Mapování paměti ARM-Cortex M

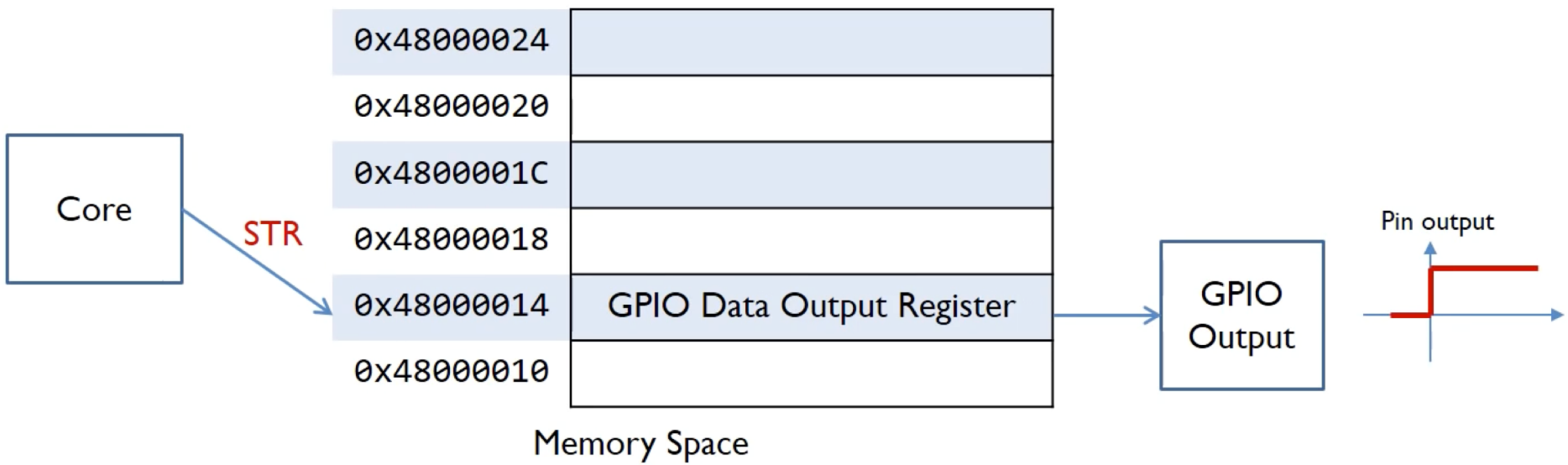
Převzato a upraveno od: web.eece.maine.edu/~zhu/book/

Interfacing Peripherals

- **Port-mapped I/O**
  - Use special CPU instructions: `Special_instruction Reg, Port`
- **Memory-mapped I/O**
  - A simpler and more convenient way to interface I/O devices
  - Each device registers is assigned to a memory address in the address space of the microprocessor
  - Use native CPU load/store instructions: `LDR/STR Reg, [Reg, #imm]`

