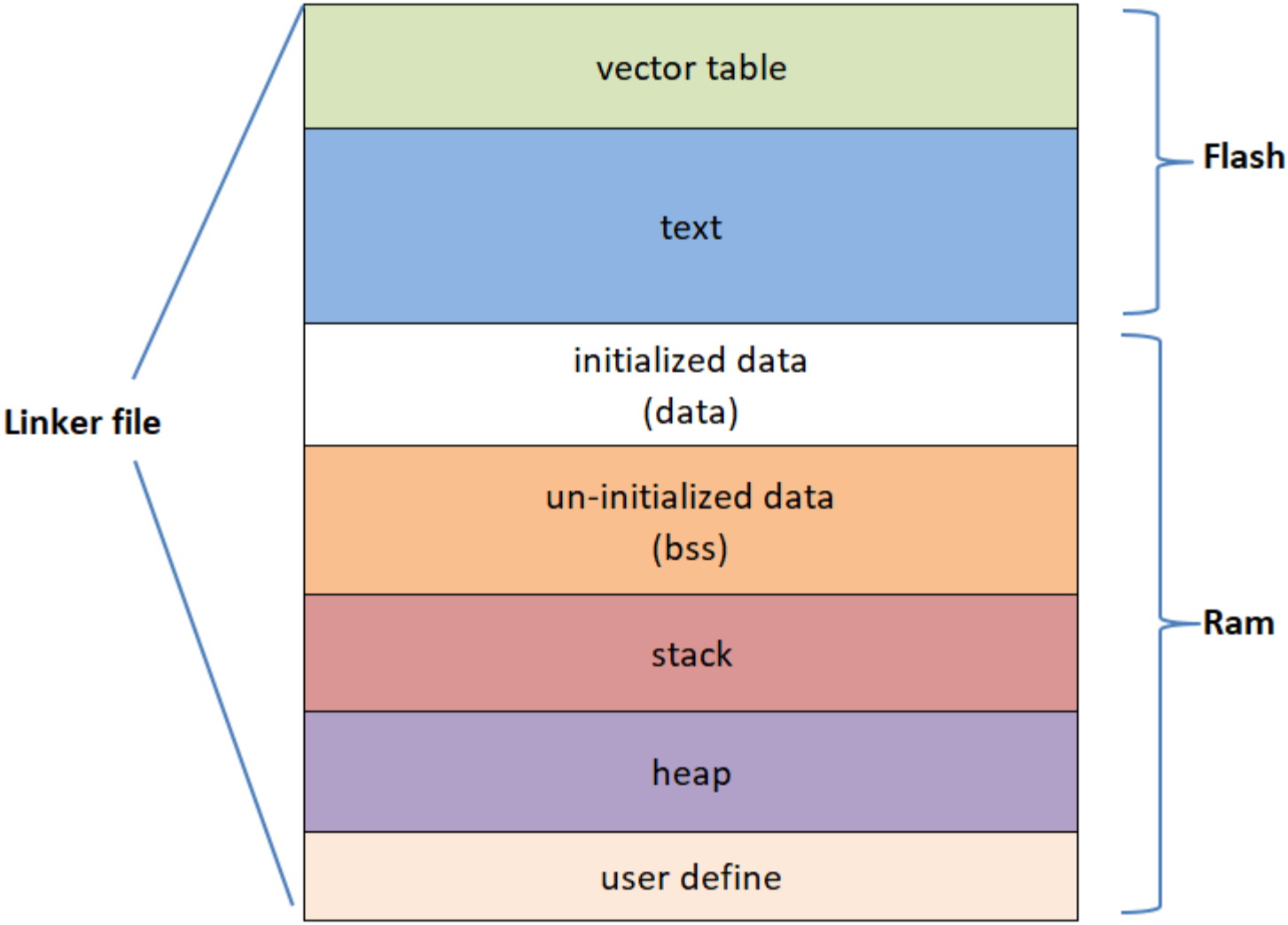
Memory management & Pointer basic

HaiND1

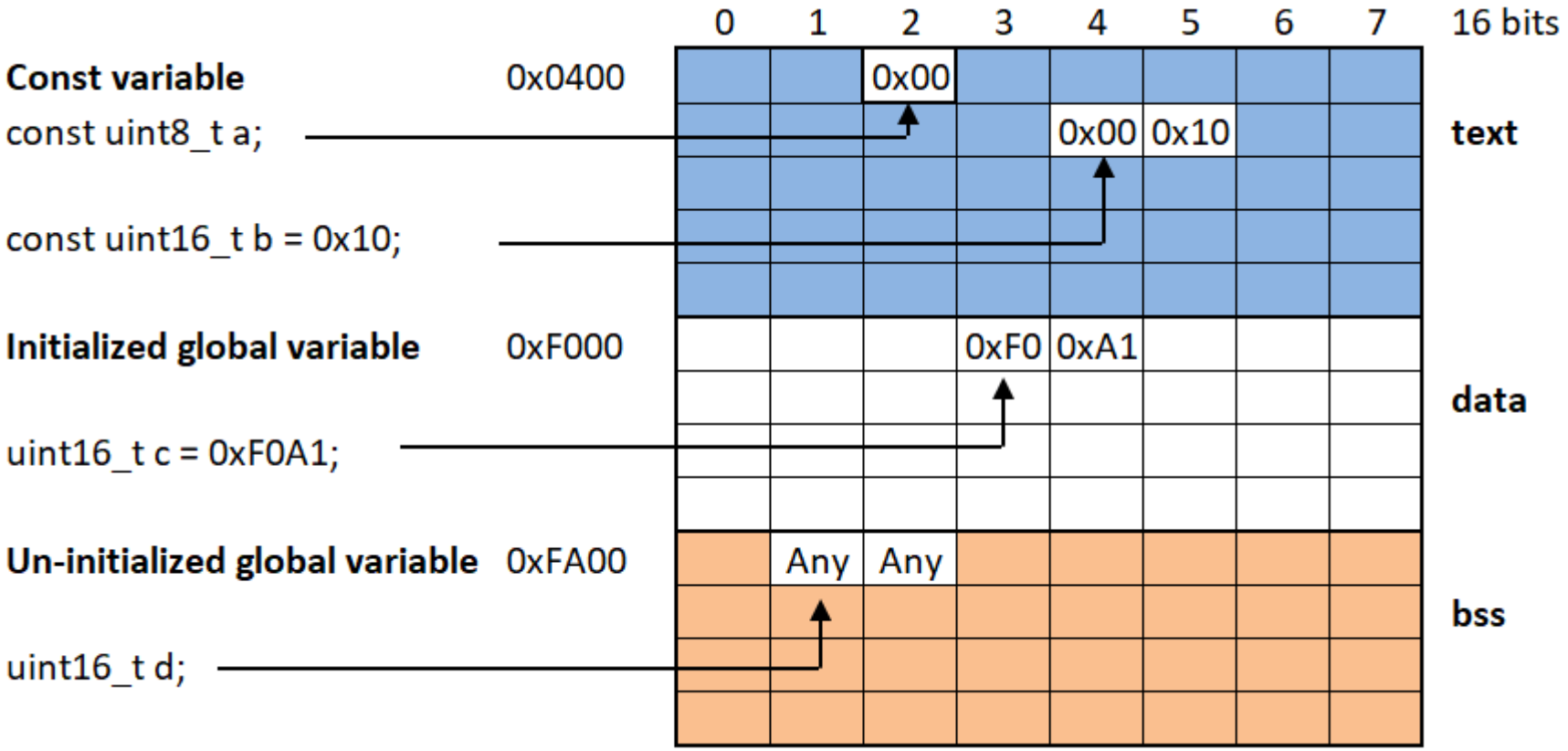
Objectives

- Memory layout
- Variable and memory location
- Linker file and memory
- Pointer variable
- Assigning values to a pointer
- Memory allocation for a pointer
- Pointer arithmetic

