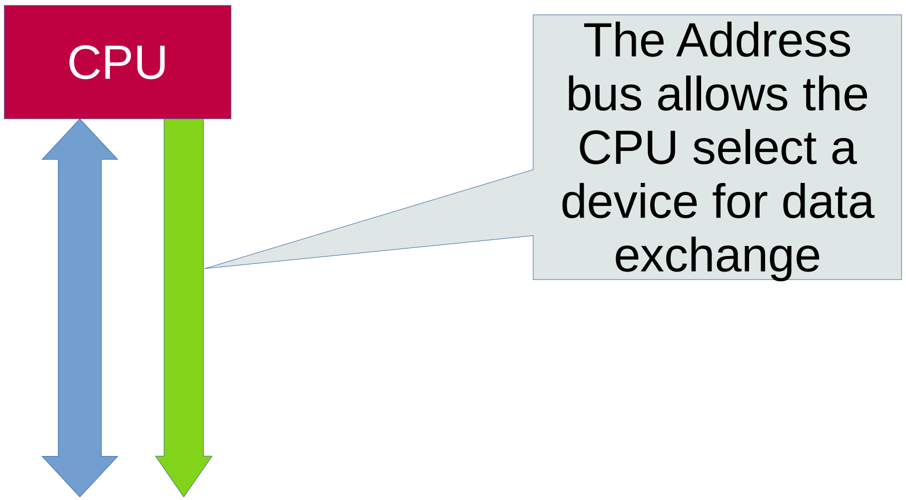
Microcontrollers: Input / Output

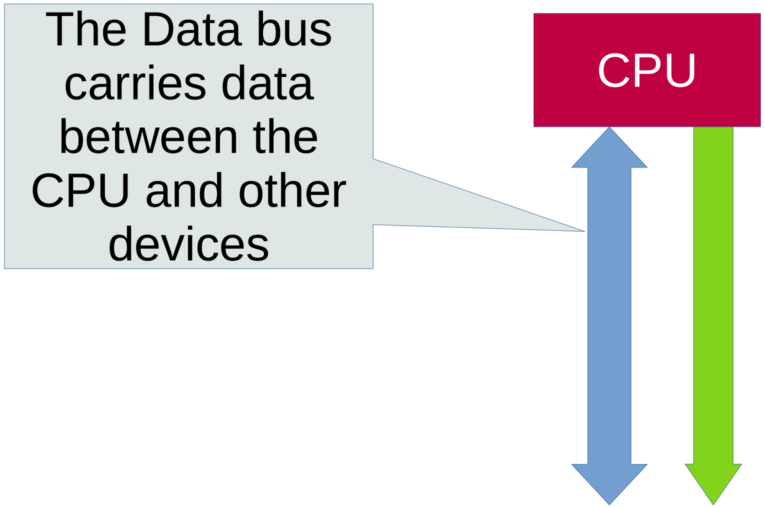
Microcontrollers: Input / Output

CPU

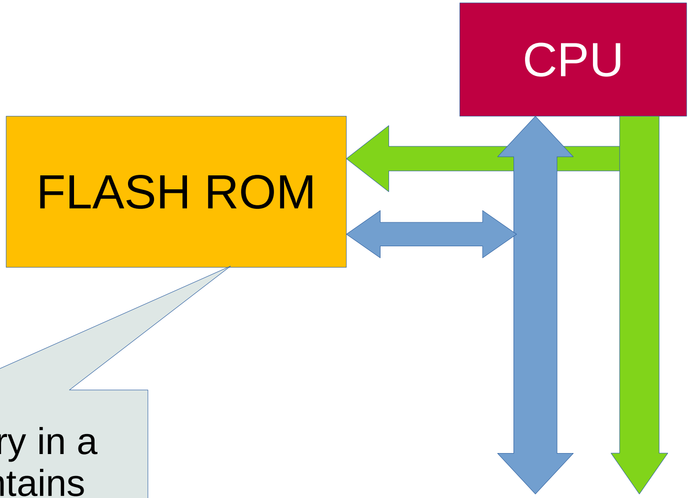
Executes instructions, manipulates data, makes decisions


