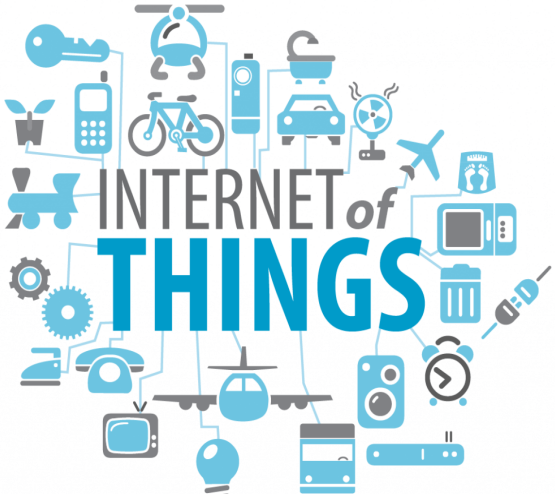
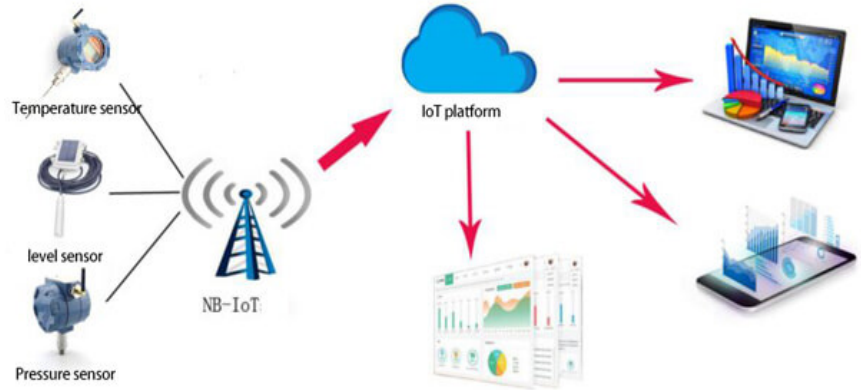
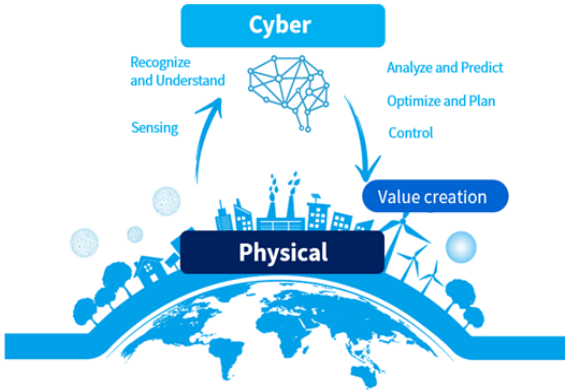
Introduction à la programmation micro-contrôleur

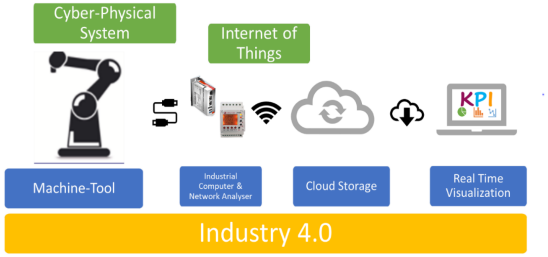
Julien Deantoni

Pourquoi ce cours ? IoT ? CPS ?