Memory layout



Variable and memory location

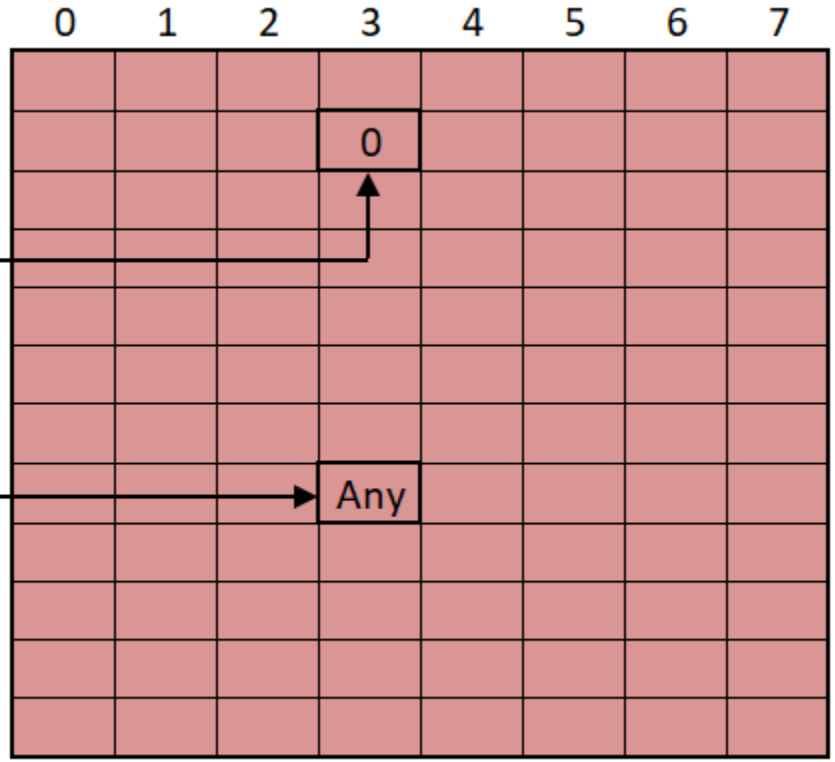


Variable and memory location

Local variable 0xFC00

```
void test(uint8_t x)    x not stored in stack
{                       stored in r1
        uint8_t y = 0;
        ...
}

uint8_t test1(uint8_t z)
{                       z stored in stack
        z = a + 5;
        ..
        return z;
}
```

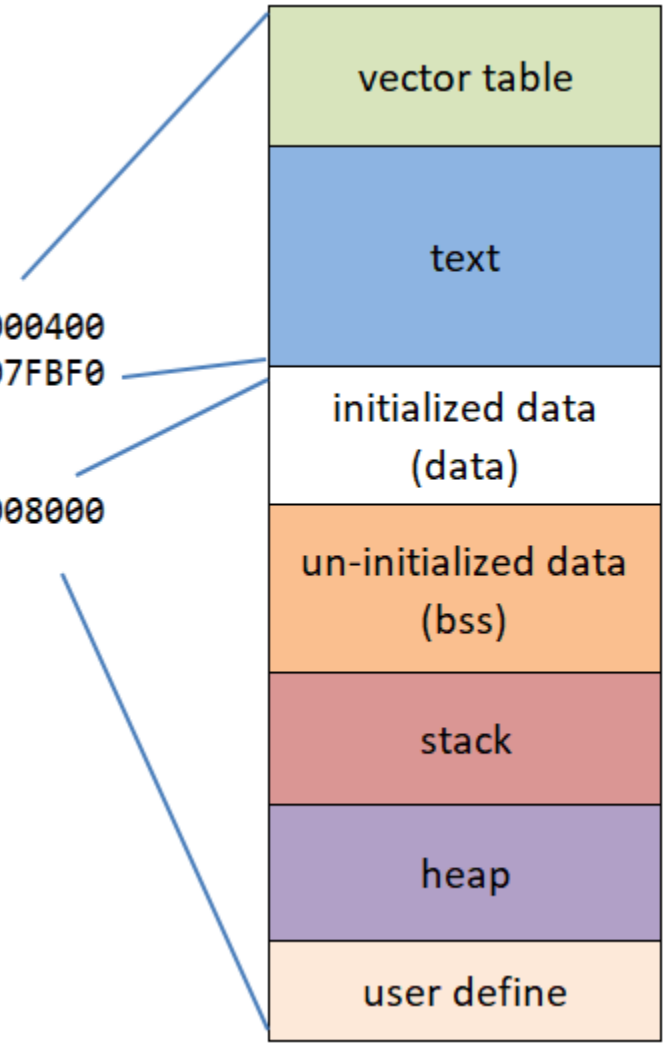


Linker file & Memory

Linker file specifies memory areas

```
MEMORY
{
  /* Flash */
  m_int   (RX) : ORIGIN = 0x00000000, LENGTH = 0x00000400
  m_text  (RX) : ORIGIN = 0x00000410, LENGTH = 0x0007FBF0

  /* Ram */
  m_data  (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00008000
}
```