CPU
The Address bus allows the CPU select a device for data exchange

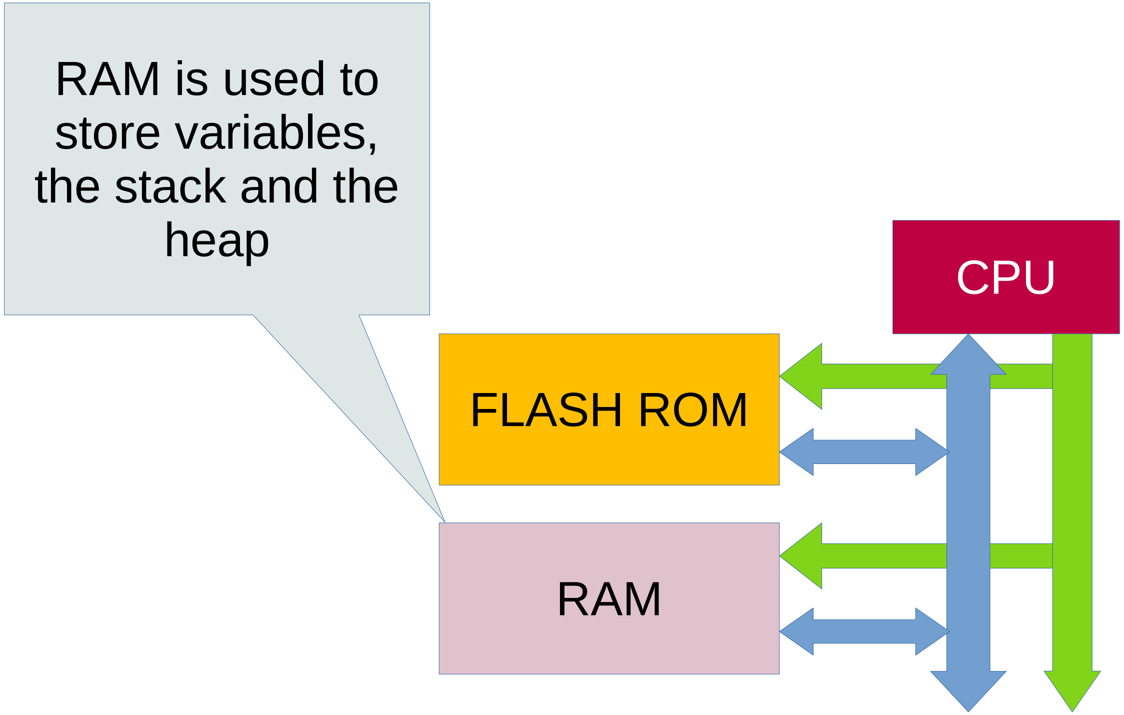
Microcontrollers: Input / Output



Microcontrollers: Input / Output

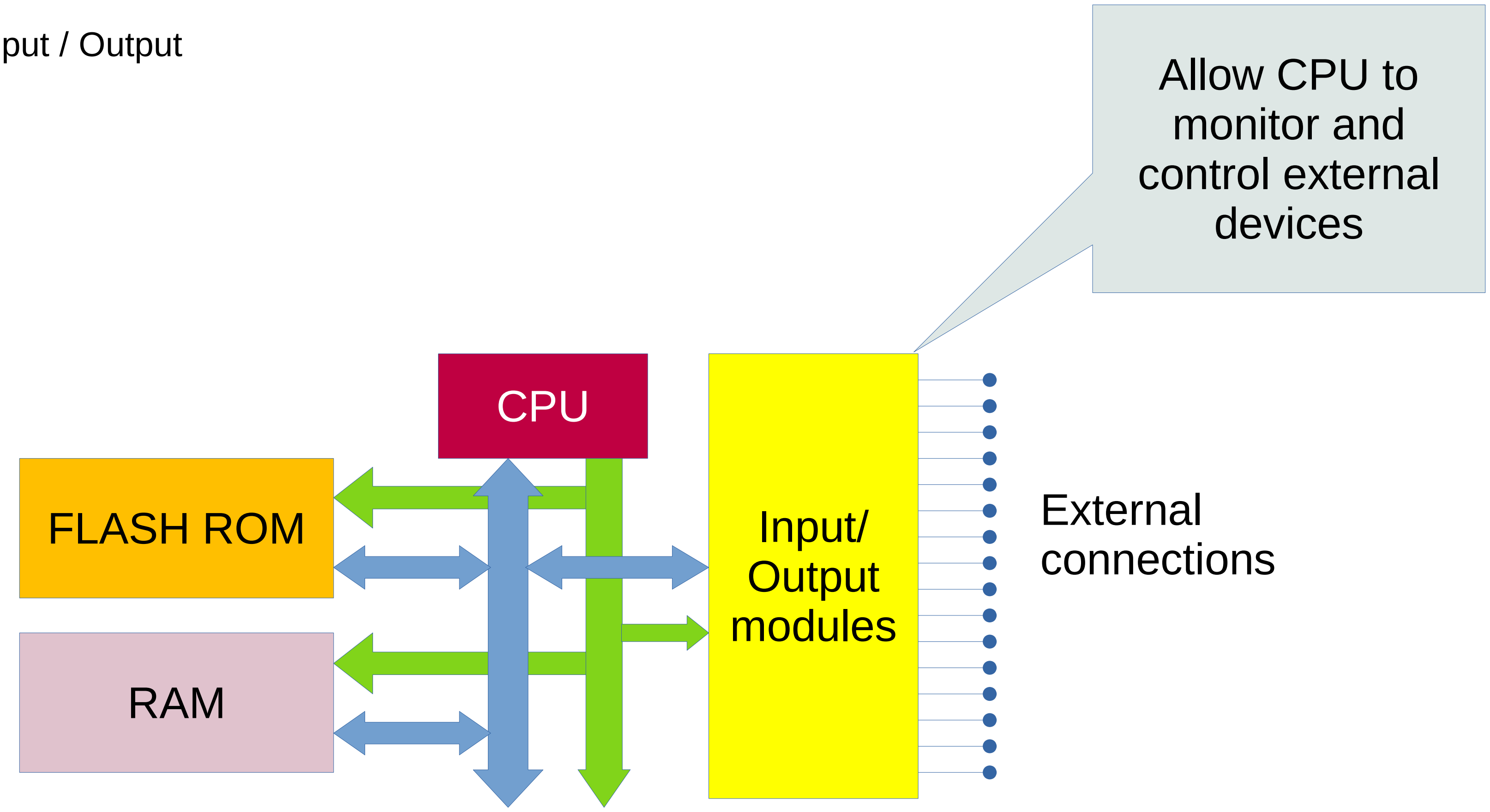


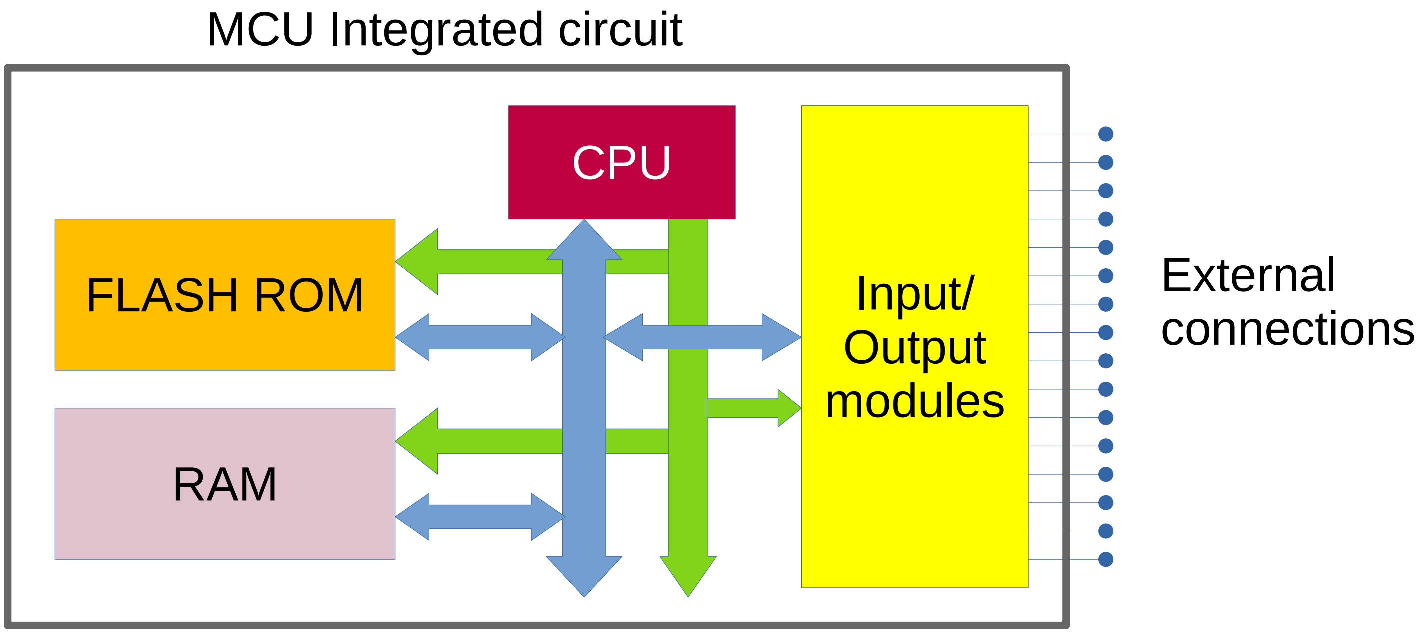
Similar to memory in a USB drive. Contains programs and constant data. Contents are maintained even if there is no power



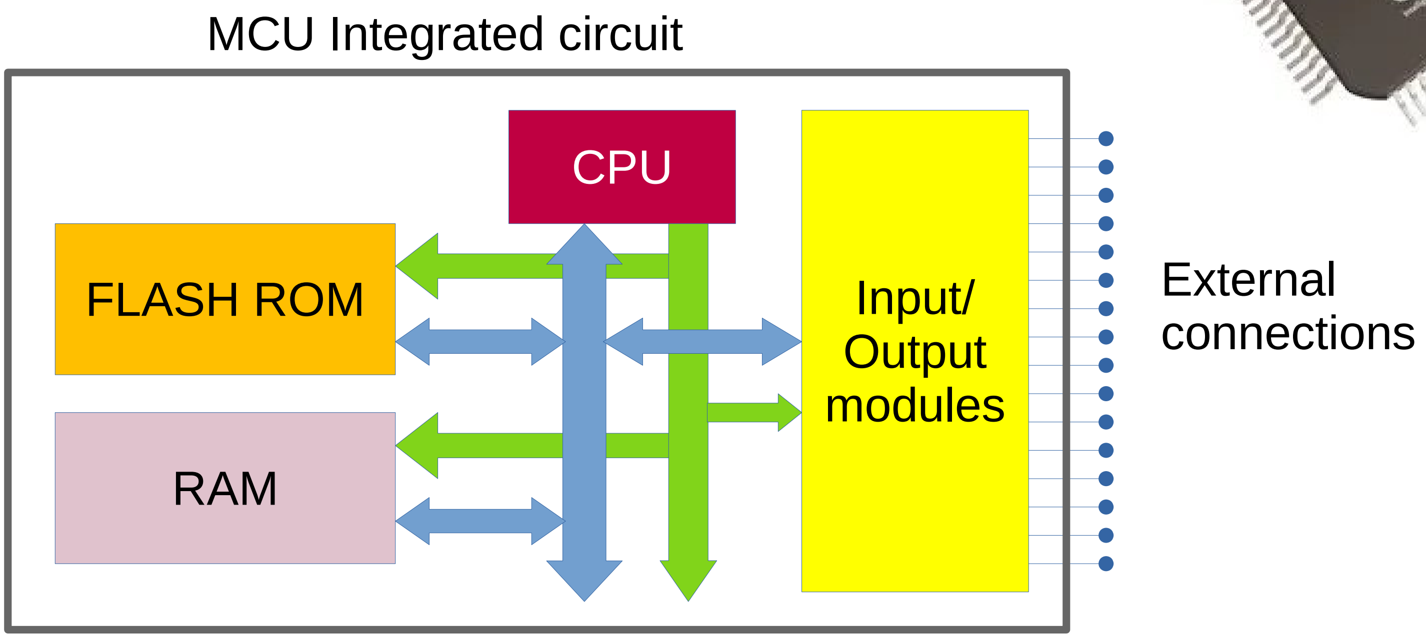
Why can't we use Flash ROM for variables?

Microcontrollers: Input / Output

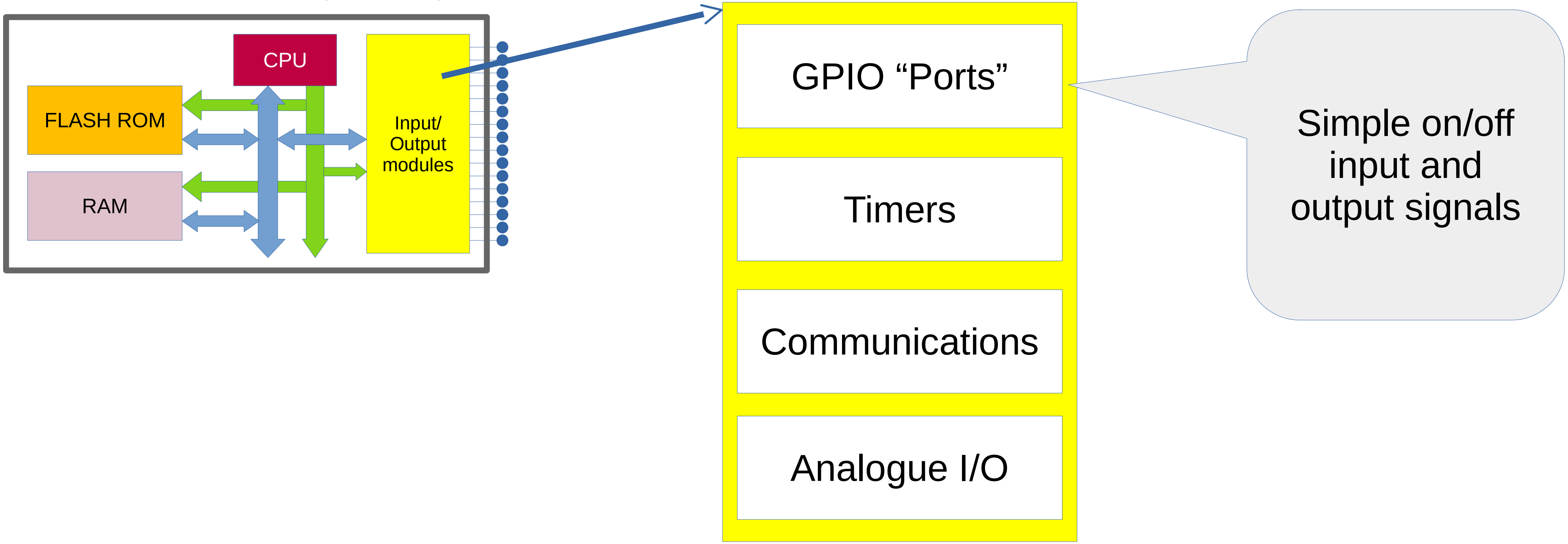



MCU Integrated circuit
CPU
FLASH ROM
RAM
Input/
Output
modules
External
connections