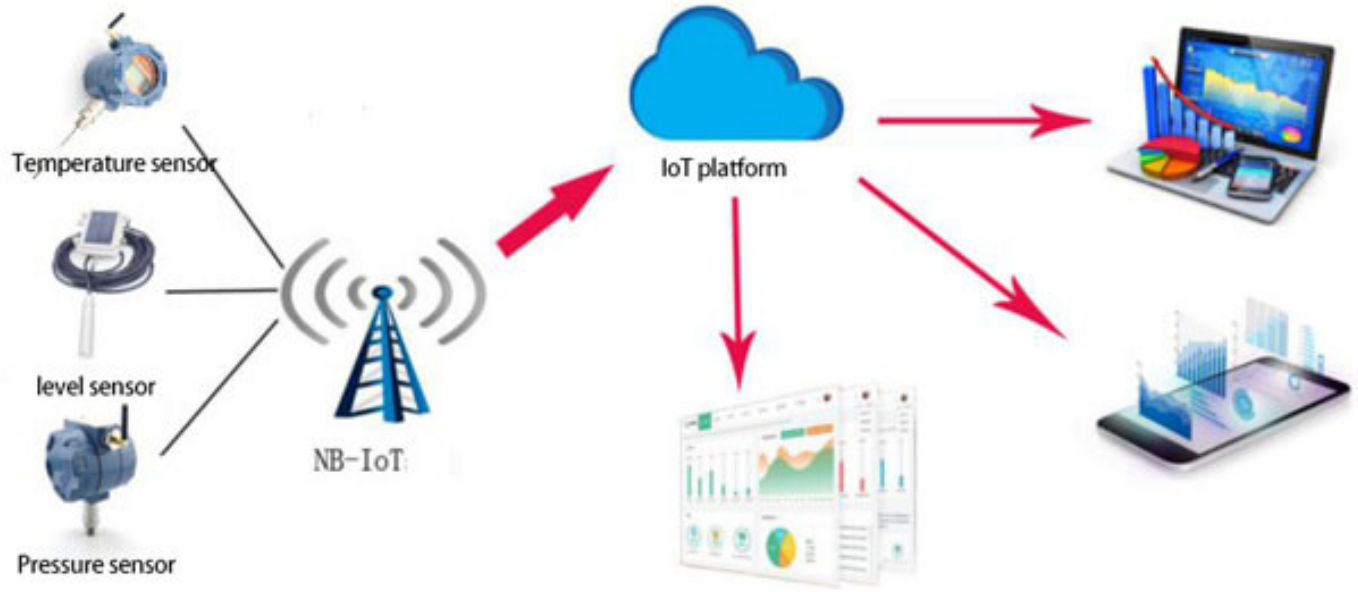






Pictures shamefully taken from the web

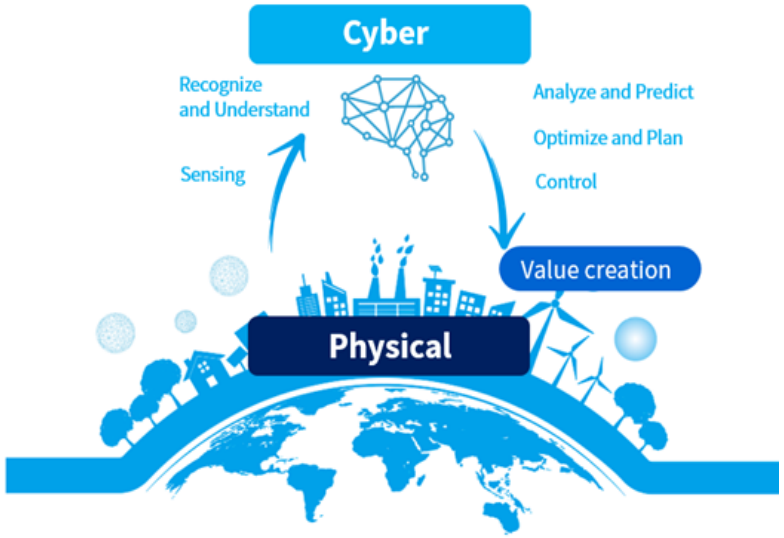
IoT yesterday



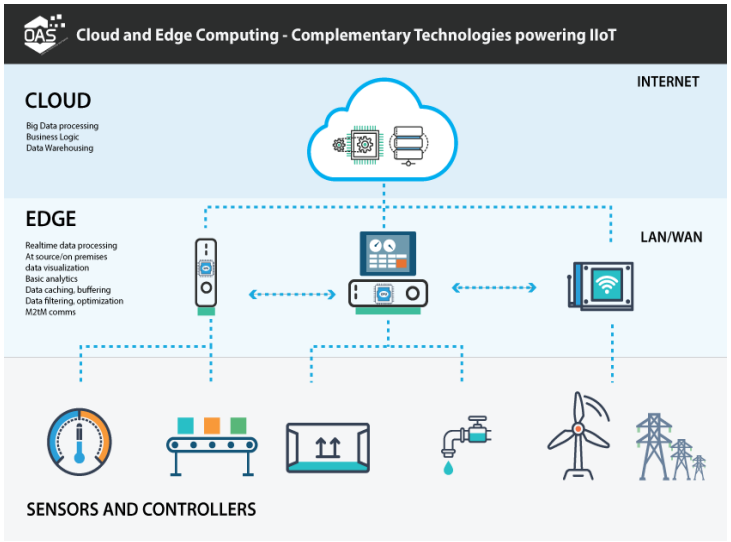
- Information gathering for analysis / dash boarding
- challenges
  - « big data »
  - scalability

Pictures shamefully taken from the web

Pourquoi ce cours ? IoT ? CPS ?

