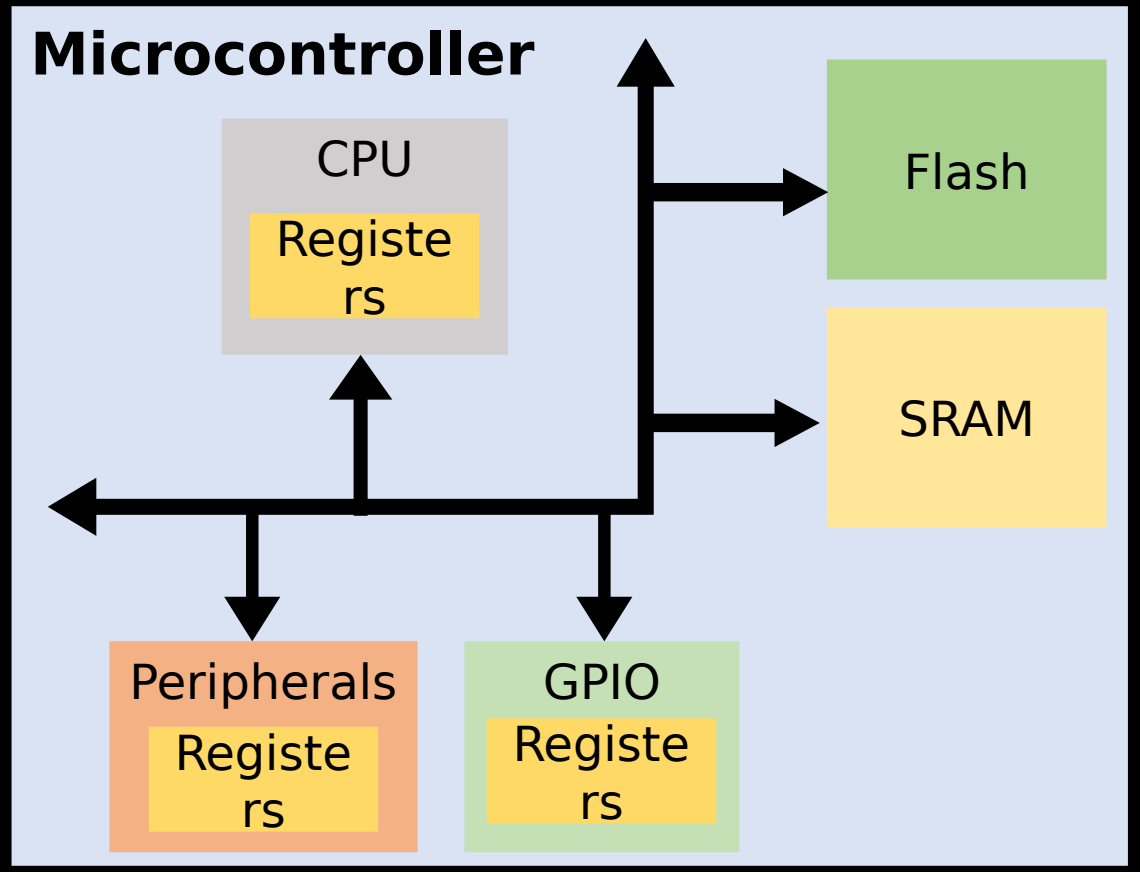
Endianness

Embedded Software Essentials

C2M1V6

Memory [S1]