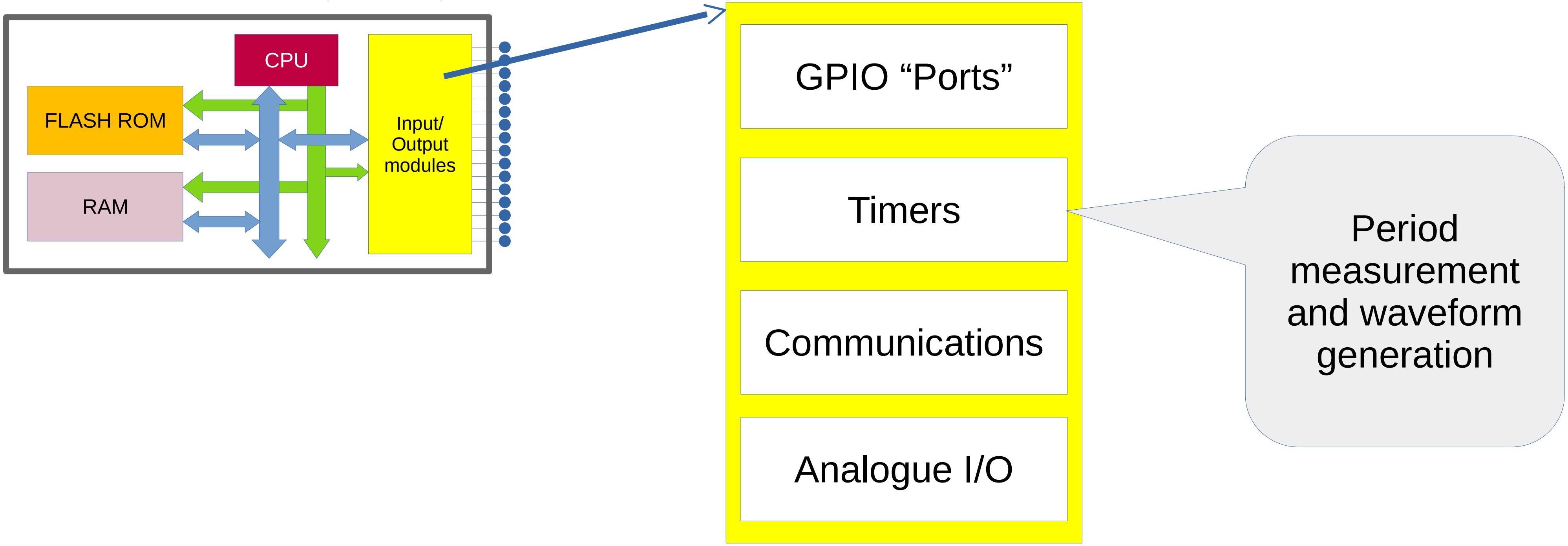
Microcontrollers: Input / Output



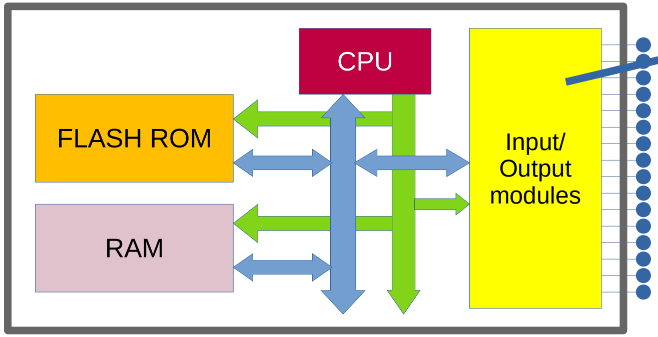
Microcontrollers: Input / Output



Microcontrollers: Input / Output



Microcontrollers: Input / Output



GPIO “Ports”

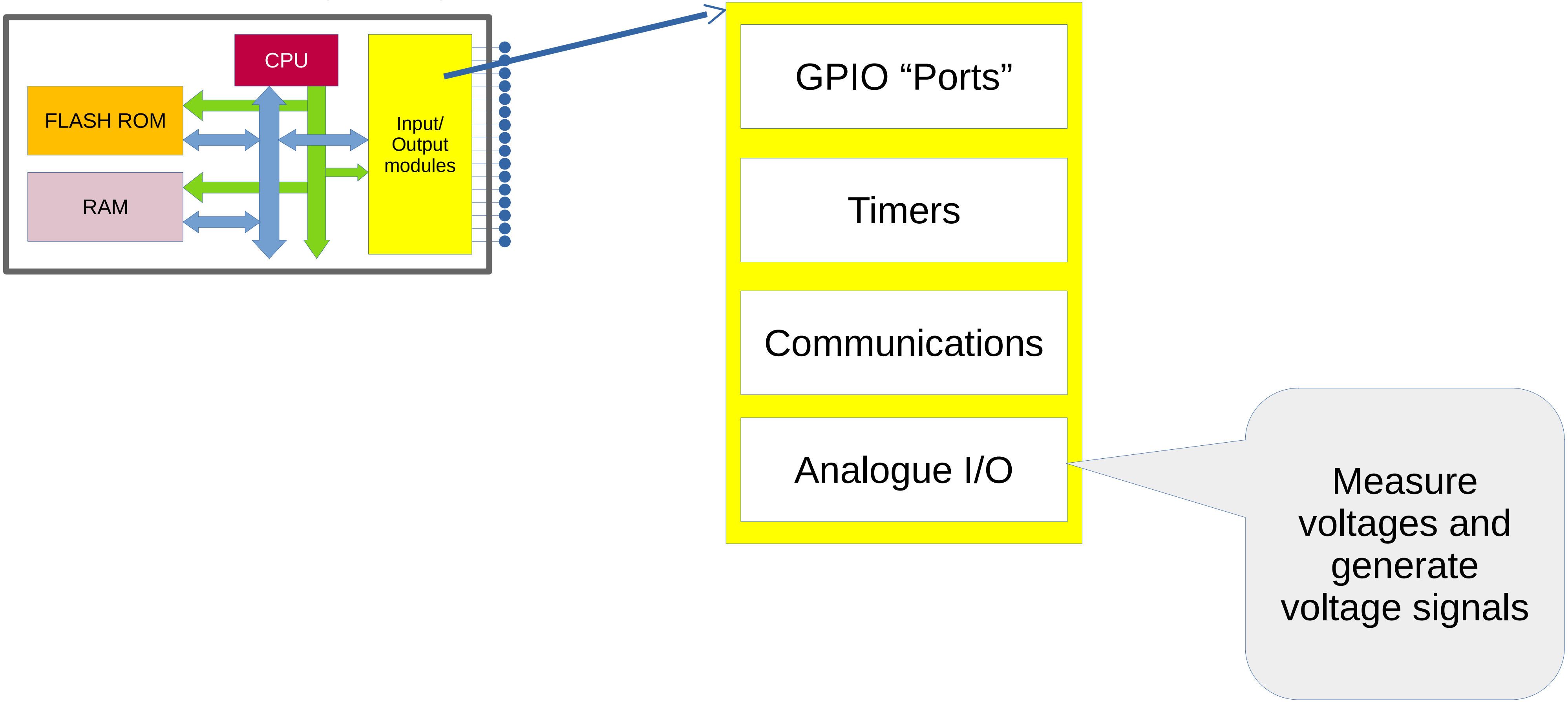
Timers

Communications

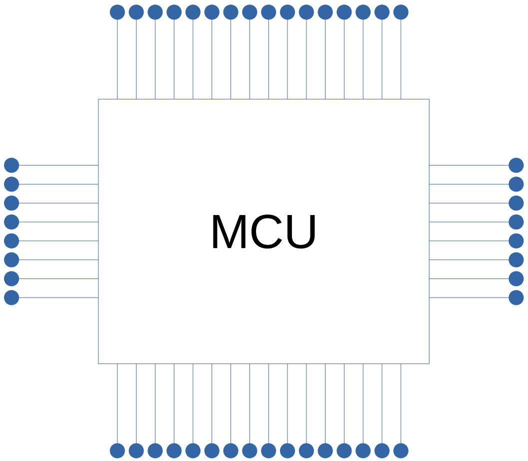
Analogue I/O

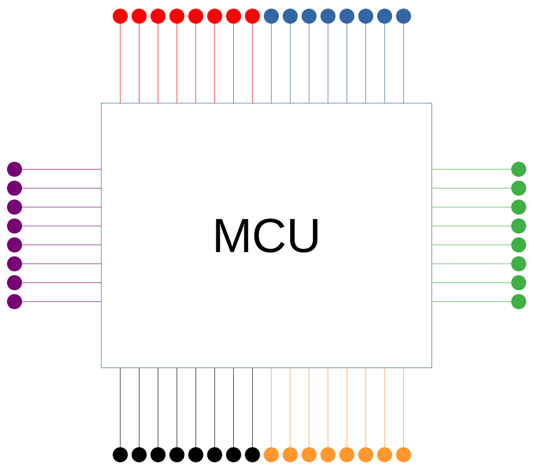
Data transmission and reception to other devices

Microcontrollers: Input / Output



Consider and MCU with 48 input/output pins





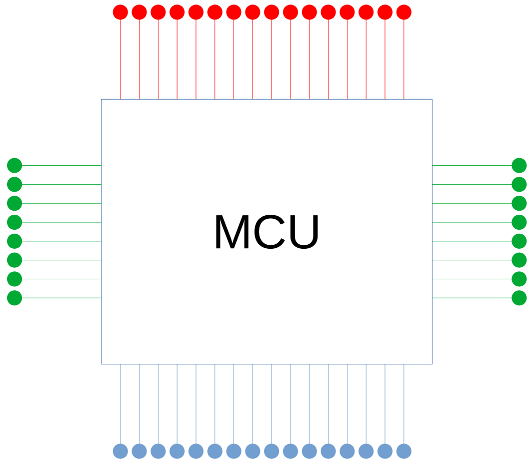
Consider and MCU with 48 input/output pins

Alternatively, configuration data could apply to blocks of I/O pins.

e.g. in groups of 8

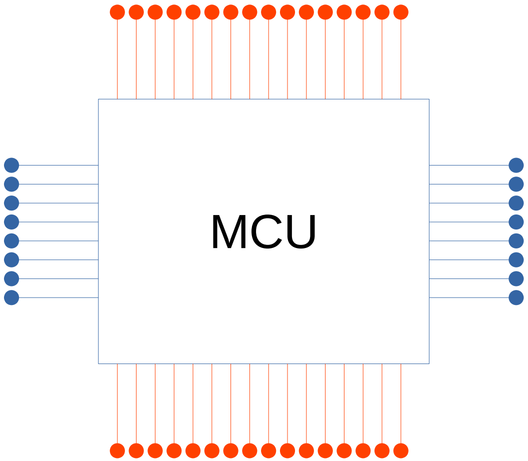
There are 6 such groupings in this case.