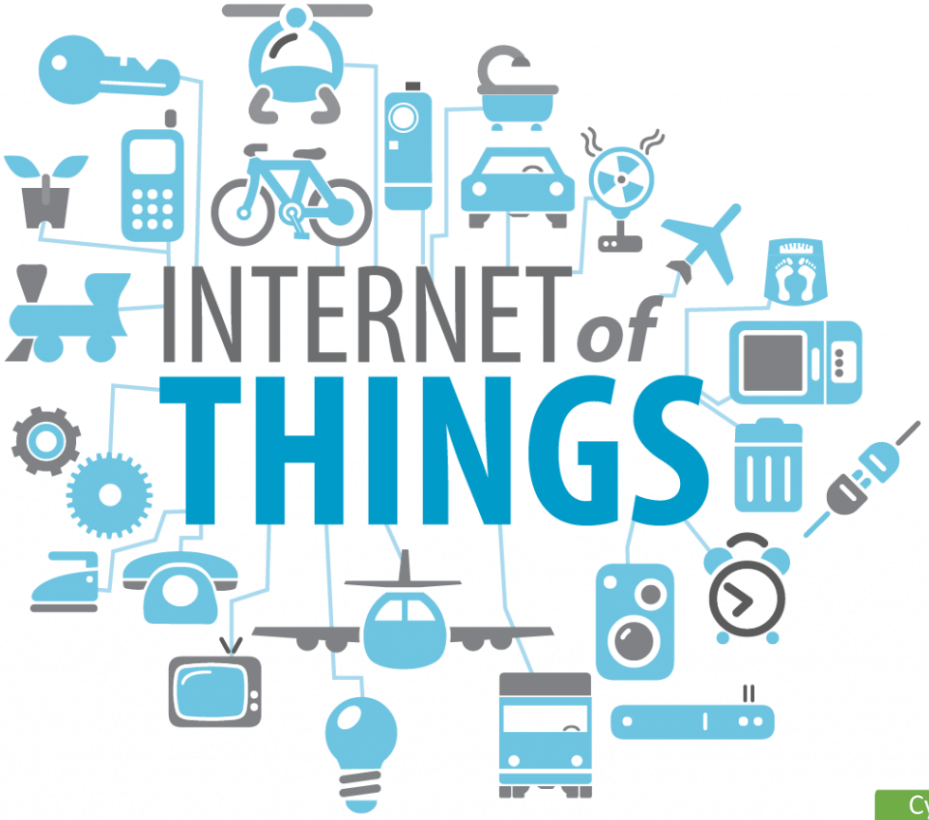


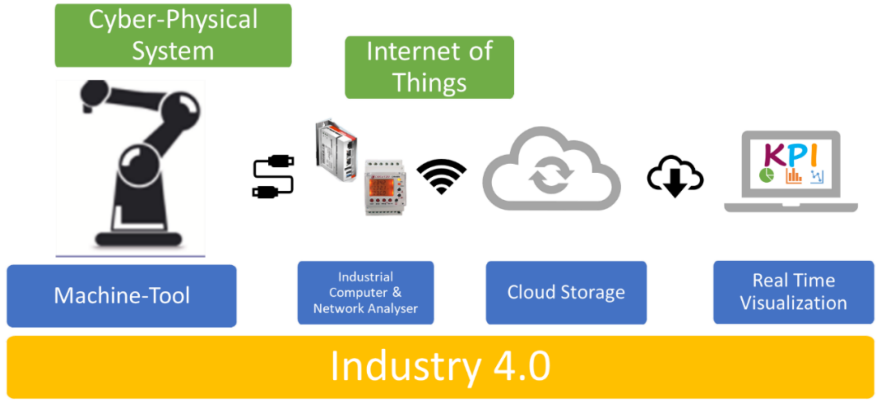
https://asia.toshiba.com/wp-content/themes/toshiba_asiapacific/img/advert-page