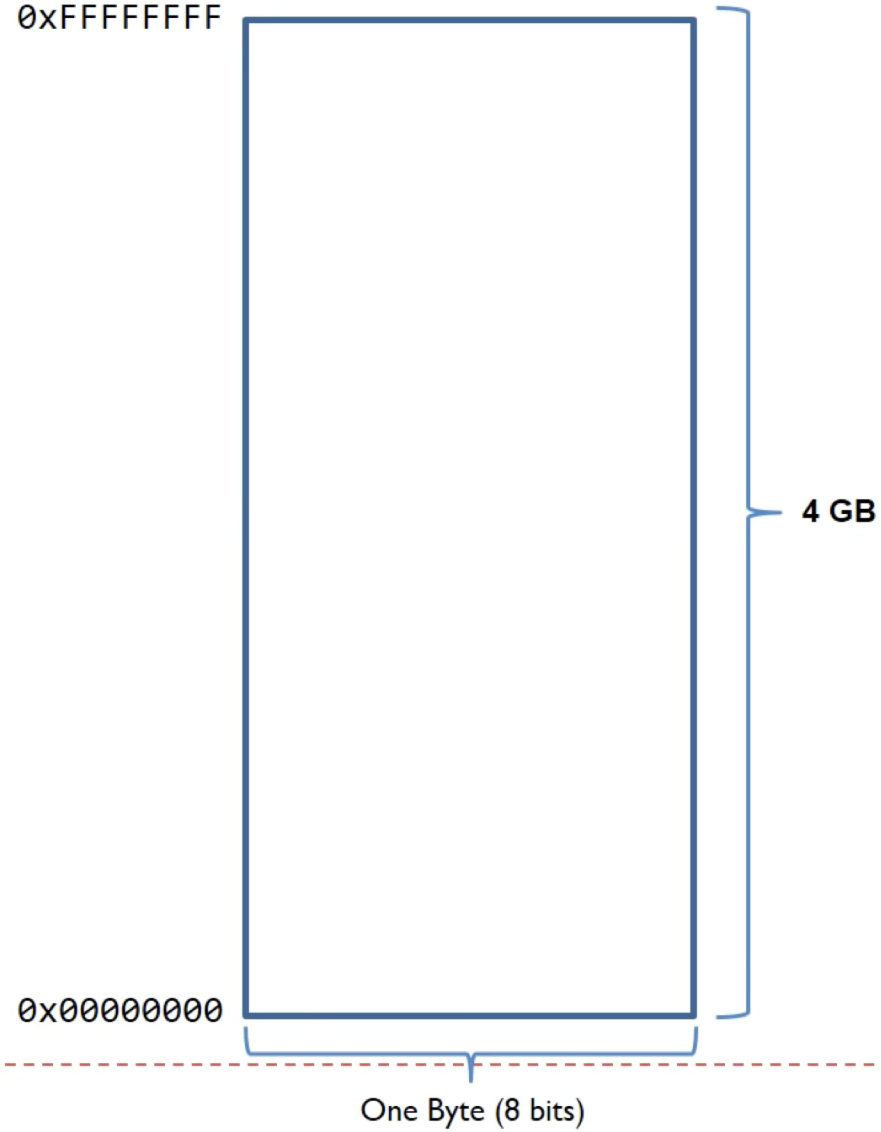
Interfacing Peripherals

- **Port-mapped I/O**
  - Use special CPU instructions: `Special_instruction Reg, Port`
- **Memory-mapped I/O**
  - A simpler and more convenient way to interface I/O devices
  - Each device registers is assigned to a memory address in the address space of the microprocessor
  - Use native CPU load/store instructions: `LDR/STR Reg, [Reg, #imm]`