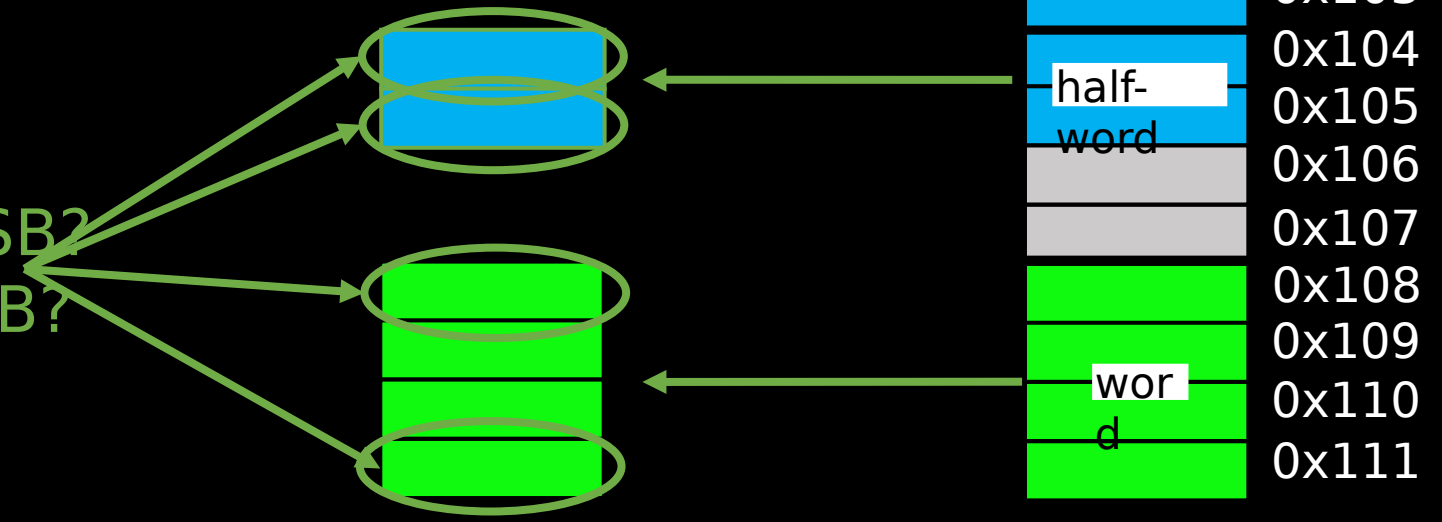
- Memory storage interacting with the CPU
  - Code Memory
  - Data Memory
  - Registers (Peripherals)
- Memory interfaces to CPU through Busses
- Load-Store architecture requires operations to occur in CPU
  - Data get loaded into CPU
  - Data is operated on
  - Data is stored back to memory



Data Order [S2]

- Each address stores 1 Byte
- Half-Words store 2 Bytes
- Words store 4 Bytes

Which location stores the MSB?
Which location stores the LSB?

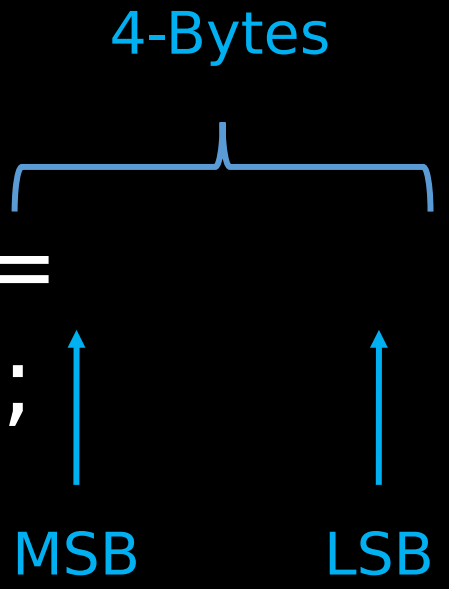


Endianness [S3]

Endianness = How data is represented → **Byte-order** in memory!