/CNA%20Article%20-%20TOSHIBA.pdf



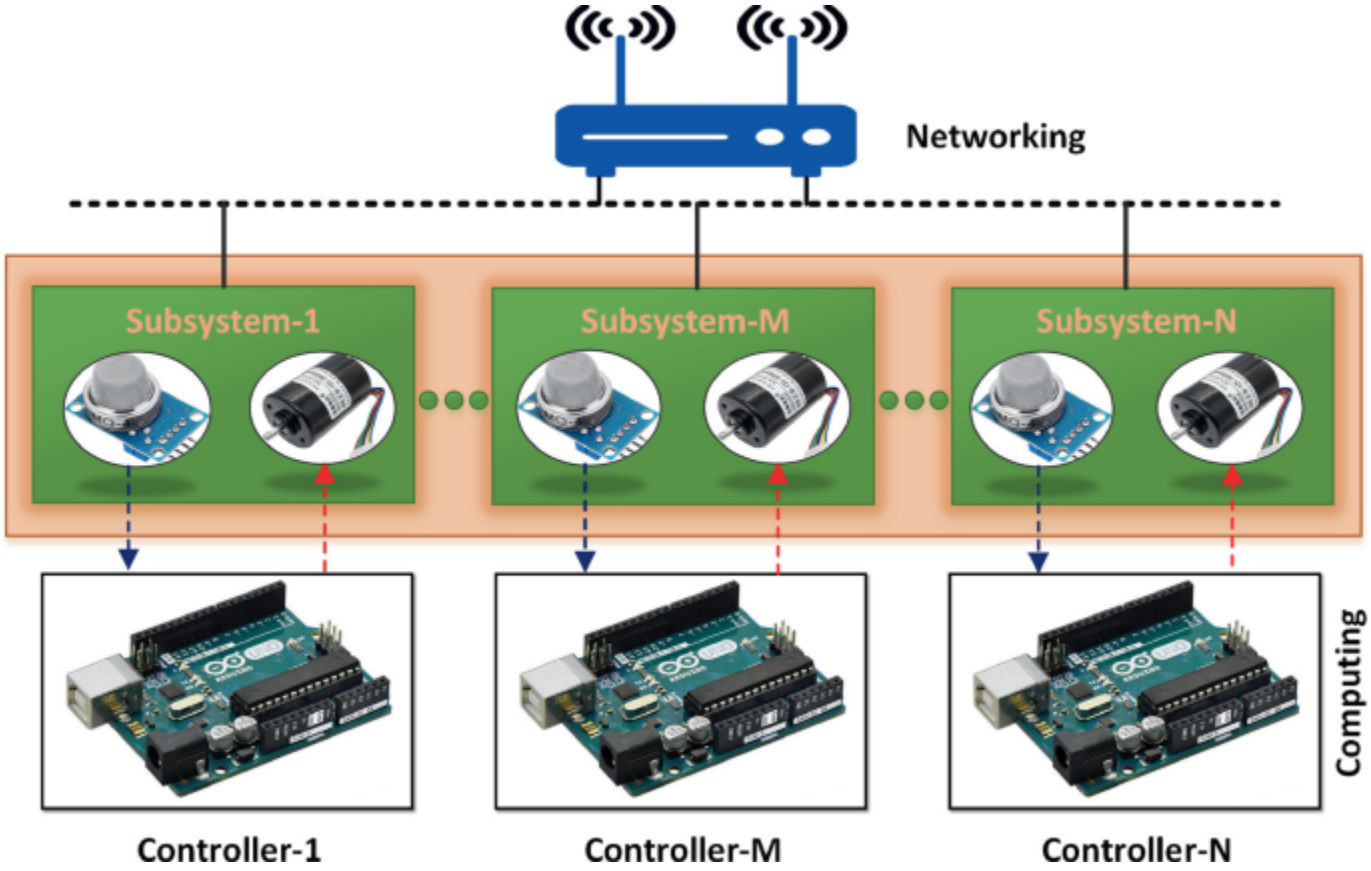
https://antlysis.com/wp-content/uploads/2019/08/edge-v-cloud-computing-graphic.png