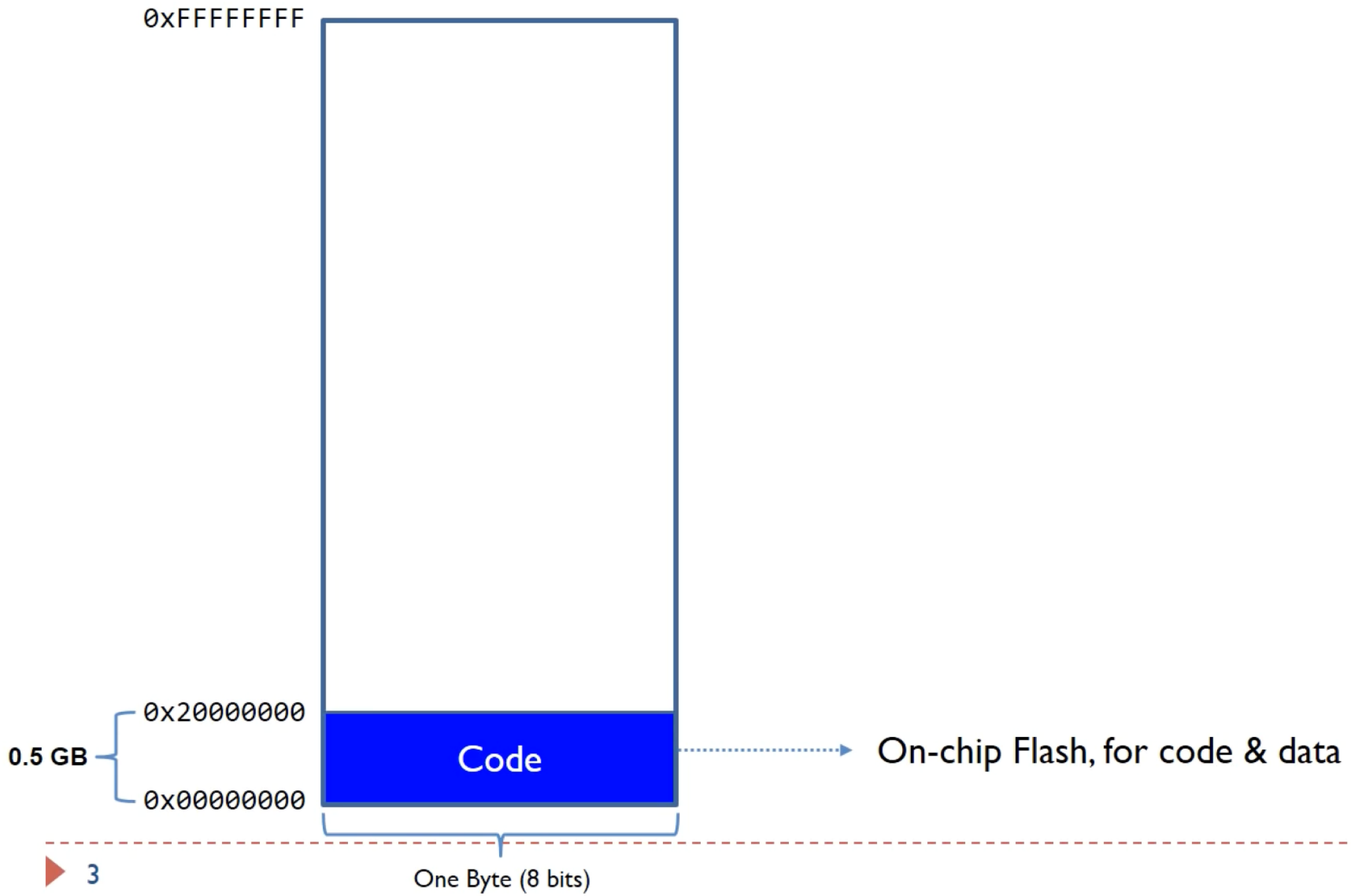


ARM Cortex-M microprocessors use memory-mapped I/O.