Linker file & Memory

Linker file defines output sections, below is example:

```
.stack __StackLimit :
{
  . = ALIGN(8);
  __stack_start__ = .;
  . += STACK_SIZE;
  __stack_end__ = .;
} > m_data
```

"." means begin at
".stack __StackLimit:"
-> Stack section begins at __StackLimit address.

```
.mySection :
{
  __mySection_start__ = .;
  . += MY_SECTION_SIZE;
  __mySection_end__ = .;
} > m_data
```

".mySection" is user defined section.
-> ".mySection :" means "user defined section begins after stack section

Linker file & Memory

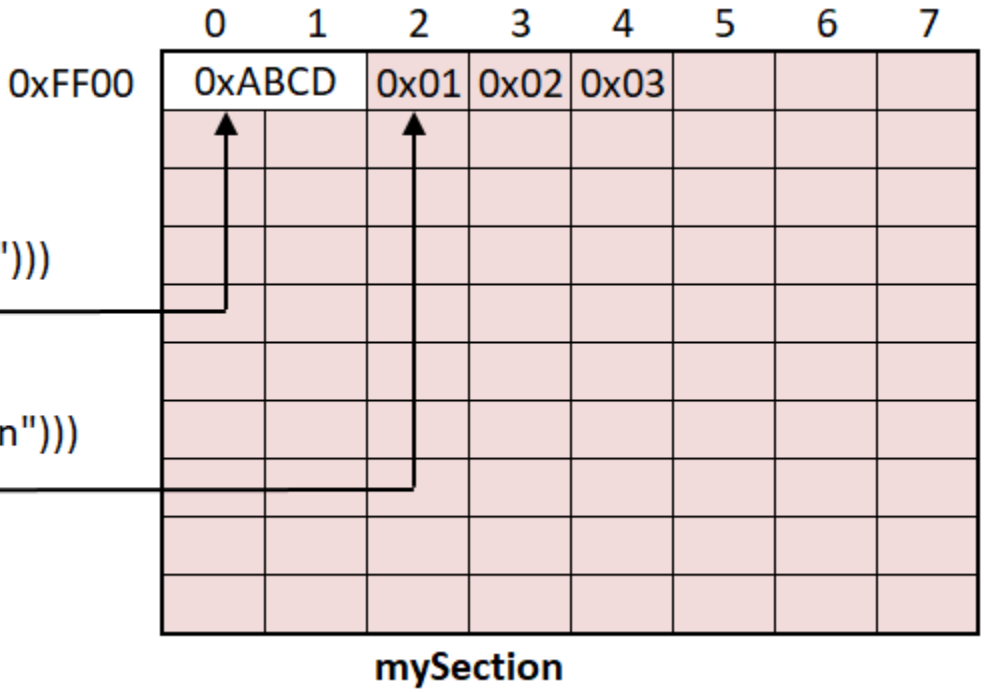
Variable in user defined section

Use instruction to put a variable into user defined section.