IoT and CPS convergence



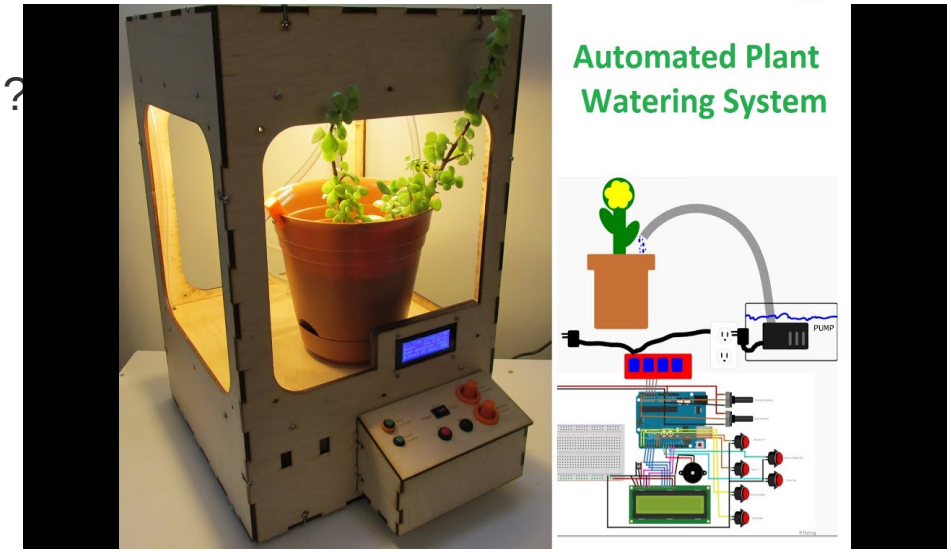


Pictures shamefully taken from the web


Networking
Subsystem-1
Subsystem-M
Subsystem-N
Computing
Controller-1
Controller-M
Controller-N

Pourquoi ce cours ?

- Que ce passe t'il en bout de chaine ?
  - Capteurs
  - Actionneurs
  - Simple contrôle
  - (communication)