Such groupings are commonly called **Ports**



Consider and MCU with 48 input/output pins

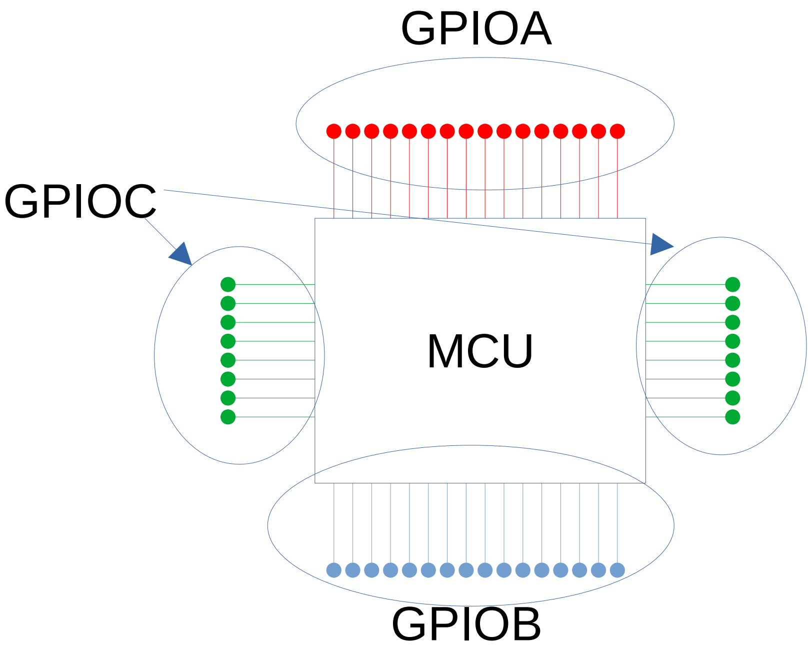
Grouping could be in blocks of 16 in which case we would have 3 ports



Consider and MCU with 48 input/output pins

They could even be grouped in 32 bit blocks giving us 1.5 ports.

The first port would have bits 0 to 31 while the second port may have bits 0 to 15



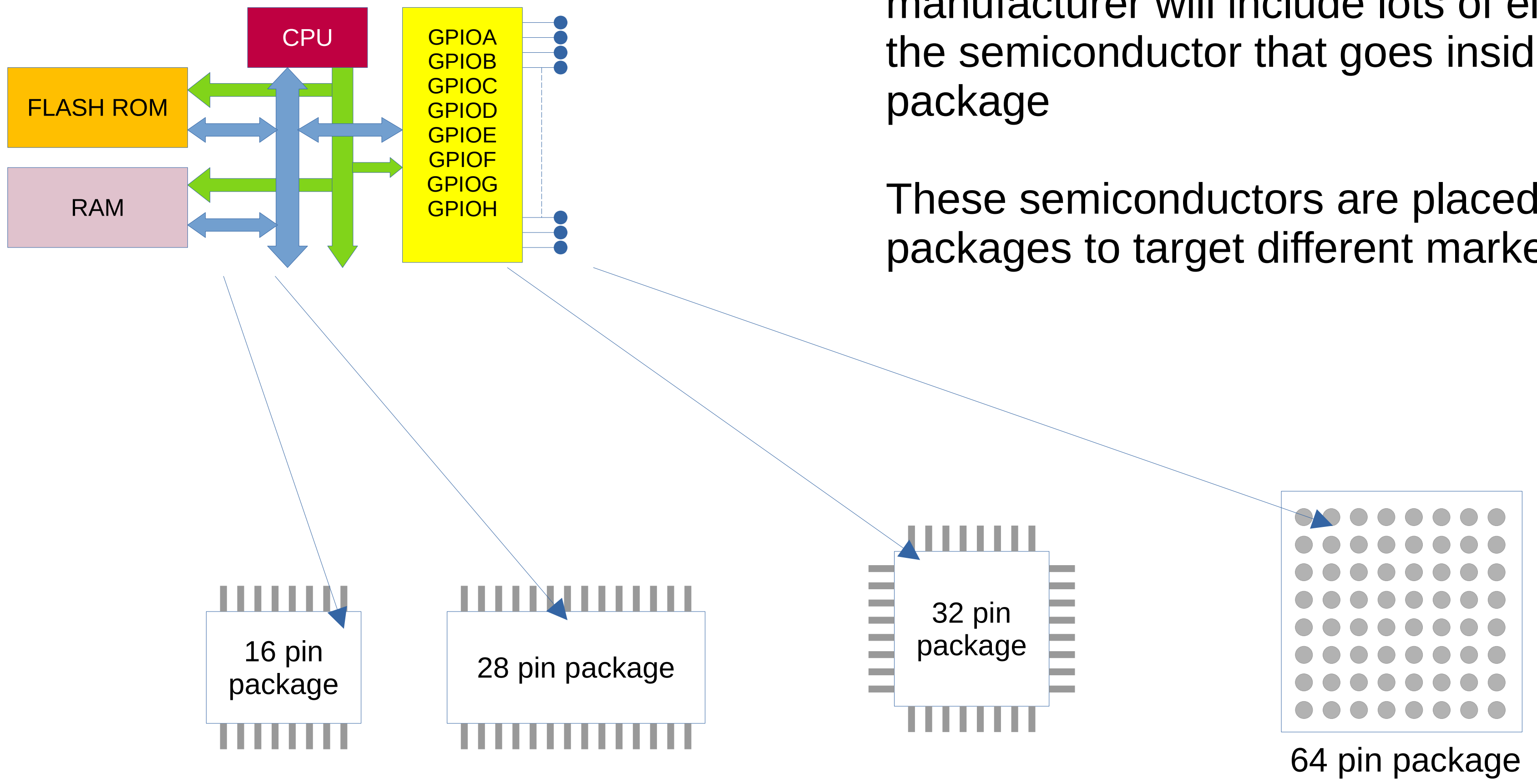
ST Microelectronics typically groups I/O pins in blocks of 16 i.e. we have 16-bit I/O ports.

These ports are called
Port A
Port B
Port C
And so on.

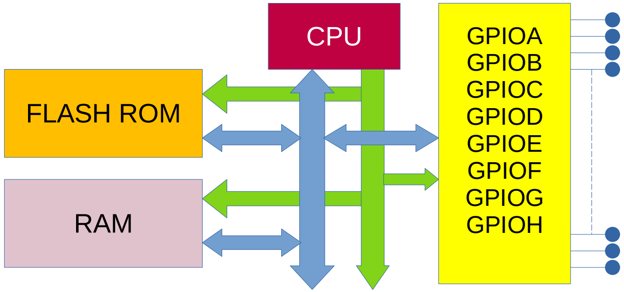
Microcontrollers: Input / Output

The design of an MCU typically consists of a set of library elements. In many cases, the manufacturer will include lots of elements in the semiconductor that goes inside the chip package

These semiconductors are placed in different packages to target different markets

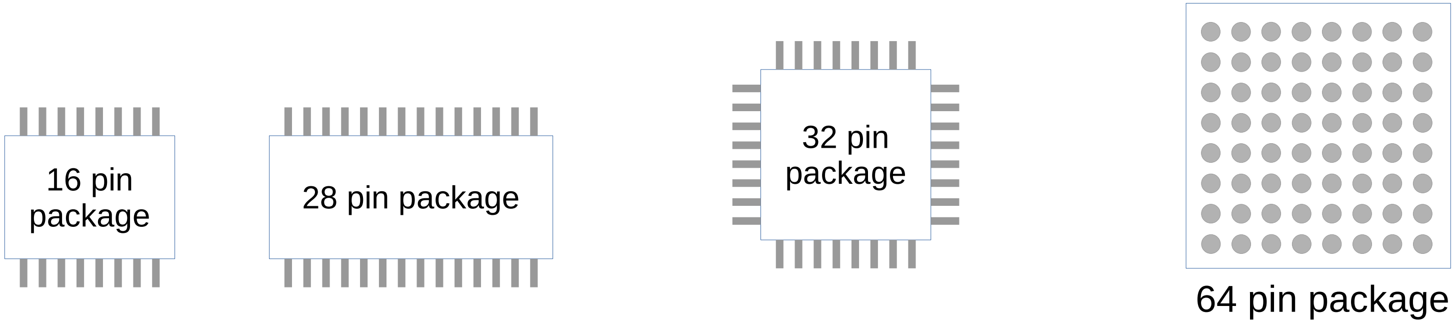


Microcontrollers: Input / Output

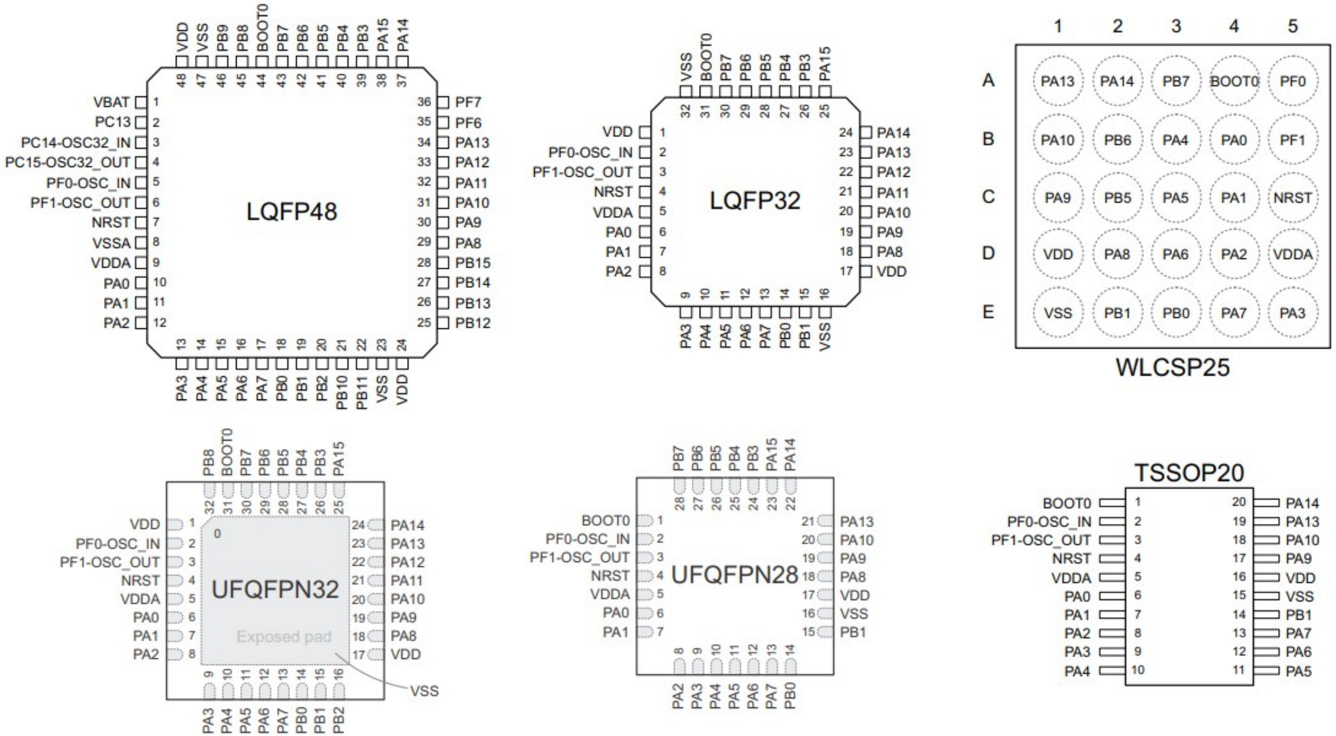


Obviously the 16 pin package can not connect all of the available GPIO pins to the outside world.