Can express data order in two different ways

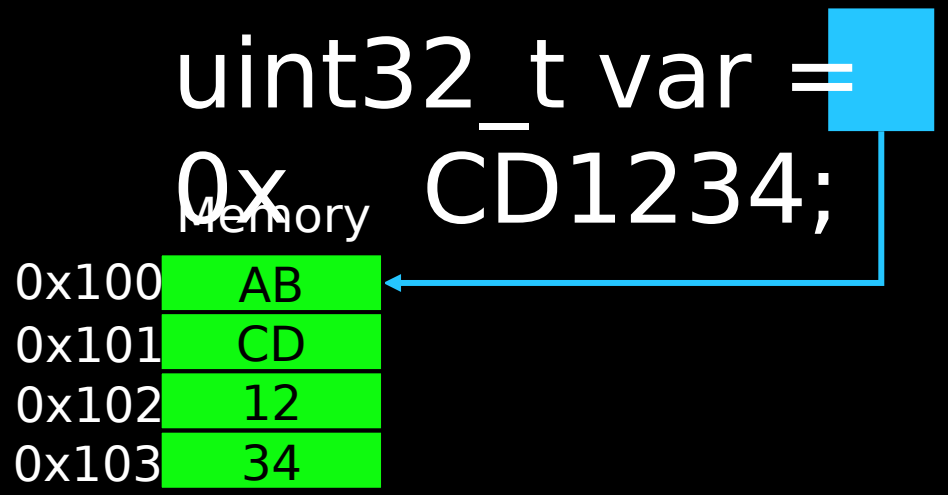
- Little Endian
- Big Endian



Endianness only relevant for **multi-byte data!**

Endianness [S4]

Endianness = How data is represented → **Byte-order** in memory!



Big Endian:
Store ***MSB*** at ***smallest address***

Endianness [S5]

Endianness = How data is represented → **Byte-order** in memory!

Big Endian:
Store ***MSB*** at ***smallest address***

Little Endian:
Store ***LSB*** at ***smallest address***

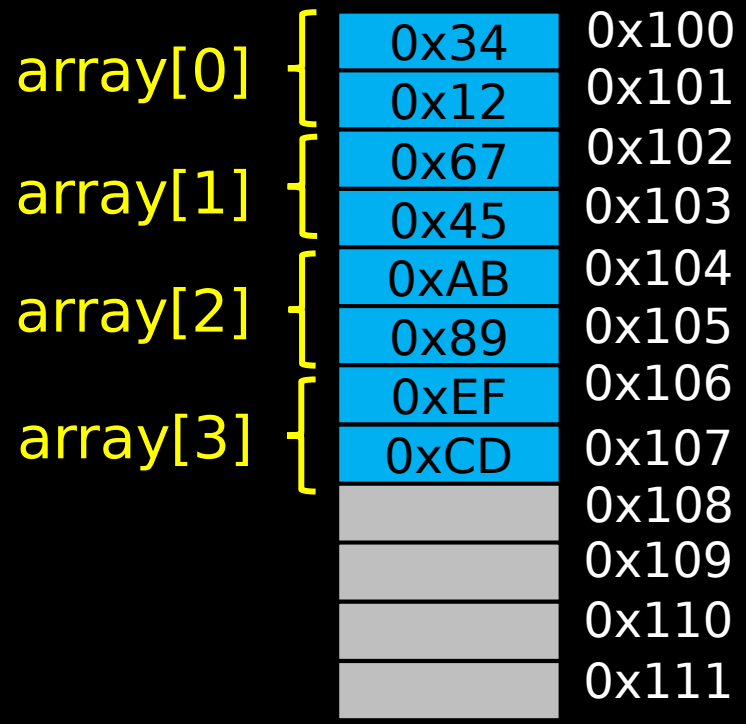
Types and Endianness [S6]

- Endianness does not affect order of elements Arrays or Structures

2-Bytes

```
uint16_t array[4] = { 0x1234,
                      0x4567,
                      0x89AB,
                      0xCDEF };
```

Little Endian Memory



Endianness Configuration [S7]