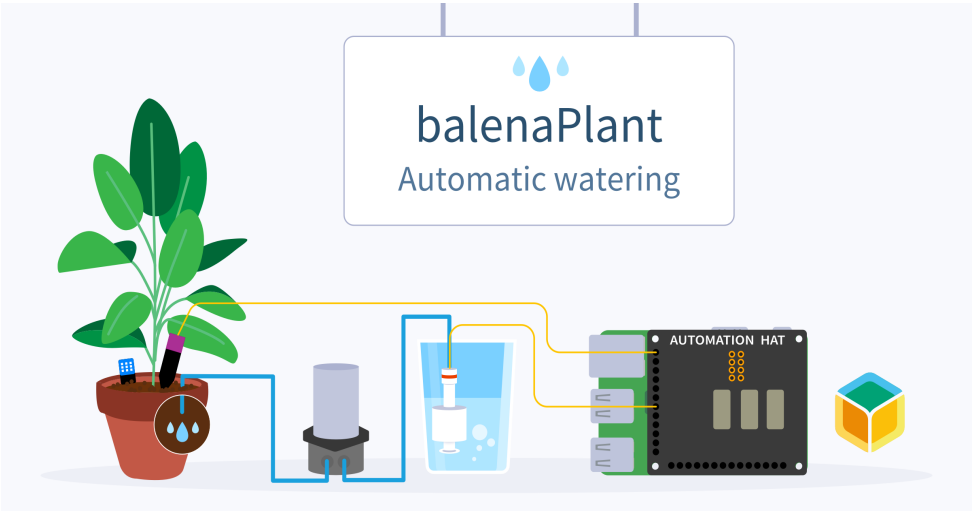


https://www.youtube.com/watch?v=8ZWP9jgEhm4

- Différents besoins
  - Pas d'OS
  - OS temps réel dédiés
  - Linux embarqués ou pseudo Linux

Pourquoi ce cours ?

- Que ce passe t'il en bout de chaine ?
  - Capteurs
  - Actionneurs
  - contrôle (bang bang, pid)
  - (communication)

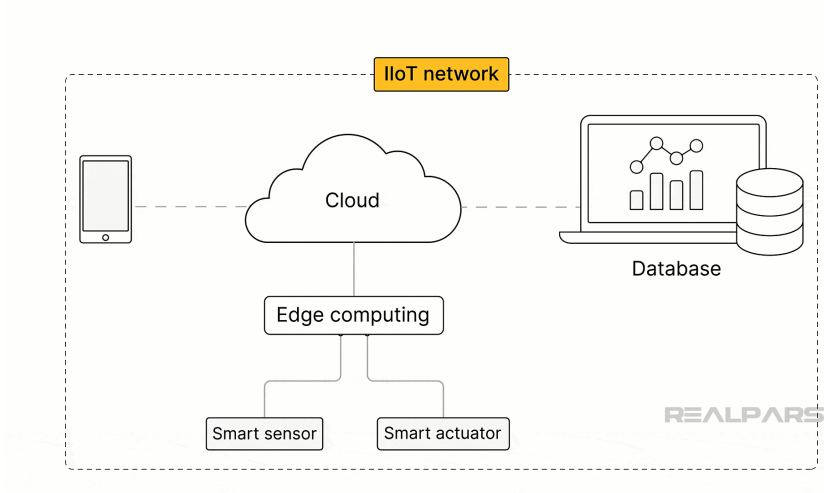


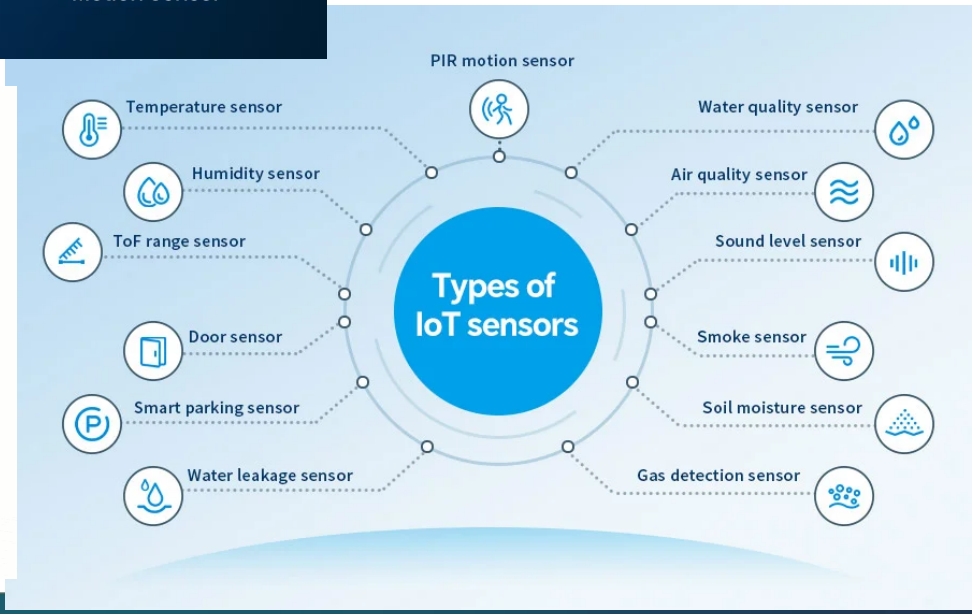
https://www.balena.io/blog/keep-your-plant-friends-happy-and-healthy-with-automated-watering-balenaplant/

- Différents besoins
  - Pas d'OS
  - OS temps réel dédiés
  - Linux embarqués ou pseudo Linux

https://fr.aliexpress.com/i/4000439727068.html#!