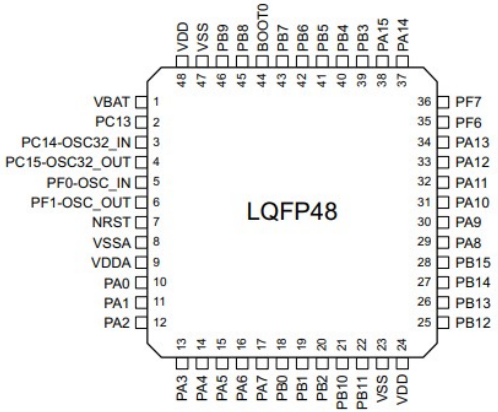
Nevertheless, it is still cheaper for the manufacturer to design MCU’s in this way

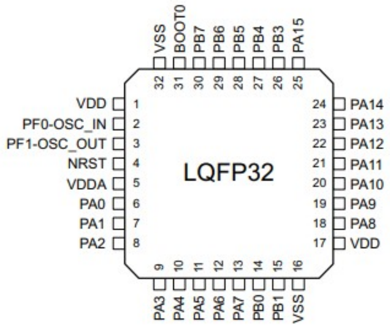


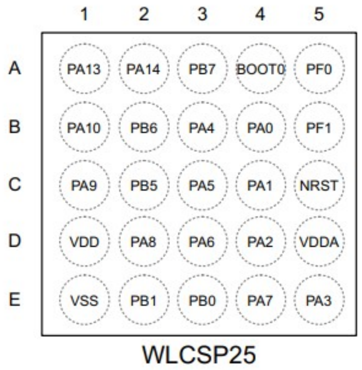
Pin PB7 for example if bit 7 of GPIO Port B
Note the strange layout of pins (why?)

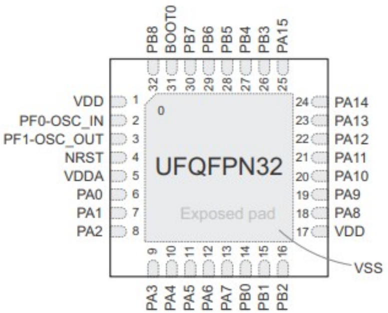


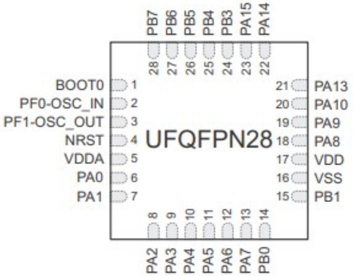
Our particular MCU : the STM32L031 is available in the following packages:

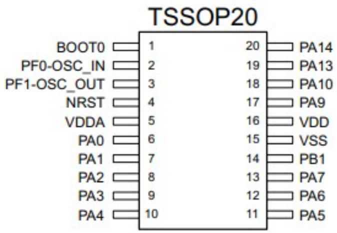




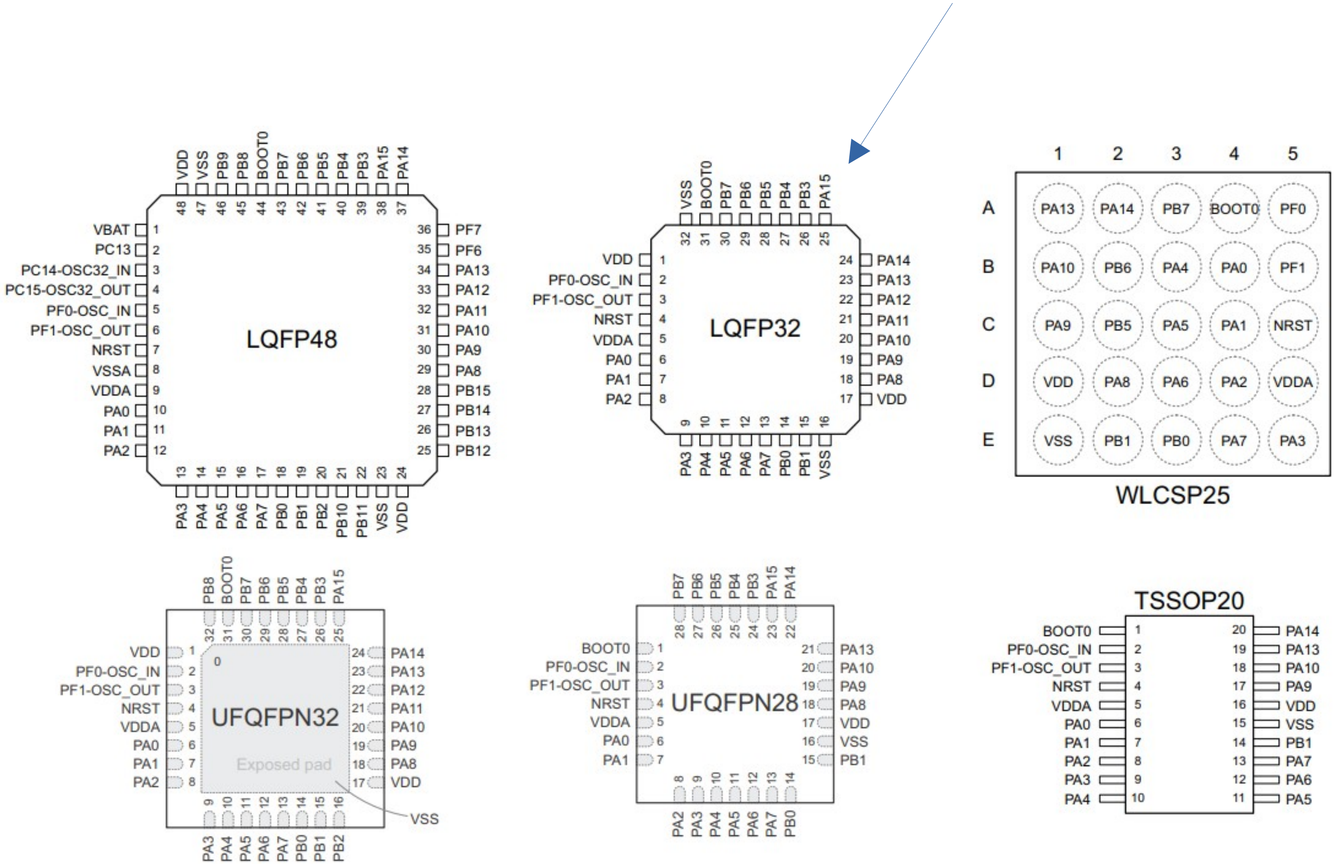




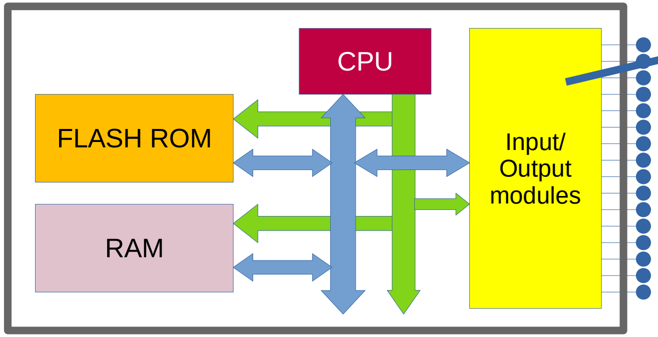


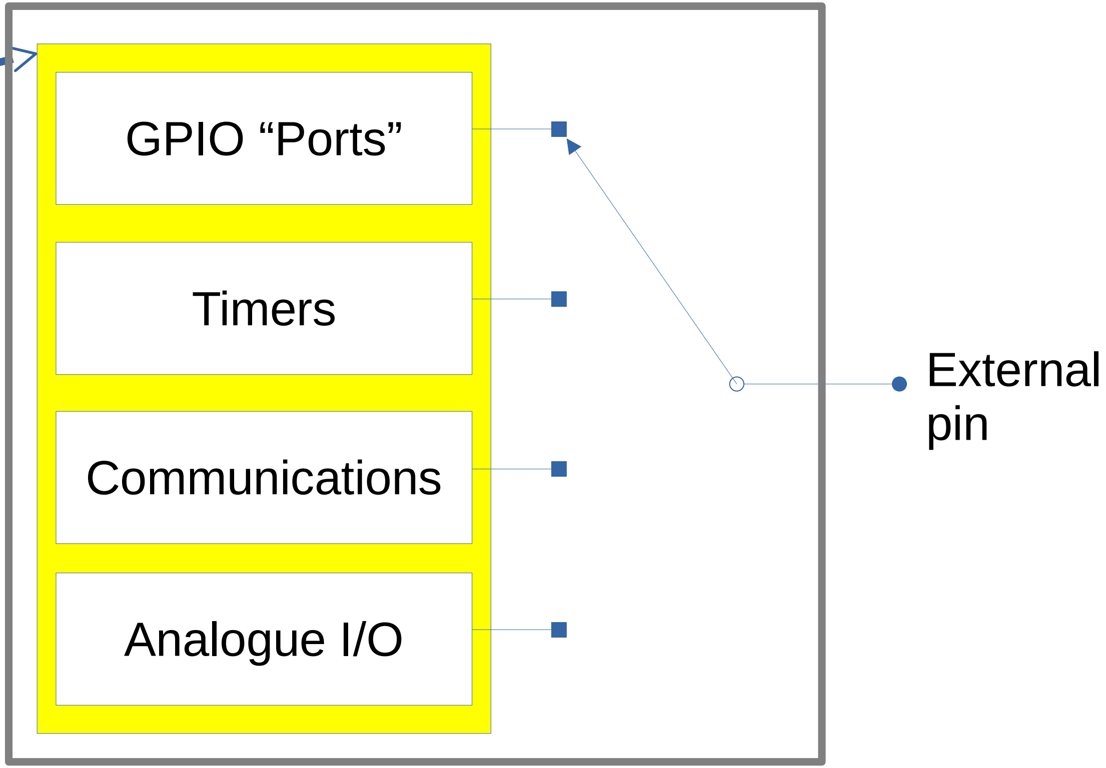


We are using the LQFP32 version



Microcontrollers: Input / Output

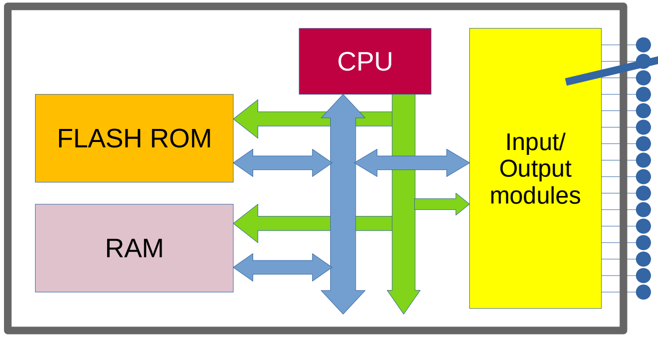


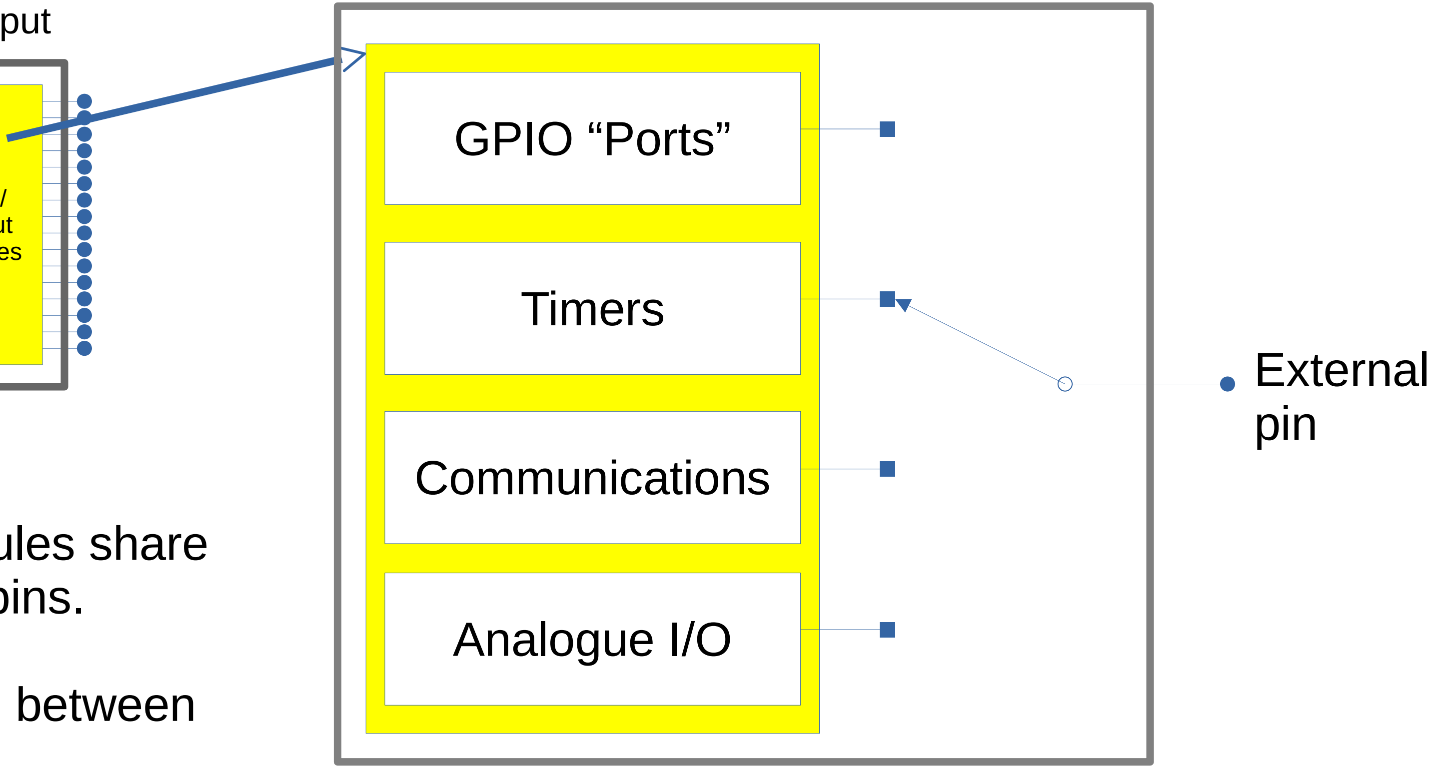


Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output

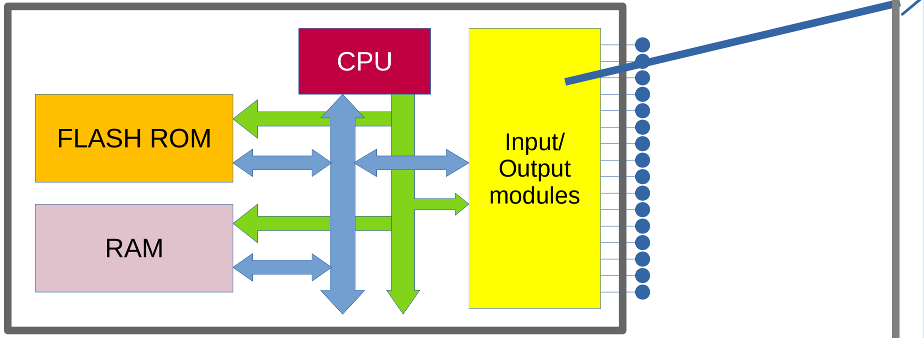


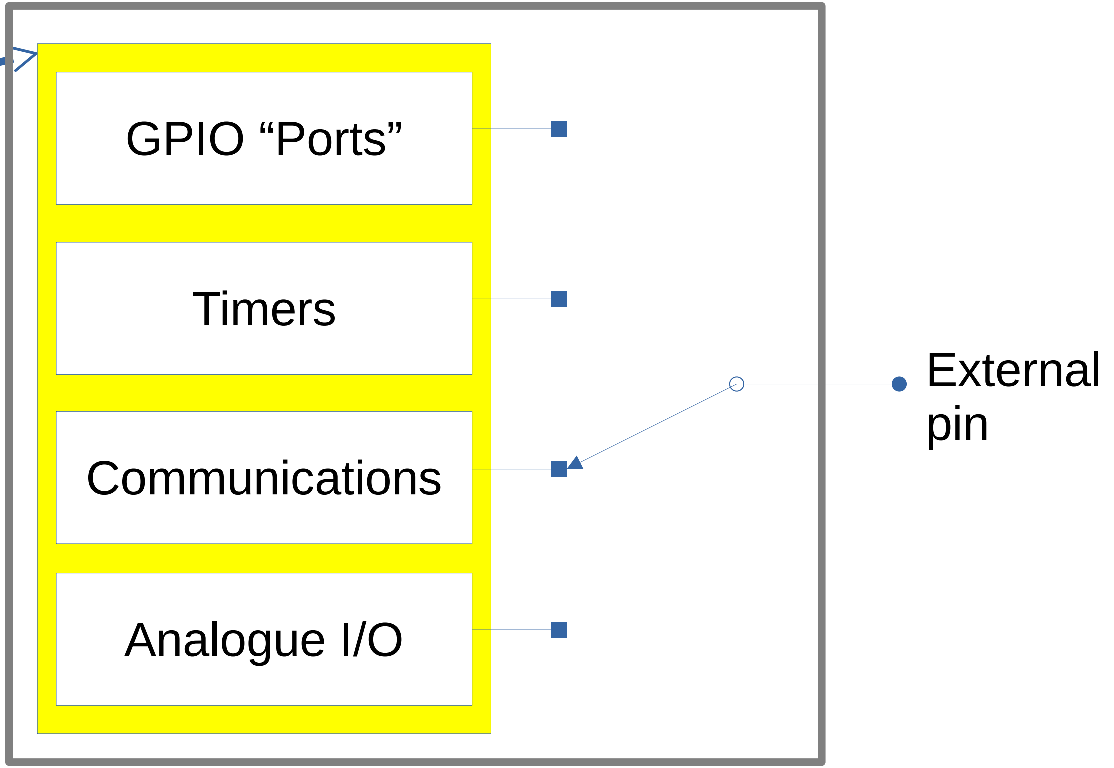


Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output

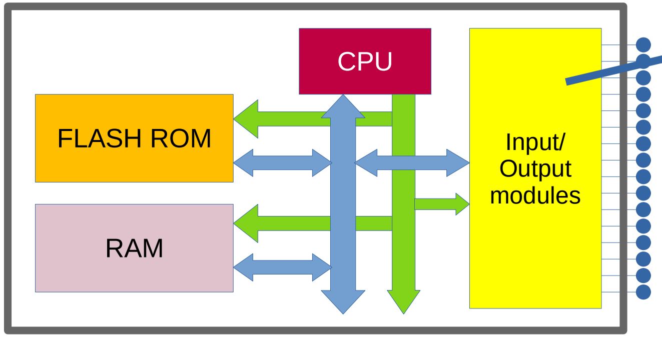


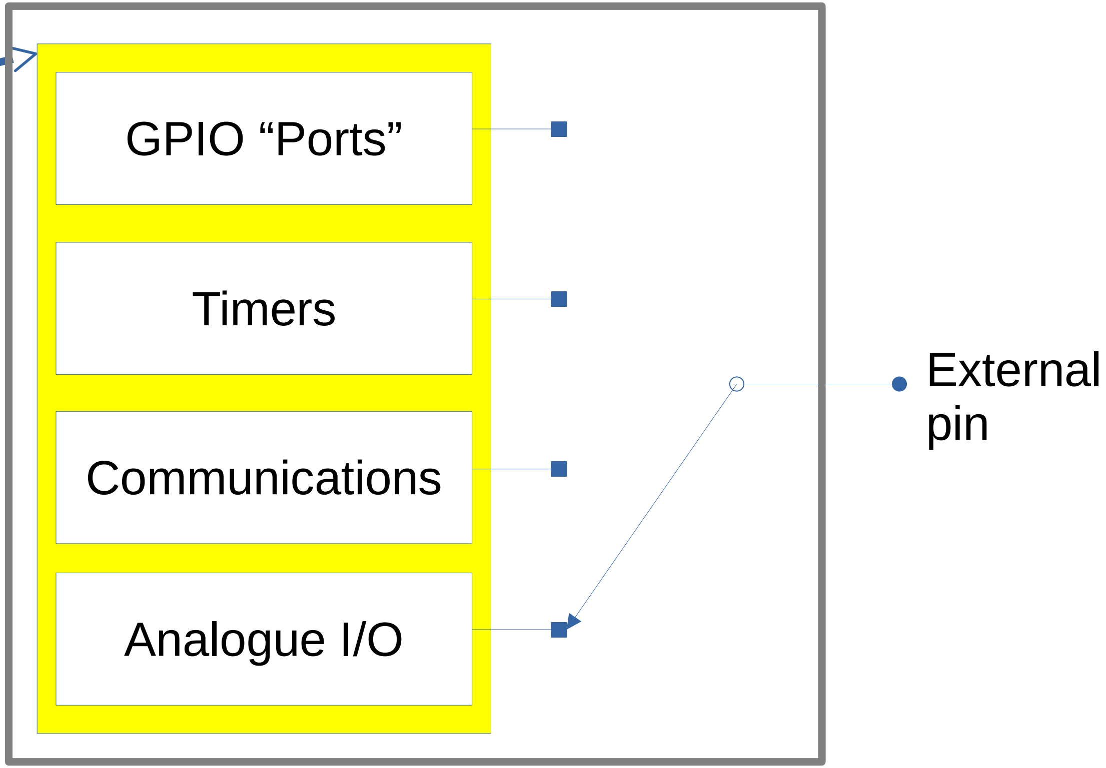


Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output





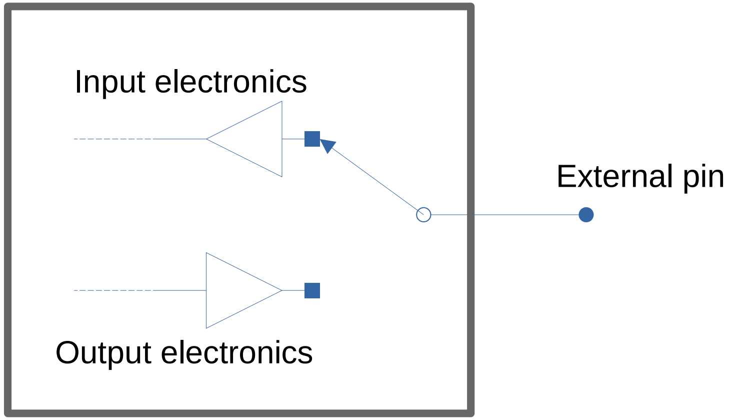
Various input/output modules share the same set of external pins.

Software can switch a pin between the different modules