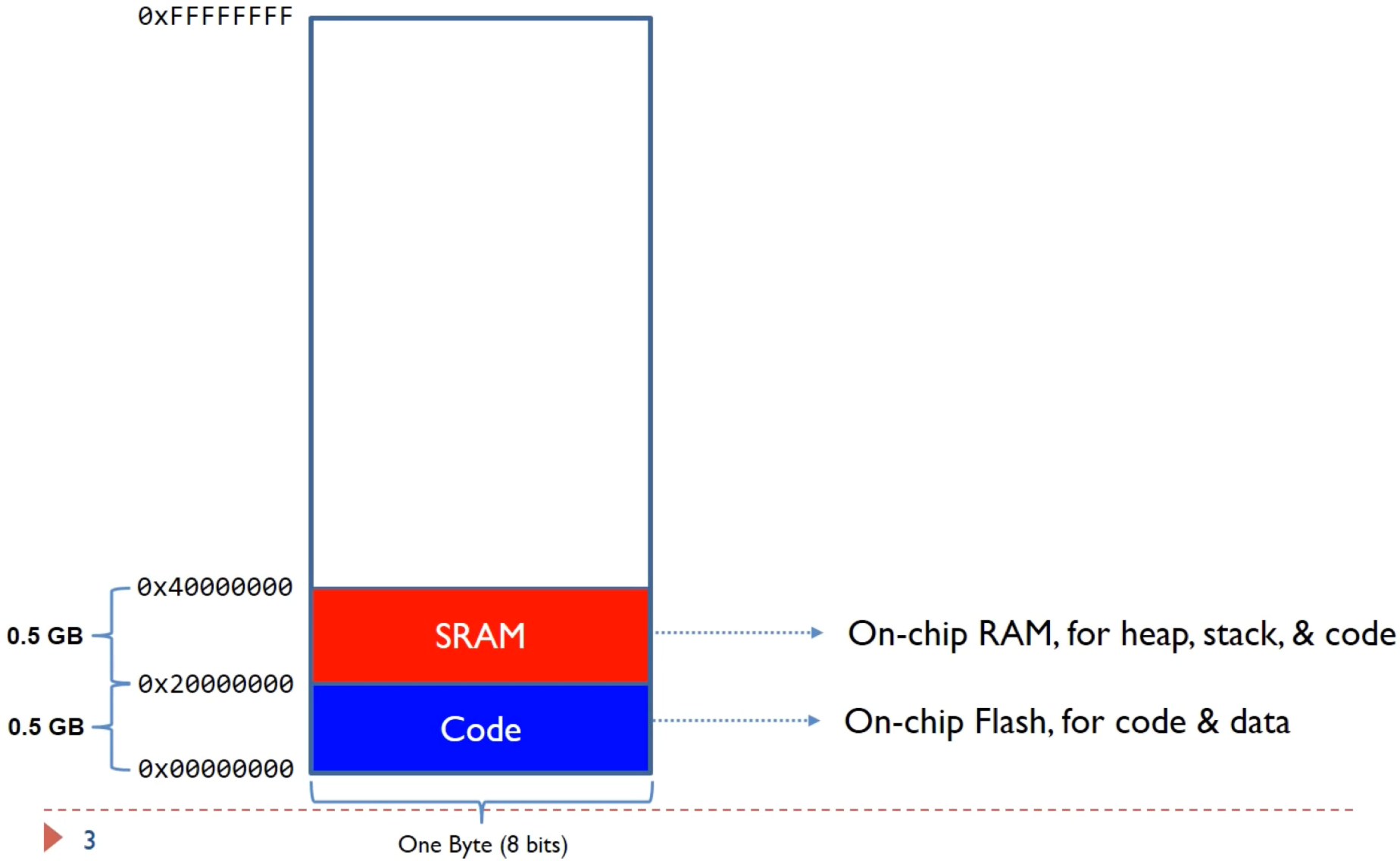
Memory Map of STM32L4



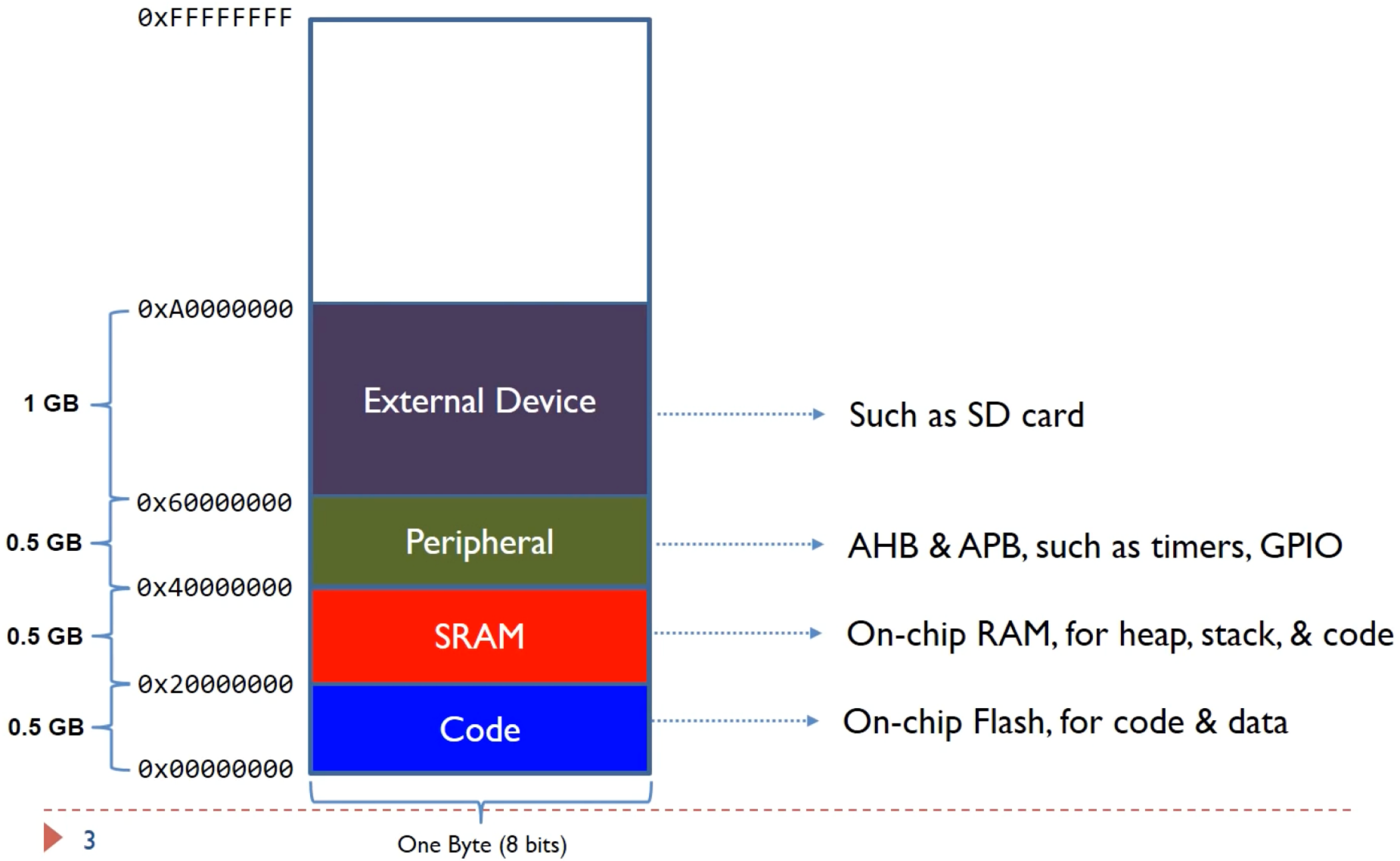
Memory Map of STM32L4



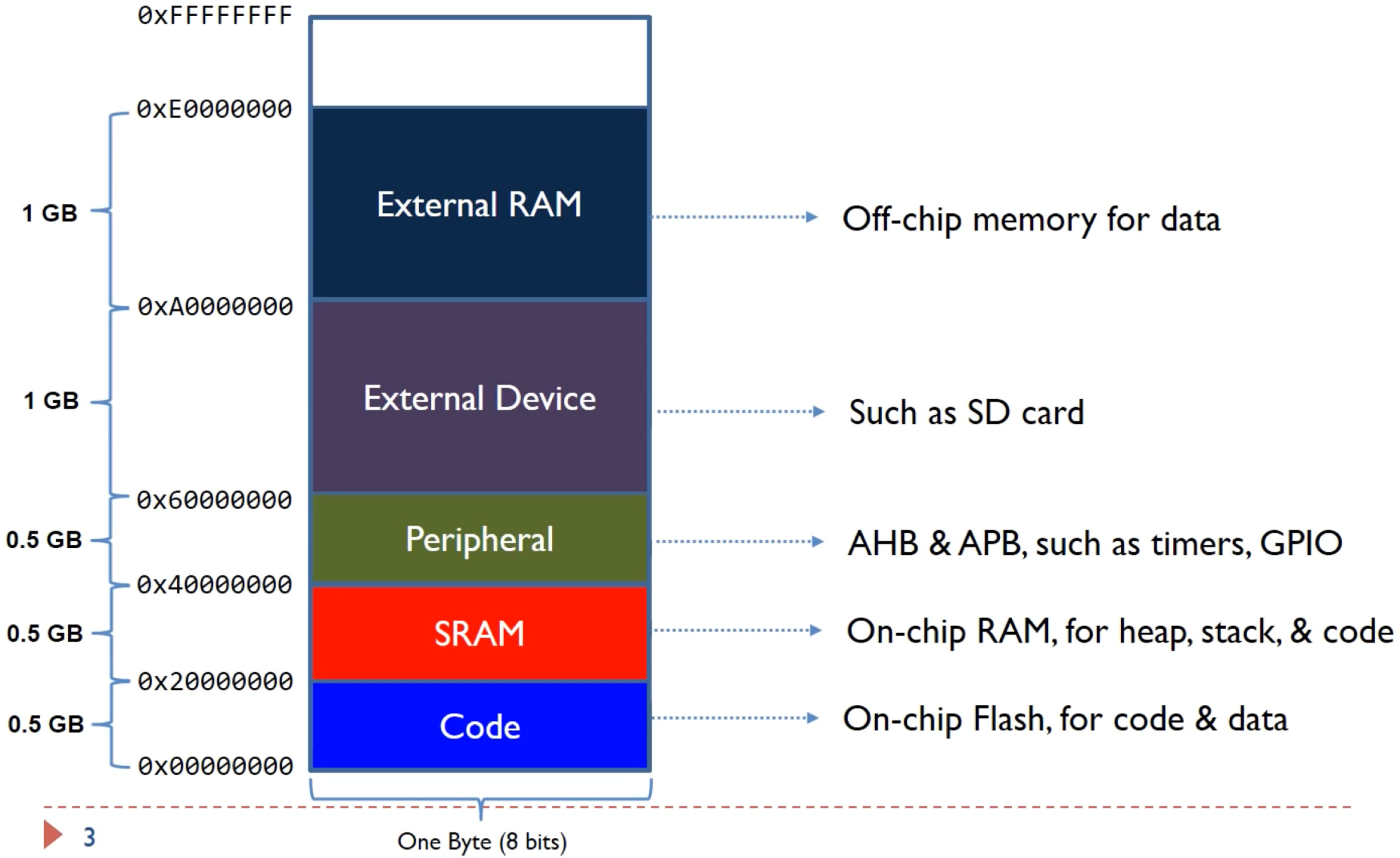
Memory Map of STM32L4



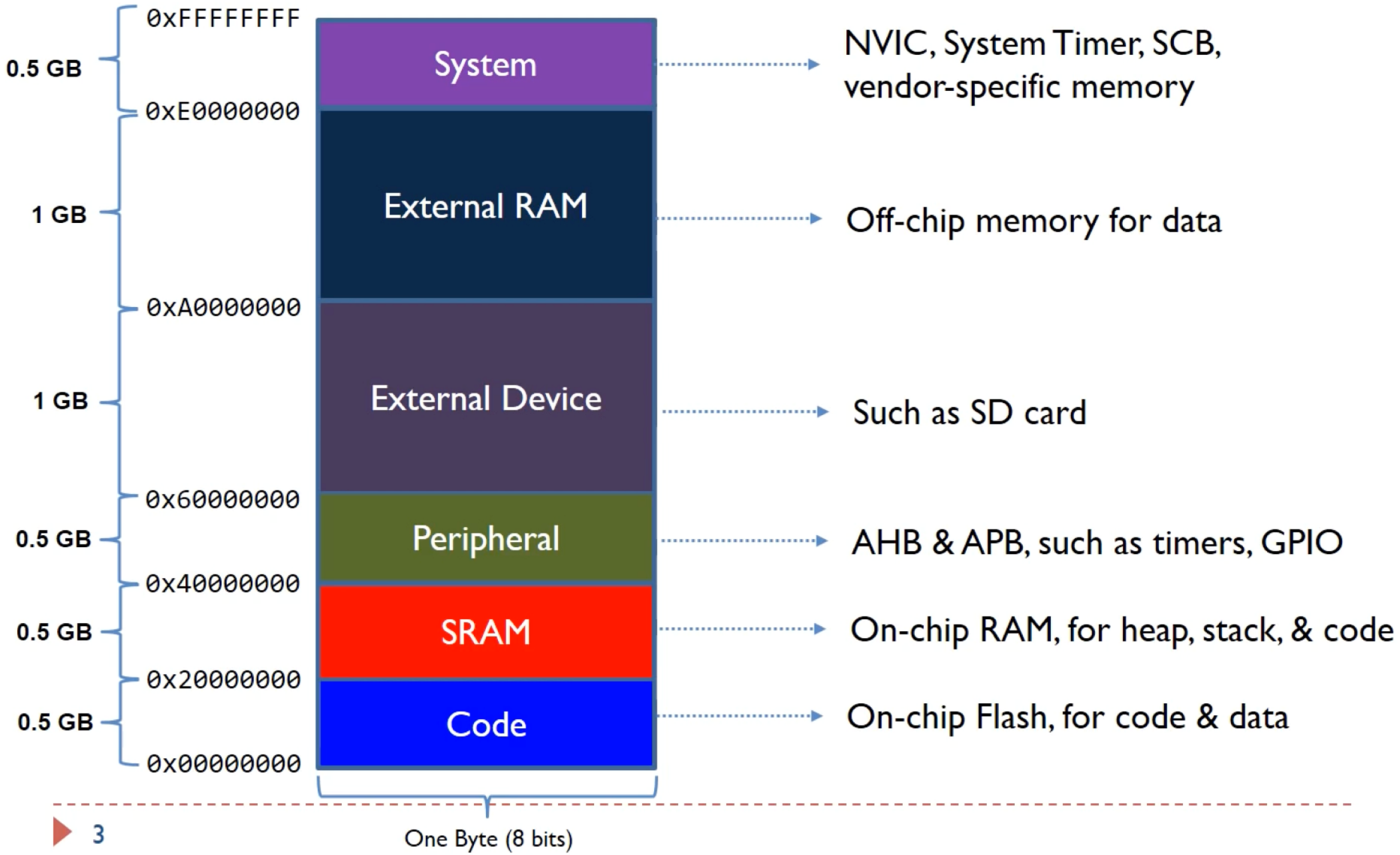
Memory Map of STM32L4



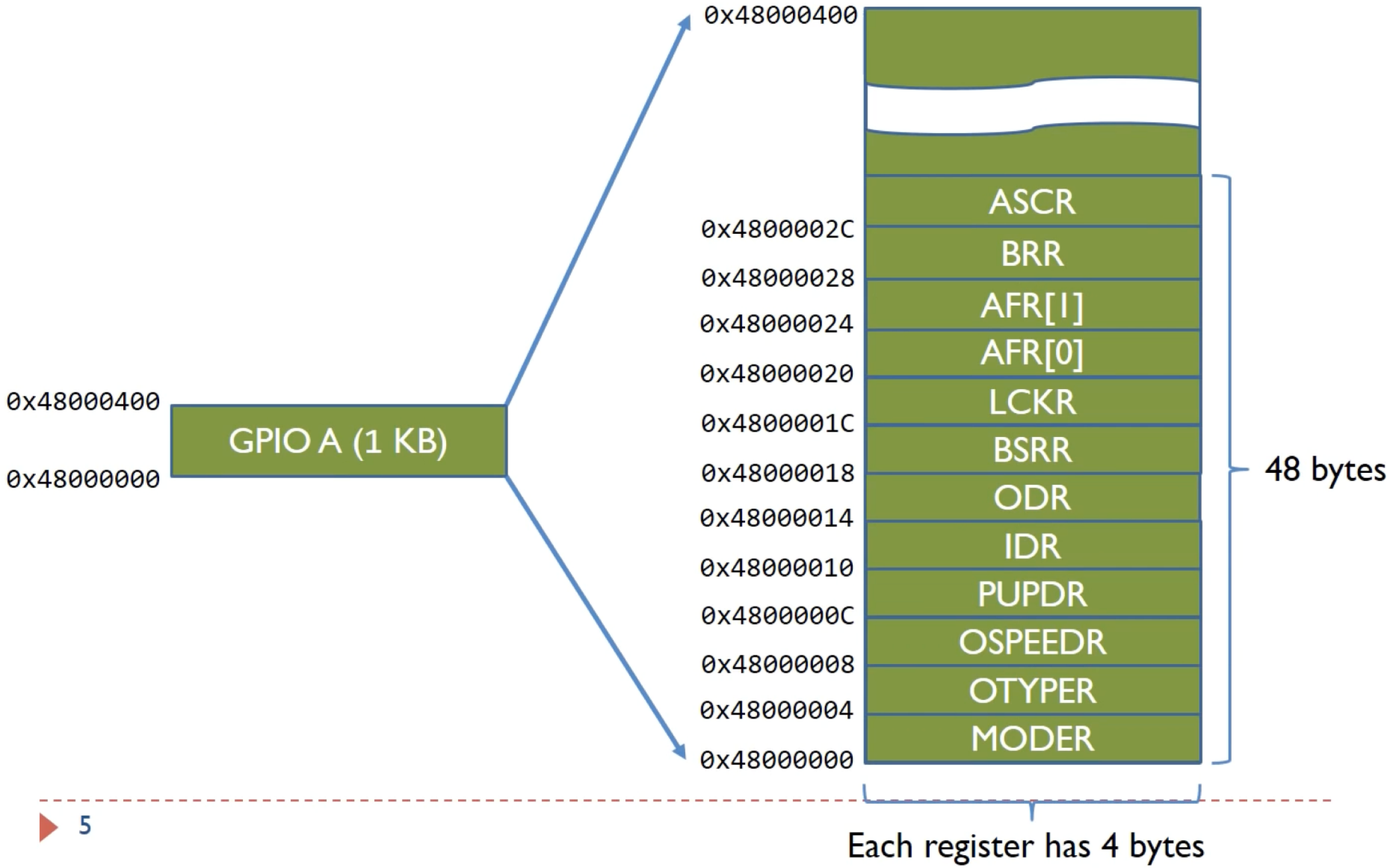
Memory Map of STM32L4



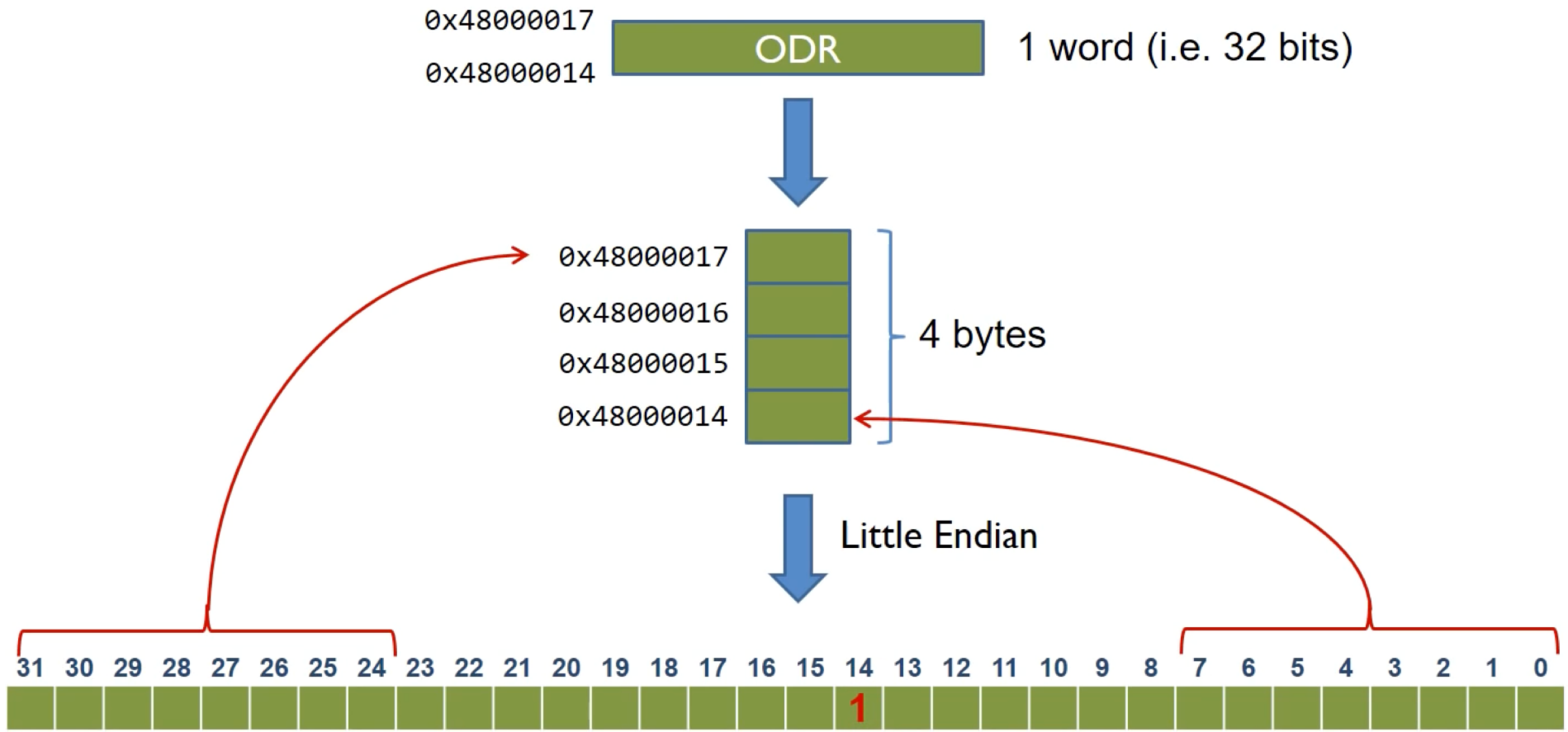
Memory Map of STM32L4



GPIO Memory Map



Output Data Register (ODR)