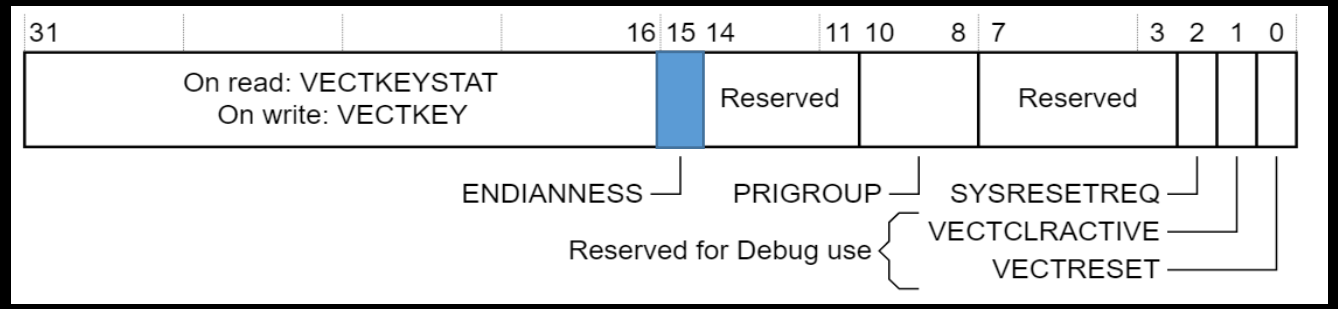
- Endianness is Configurable on many modern platforms
- ARM Cortex-M allows for configuration of Data Memory Endianness
- Code Memory is set to little endian

Endianness Configuration [S8]

- Application Interrupt and Reset Control Register (AIRCR)
  - Allows for reconfiguration of Data Memory Endianness

- Bit 15 of AIRCR Register
  - 0 = Little Endian
  - 1 = Big Endian

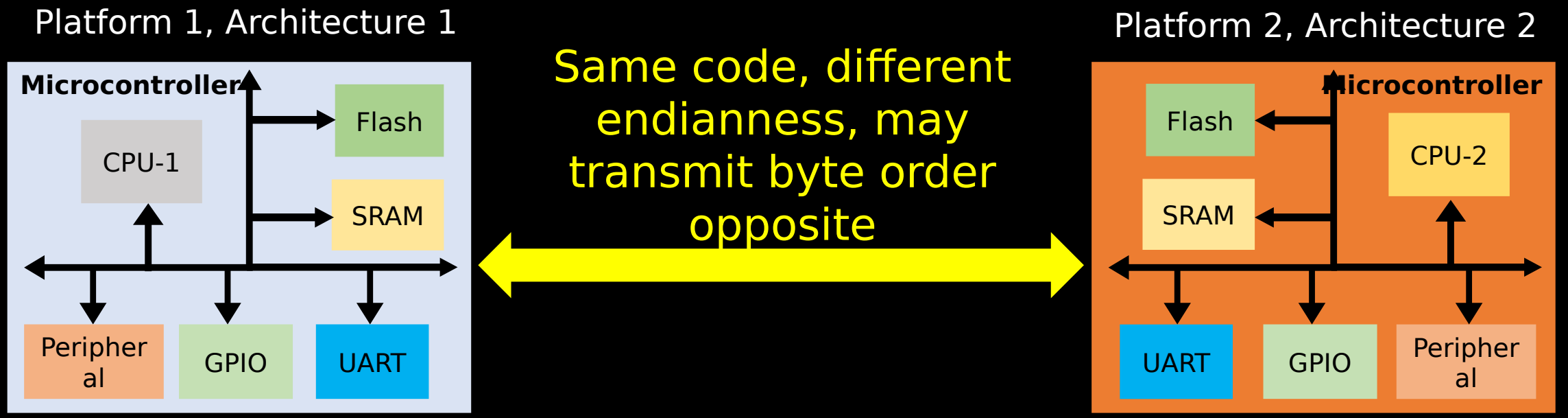
AIRCR Register Bit Fields



Changing Endianness **requires a reset**

Endianness Trouble [S9]

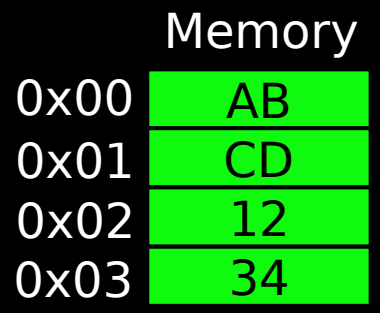
- Endianness must be accounted for
  - Supporting Multiple platforms/Architectures with the same code base
  - Connecting multiple systems together