Microcontrollers: Input / Output

- GPIO “Ports”
- Timers
- Communications
- Analogue I/O

GPIO pins can be configured to be digital inputs (the default case)



Microcontrollers: Input / Output

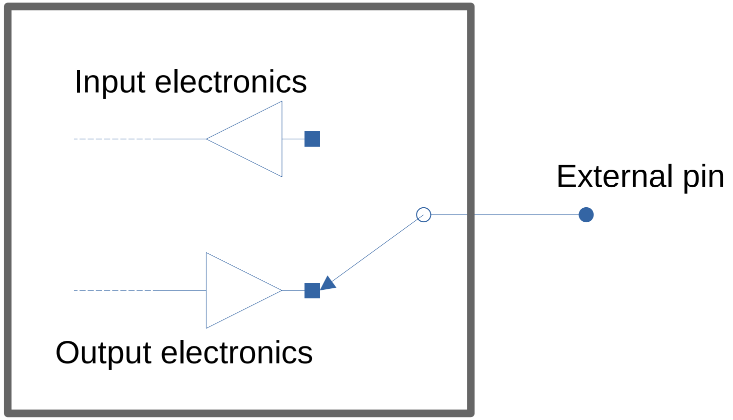
GPIO “Ports”

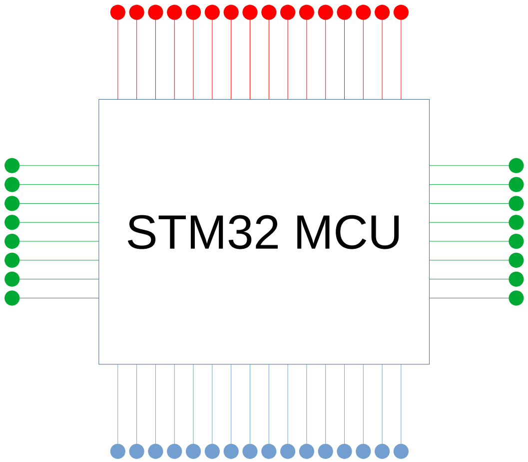
Timers

Communications

Analogue I/O

Or as outputs





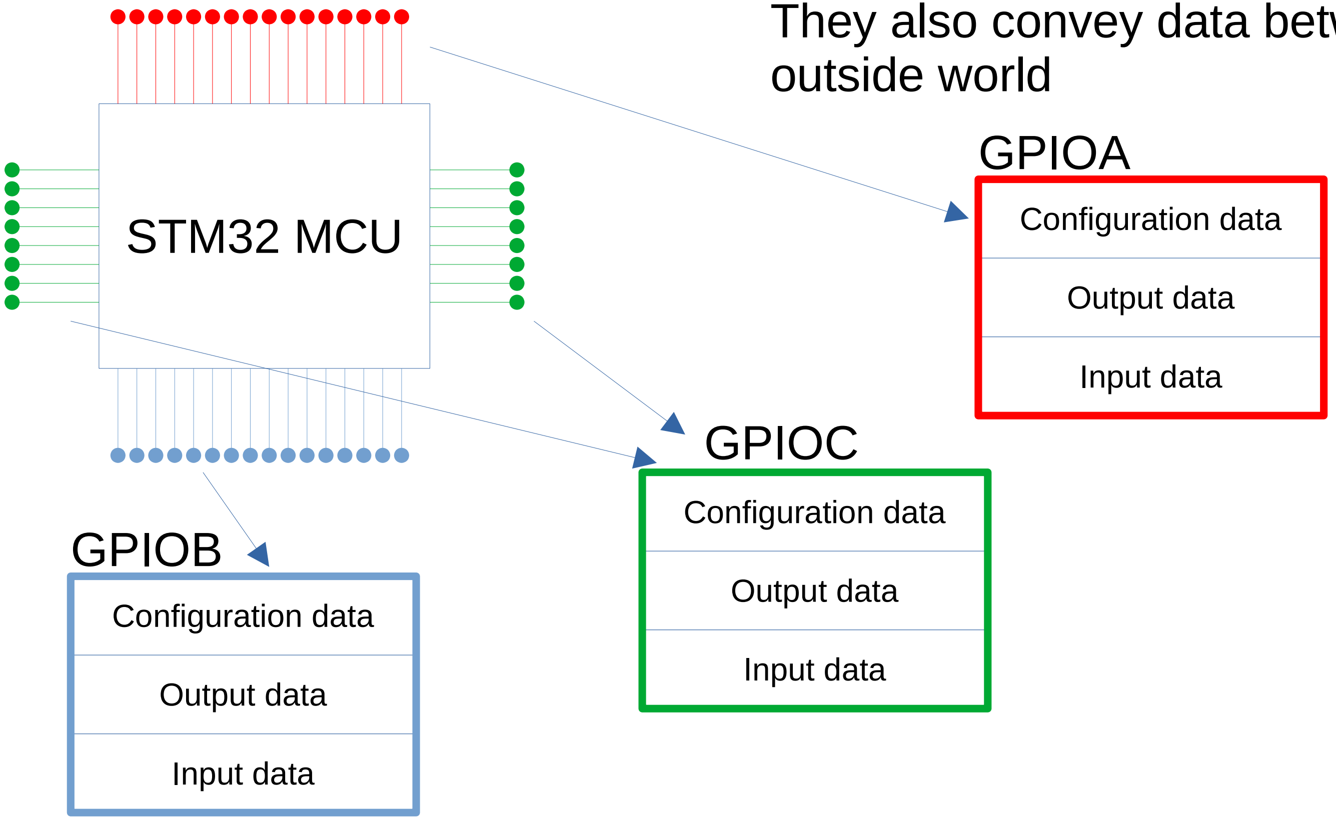
We have seen that ports require configuration data

They also convey data between the MCU and they outside world

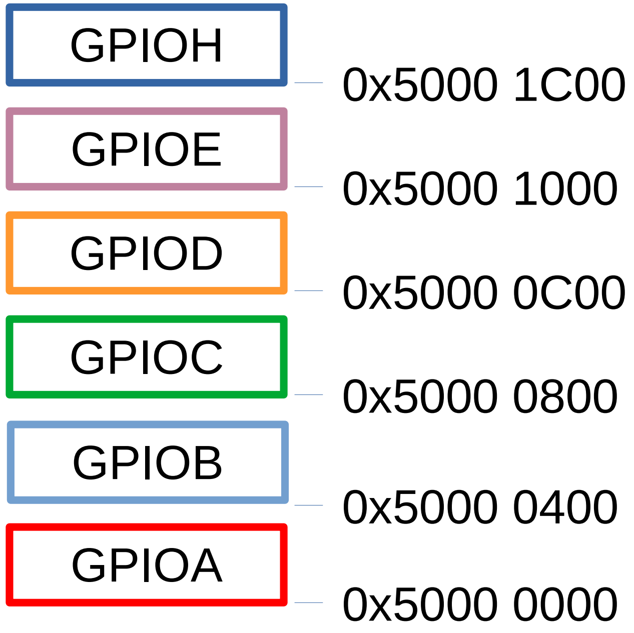
Configuration data
Output data
Input data

We have seen that ports require configuration data

They also convey data between the MCU and the outside world



The Ports (data blocks) in the STM32L031 are placed at these memory addresses



Each of these Port memory areas are organized in the same way.