An All-Inclusive Guide On
The Top IoT Sensors In The Market
intuz
IoT
Infrared Sensor
Pressure Sensor
Accelerometer Sensor
Temperature Sensor
Proximity Sensor
Gyroscope Sensor
Smoke Sensor
Motion Sensor

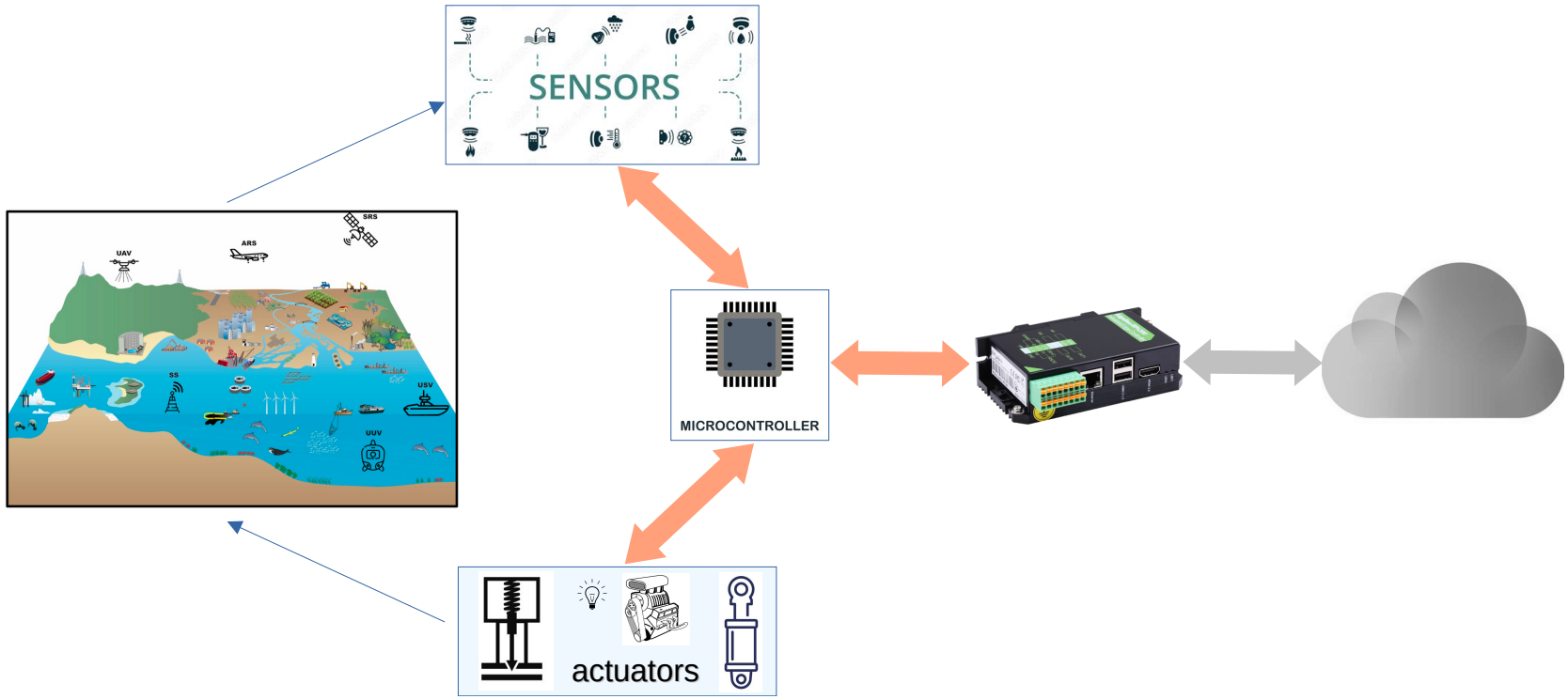

IIoT network
Cloud
Database
Edge computing
Smart sensor
Smart actuator
REALPARS


PIR motion sensor
Temperature sensor
Water quality sensor
Humidity sensor
Air quality sensor
ToF range sensor
Sound level sensor
Types of IoT sensors
Door sensor
Smoke sensor
Smart parking sensor
Soil moisture sensor
Water leakage sensor
Gas detection sensor