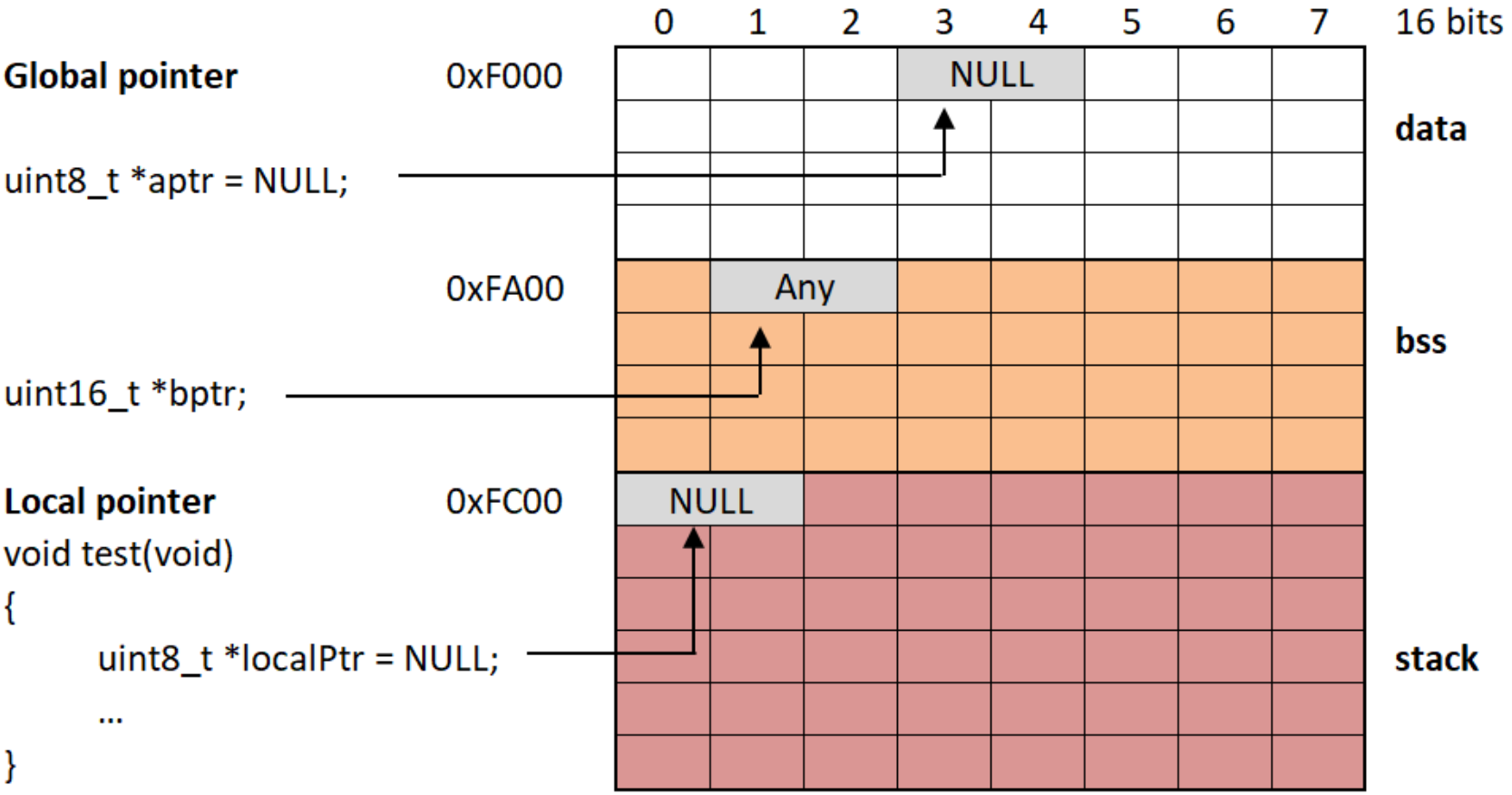
```
uint16_t x __attribute__((section (".mySection")))
        = 0xABCD;

uint8_t y[3] __attribute__((section (".mySection")))
        = { 0x01, 0x02, 0x03 };
```