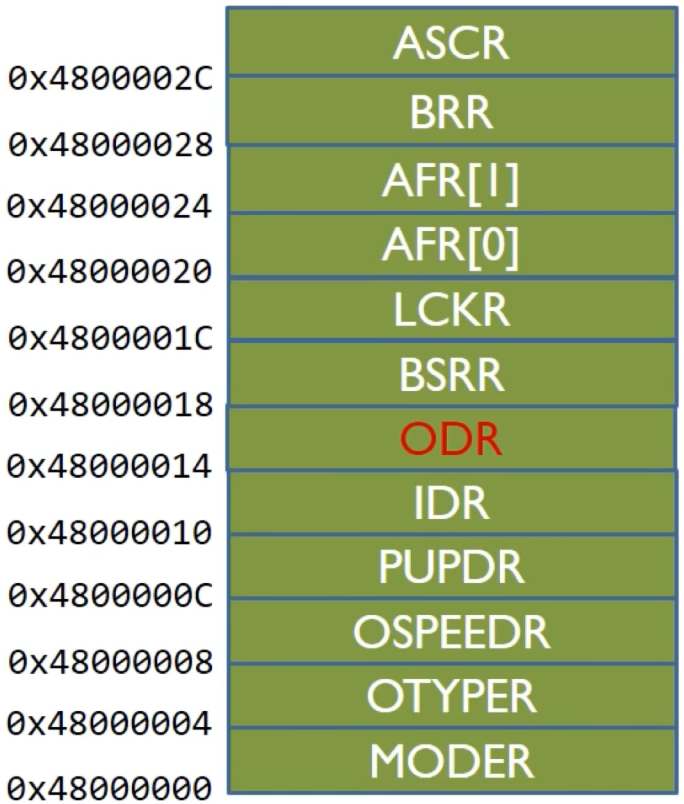



```
*((uint32_t *) 0x48000014) |= 1UL<<14;
```

Dereferencing a pointer

Bitwise OR

Dereferencing a Memory Address




```
typedef struct {
  volatile uint32_t MODER;    // Mode register
  volatile uint32_t OTYPER;   // Output type register
  volatile uint32_t OSPEEDR;  // Output speed register
  volatile uint32_t PUPDR;    // Pull-up/pull-down register
  volatile uint32_t IDR;      // Input data register