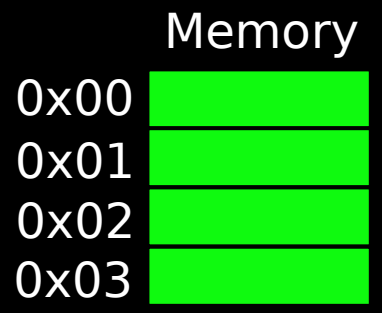


Byte-Swapping [S10a]

```c
/* Switches endianness of variable pointed by ptr */
void byte_swap32(uint32_t * ptr){
   uint8_t i, temp_byte;

   for (i = 0; i < 2; i++){
      temp_byte = *((uint8_t*)ptr + (3-i));
      *((uint8_t*)ptr + (3-i)) = *((uint8_t*)ptr +
i)
      *((uint8_t*)ptr + i) = temp_byte;
   }
}
```

```c
/* Assume little endian */
void main(){
   uint32_t var =
0xABCD1234;
   uint32_t * ptr = &var;

   byte_swap32(ptr);

   while (1);

}
```

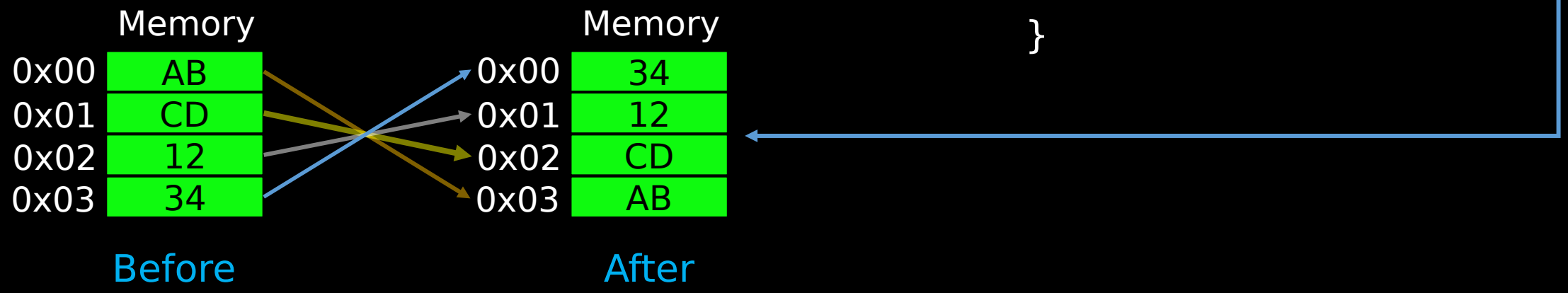




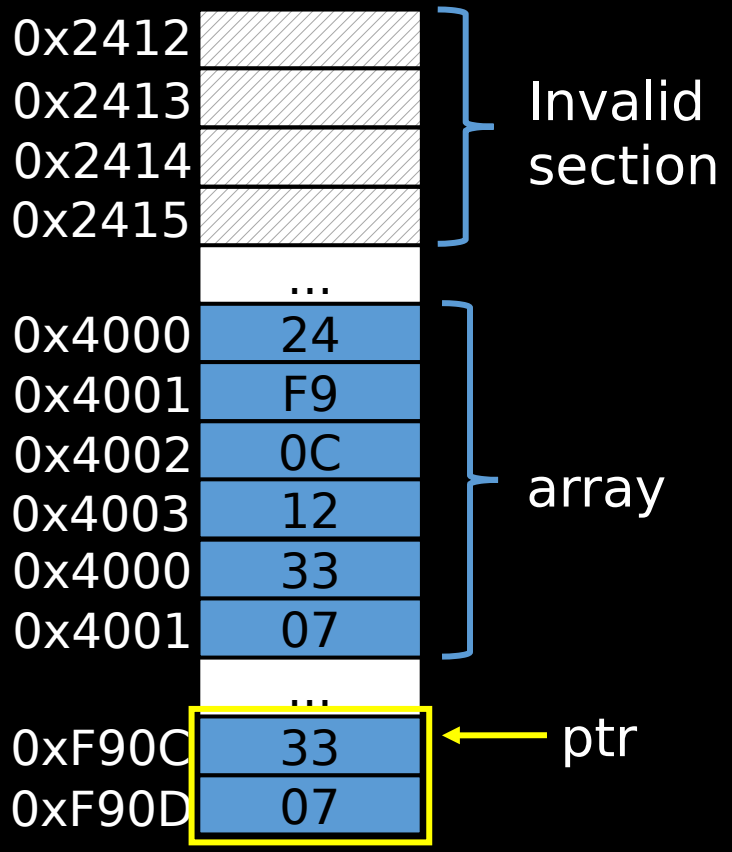
Byte-Swapping [S10b]

```c
/* Switches endianness of variable pointed by ptr */
void byte_swap32(uint32_t * ptr){
   uint8_t i, temp_byte;

   for (i = 0; i < 2; i++){
      temp_byte = *((uint8_t*)ptr + (3-i));
      *((uint8_t*)ptr + (3-i)) = *((uint8_t*)ptr +
i)
      *((uint8_t*)ptr + i) = temp_byte;
   }
}
```

```c
/* Assume little endian */
void main(){
   uint32_t var =
0xABCD1234;
   uint32_t * ptr = &var;

   byte_swap32(ptr);

   while (1);

}
```