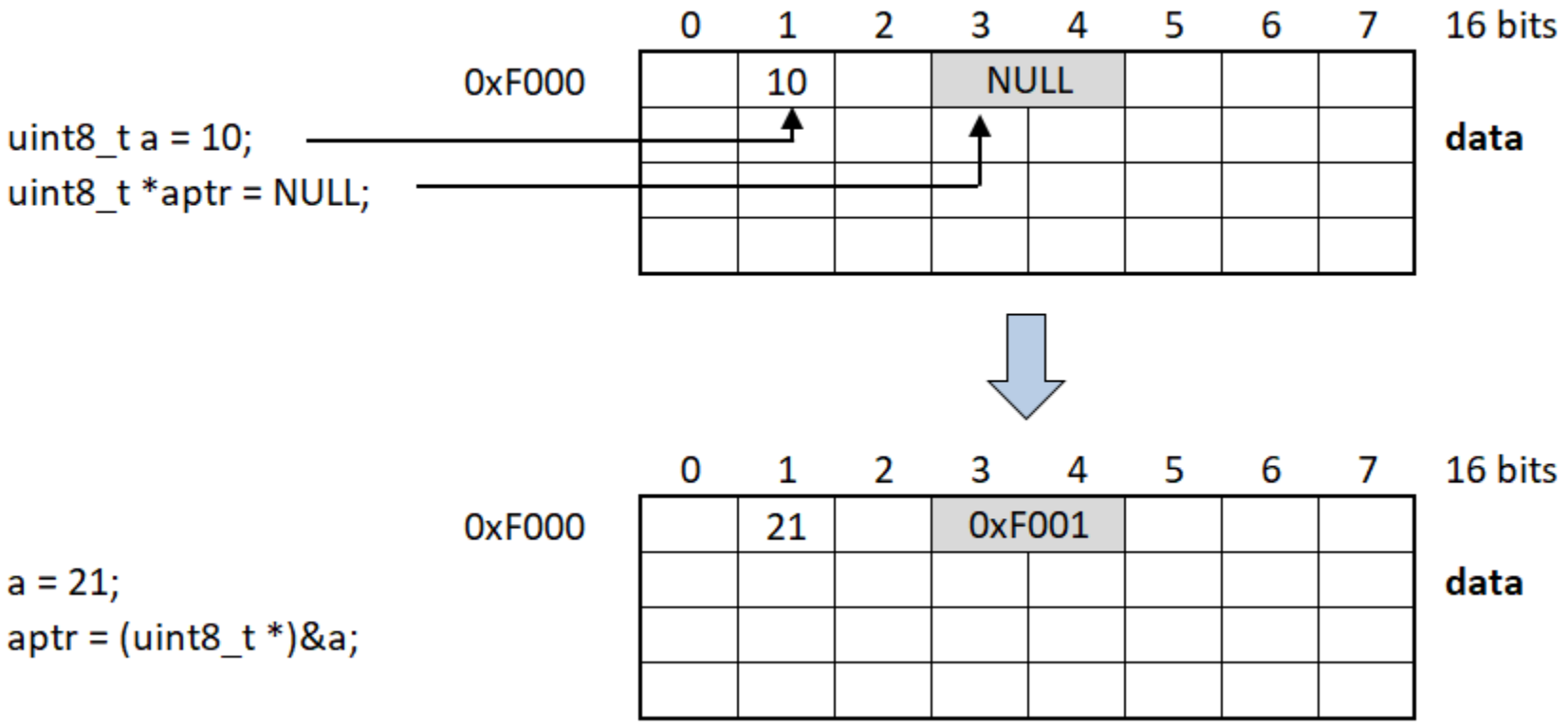
instruction is different between toolchains.