  volatile uint32_t ODR;      // Output data register
  volatile uint32_t BSRR;     // Bit set/reset register
  volatile uint32_t LCKR;     // Configuration lock register
  volatile uint32_t AFR[2];   // Alternate function registers
  volatile uint32_t BRR;      // Bit Reset register
  volatile uint32_t ASCR;     // Analog switch control register
} GPIO_TypeDef;

// Casting memory address to a pointer
#define GPIOA ((GPIO_TypeDef *) 0x48000000)
```

```
GPIOA->ODR |= 1UL<<14;
```

or

```
(*GPIOA).ODR |= 1UL<<14;
```

Programming in Assembly

In C

```
GPIOA->ODR |= 1UL<<14;    // Set bit 14 to 1
```

In Assembly

```
GPIOA_BASE EQU   0x48000000
GPIO_ODR   EQU   0x14

LDR r7, =GPIOA_BASE       ; Load GPIO port A base address
LDR r1, [r7, #GPIO_ODR]   ; r1 = GPIOA->ODR
ORR r1, r1, #(1<<14)      ; Set output of pin 14 to high
STR r1, [r7, #GPIO_ODR]   ; Write the output data register
```

Programming in Assembly

In C

```
GPIOA->ODR |= 1UL<<14;    // Set bit 14 to 1
```

In Assembly

```
GPIOA_BASE EQU   0x48000000
GPIO_ODR   EQU   0x14

LDR r7, =GPIOA_BASE       ; Load GPIO port A base address
LDR r1, [r7, #GPIO_ODR]   ; r1 = GPIOA->ODR
ORR r1, r1, #(1<<14)      ; Set output of pin 14 to high
STR r1, [r7, #GPIO_ODR]   ; Write the output data register
```

KONEC