Portability with Endianness [S9a]

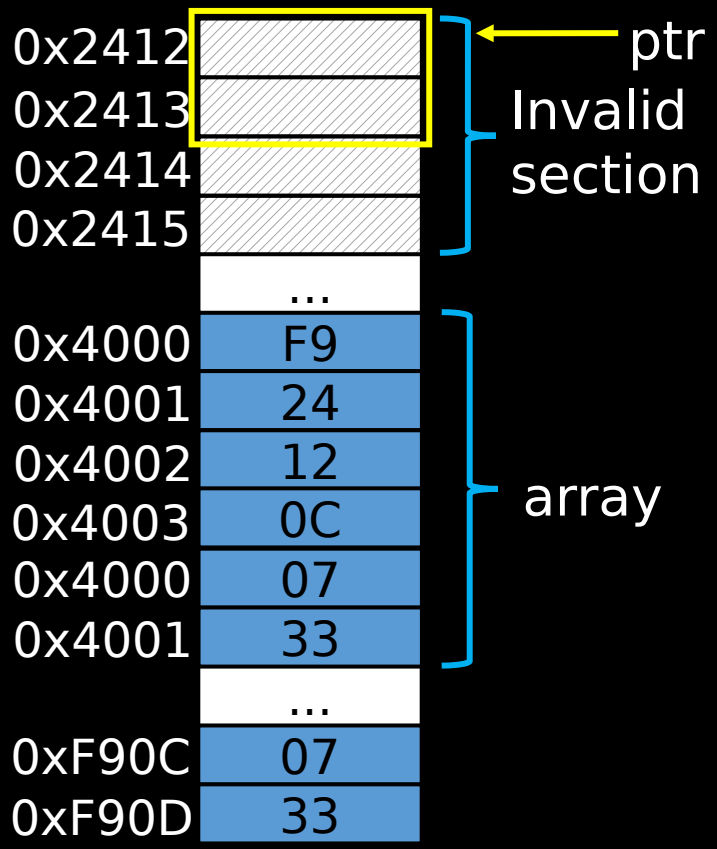



```c
/* Program depends on LE architecture. */
void main(){
   uint16_t array[] = {0xF924, 0x120C, 0};
   uint16_t * ptr = (uint16_t*) ((array[0] & 0xFF00) | (array[1] &
0xFF));

   array[2] = *ptr;

   // Rest of code...
}
```

Dereference at valid address

n little endian architecture...

Portability with Endianness [S9b]




```c
/* Program depends on LE architecture. */
void main(){
   uint16_t array[] = {0xF924, 0x120C, 0};
   uint16_t * ptr = (uint32_t) ((array[0] & 0xFF00) | (array[1] &
0xFF));

   array[2] = *ptr;

   // Rest of code...
}
```

Dereferenced at **invalid** address!

In big endian architecture...