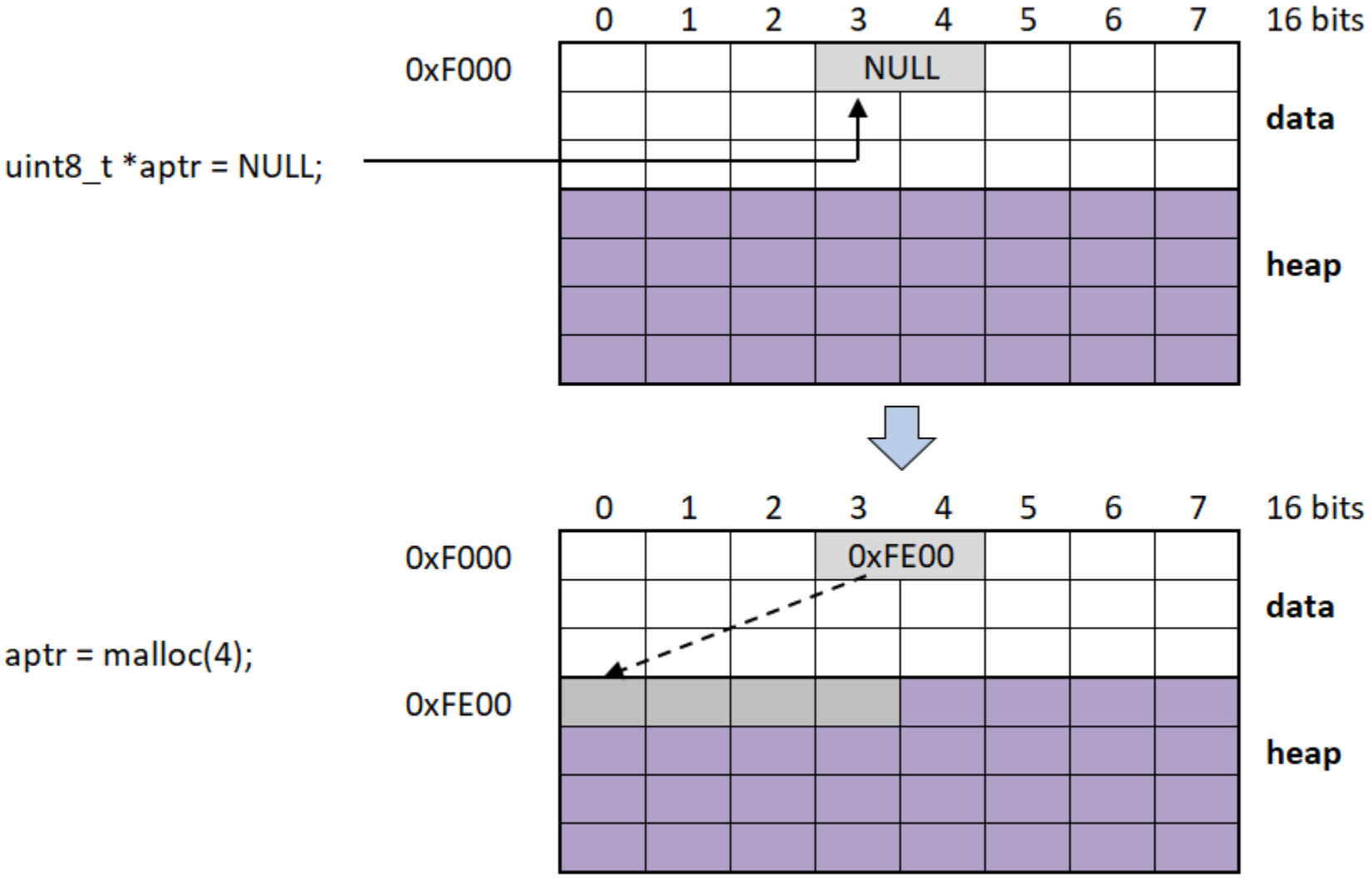
Pointer variable



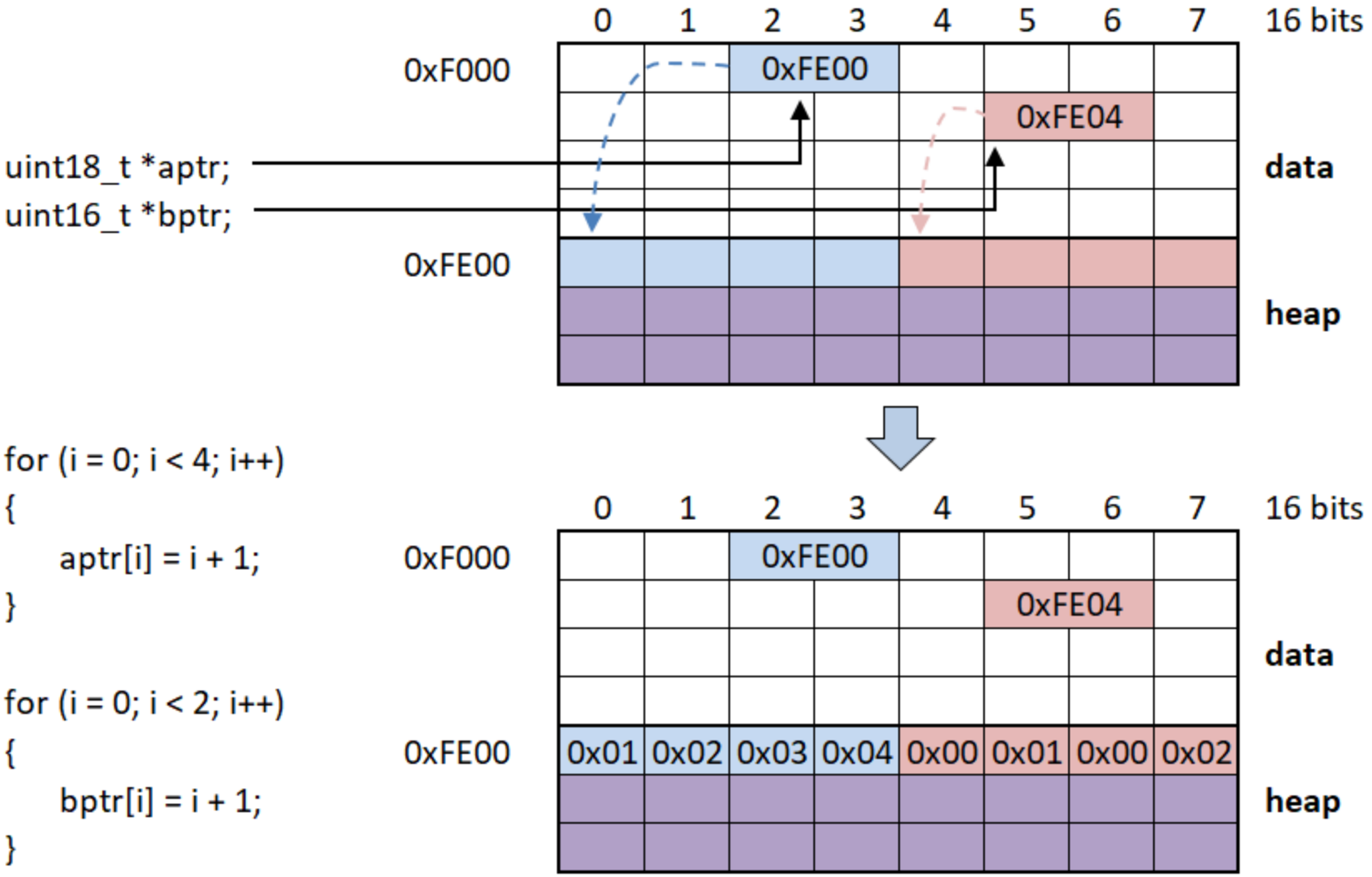
Assigning values to a pointer



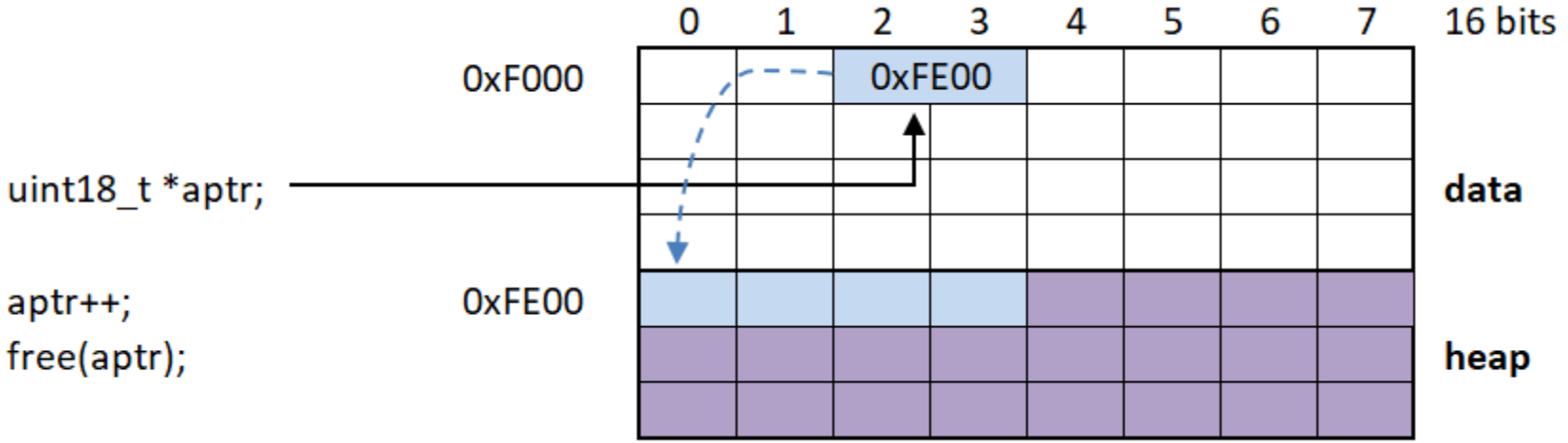
Memory allocation for a pointer



Pointer arithmetic



Question


0 1 2 3 4 5 6 7 16 bits
0xF000
0xFE00
uint18_t *aptr;
data
aptr++;
free(aptr);
0xFE00
heap

Q & A