GPIOA

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

GPIOA

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

These special memory locations are often called **Registers** because they do more than just store data; they change the operation of hardware

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

Mode Register:
Configure the port pins to simple inputs, or outputs or something else

Input Data Register:
Shows that status of the port pins when they are configured as simple digital inputs

Register	Address
BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

Output Data Register:
Sets the status of the port pins when configured as simple digital outputs

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

We could use these port registers in a C program as follows:

```
#include <stdint.h>
uint32_t * GPIOA_MODER = 0x50000000;
uint32_t * GPIOA_IDR = 0x50000010;
uint32_t * GPIOA_ODR = 0x50000014;

int main()
{

    *GPIOA_MODER = 1; // configure bit 0 as an output
    *GPIOA_ODR = 1; // make bit 0 a 1
    *GPIOA_ODR = 0; // make bit 0 a 0
}
```

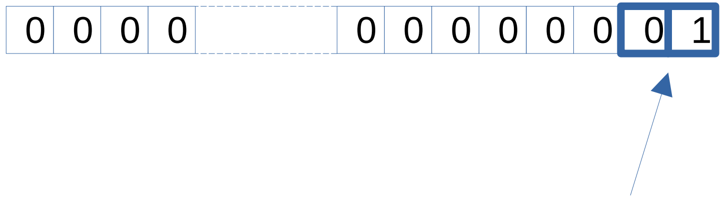
Microcontrollers: Input / Output

```c
#include <stdint.h>
uint32_t * GPIOA_MODER = 0x50000000;
uint32_t * GPIOA_IDR = 0x50000010;
uint32_t * GPIOA_ODR = 0x50000014;

int main()
{

    *GPIOA_MODER = 1; // configure bit 0 as an output
    *GPIOA_ODR = 1; // make bit 0 a 1
    *GPIOA_ODR = 0; // make bit 0 a 0
}
```

Mode register for GPIOA



Two bits are used to configure each pin. This gives 4 possibilities for each pin of GPIOA:

00 = simple digital input
01 = simple digital output
10 = alternative function
11 = analogue input

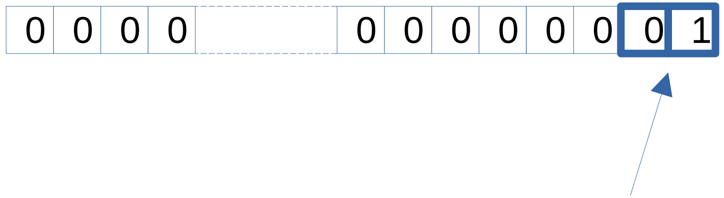
Microcontrollers: Input / Output

```c
#include <stdint.h>
uint32_t * GPIOA_MODER = 0x50000000;
uint32_t * GPIOA_IDR = 0x50000010;
uint32_t * GPIOA_ODR = 0x50000014;

int main()
{

    *GPIOA_MODER = 1; // configure bit 0 as an output
    *GPIOA_ODR = 1; // make bit 0 a 1
    *GPIOA_ODR = 0; // make bit 0 a 0
}
```

Mode register for GPIOA



Two bits are used to configure each pin.

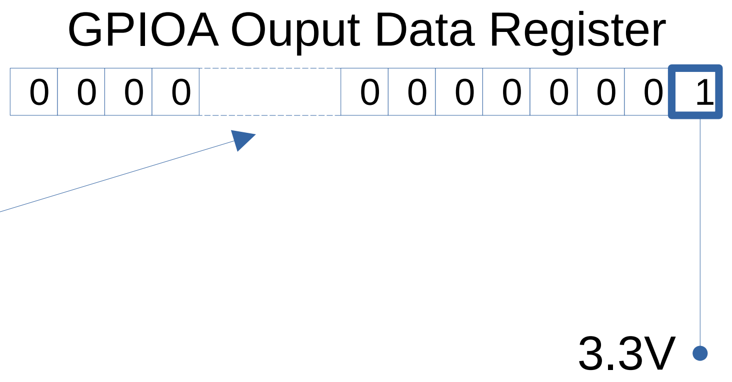
The MODE register has 32 bits so this allows it to configure up to 16 pins for GPIOA.

Microcontrollers: Input / Output

```c
#include <stdint.h>
uint32_t * GPIOA_MODER = 0x50000000;
uint32_t * GPIOA_IDR = 0x50000010;
uint32_t * GPIOA_ODR = 0x50000014;

int main()
{

    *GPIOA_MODER = 1; // configure bit 0 as an output
    *GPIOA_ODR = 1; // make bit 0 a 1
    *GPIOA_ODR = 0; // make bit 0 a 0
}
```



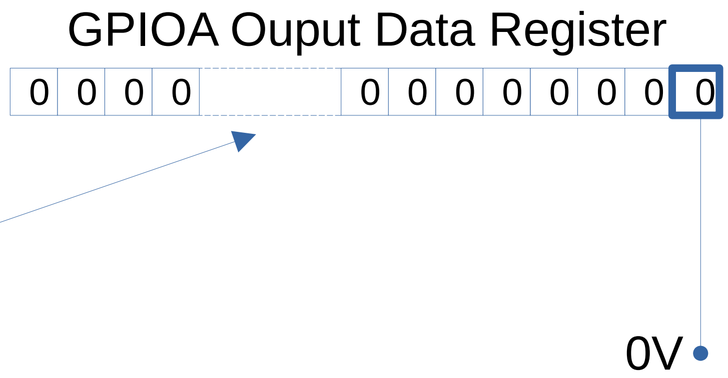
Output data register has 32 bits but only the first 16 are used

Microcontrollers: Input / Output

```c
#include <stdint.h>
uint32_t * GPIOA_MODER = 0x50000000;
uint32_t * GPIOA_IDR = 0x50000010;
uint32_t * GPIOA_ODR = 0x50000014;

int main()
{

    *GPIOA_MODER = 1; // configure bit 0 as an output
    *GPIOA_ODR = 1; // make bit 0 a 1
    *GPIOA_ODR = 0; // make bit 0 a 0
}
```



The output voltage signal could be used to turn on or off an LED or (with amplification) control bigger systems.

Microcontrollers: Input / Output

AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

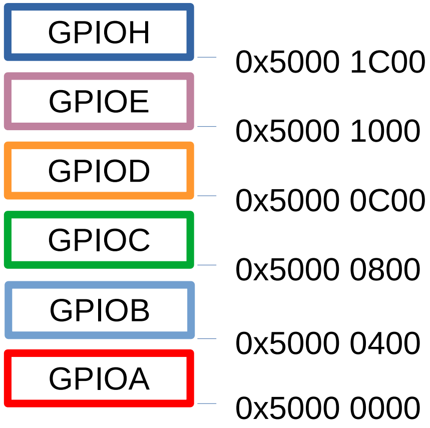
W

BRR	0x5000 0028
AFR[1]	0x5000 0024
AFR[0]	0x5000 0020
LCKR	0x5000 001C
BSRR	0x5000 0018
ODR	0x5000 0014
IDR	0x5000 0010
PUPDR	0x5000 000C
OSPEEDR	0x5000 0008
OTYPER	0x5000 0004
MODER	0x5000 0000

We can represent the Port data block as a structure as follows:

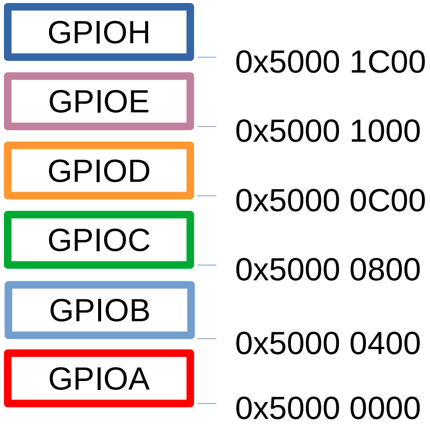
```
typedef struct
{
  uint32_t MODER;       /*!< GPIO port mode register,  Address offset: 0x00 */
  uint32_t OTYPER;      /*!< GPIO port output type register, Address offset: 0x04 */
  uint32_t OSPEEDR;     /*!< GPIO port output speed register, Address offset: 0x08 */
  uint32_t PUPDR;       /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
  uint32_t IDR;         /*!< GPIO port input data register, Address offset: 0x10 */
  uint32_t ODR;         /*!< GPIO port output data register, Address offset: 0x14 */
  uint32_t BSRR;        /*!< GPIO port bit set/reset register, Address offset: 0x18 */
  uint32_t LCKR;        /*!< GPIO port configuration lock register, Address offset: 0x1C */
  uint32_t AFR[2];      /*!< GPIO alternate function register, Address offset: 0x20-0x24 */
  uint32_t BRR;         /*!< GPIO bit reset register, Address offset: 0x28 */
} GPIO_TypeDef;
```

Microcontrollers: Input / Output



Our various ports can then be defined as follows:

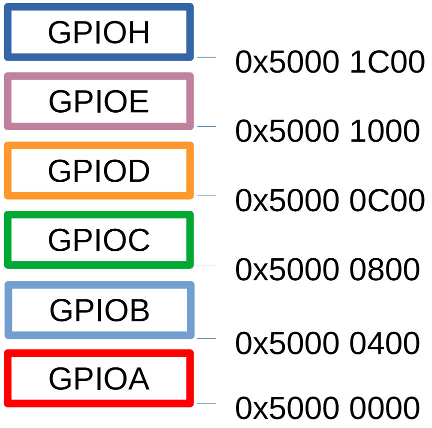
```
#define GPIOA ((GPIO_TypeDef *)0x50000000)
#define GPIOB ((GPIO_TypeDef *)0x50000400)
#define GPIOC ((GPIO_TypeDef *)0x50000800)
```



And we can use these definitions like this:

```c
int main()
{

    GPIOA->MODER = 1; // configure bit 0 as an output
    GPIOA->ODR = 1; // make bit 0 a 1
    GPIOA->ODR = 0; // make bit 0 a 0
}
```




```
int main()
{

   GPIOA->MODER = 1; // configure bit 0 as an output
   GPIOA->ODR = 1; // make bit 0 a 1
   GPIOA->ODR = 0; // make bit 0 a 0
}
```

The benefit of this approach is that it allows us write generalised functions that can operate on any port. The caller simply passes the function a reference to which port to work on.

It also means that if we port software from one STM32 device to another we only need to change the start addresses for each port data block (assuming the memory layouts are different)

Example 1: **blinky**

```
#include <stdint.h>
#include <stm32l031xx.h>
void delay(volatile uint32_t dly)
{
    while(dly--);
}
int main()
{
    RCC->IOPENR |= (1 << 1);
    GPIOB->MODER = (1 << 2*3);
    while(1)
    {
        GPIOB->ODR = (1 << 3);
        delay(100000);
        GPIOB->ODR = 0;
        delay(100000);
    }
}
```