Smart Sensors/Actuators

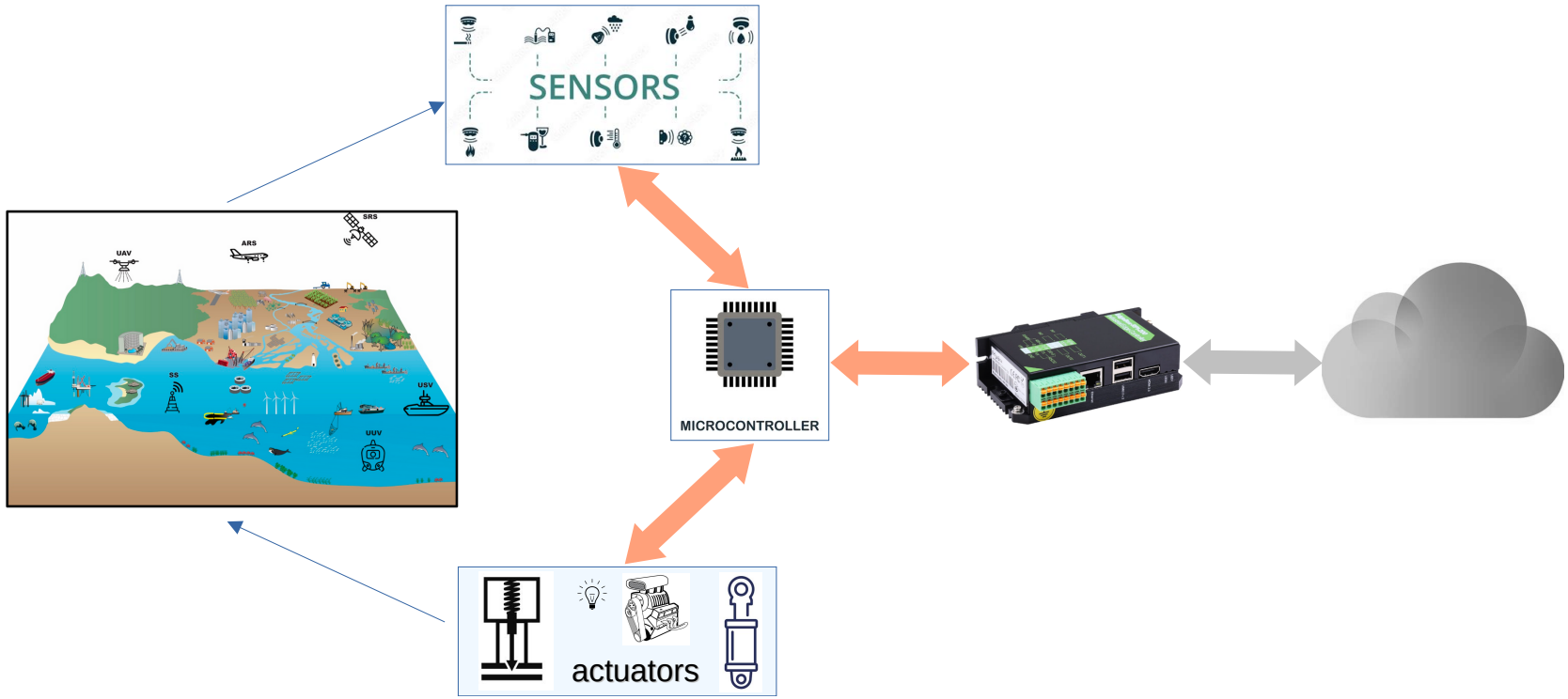
- Smart sensors :
  - Les capteurs intelligents (smart sensors) sont des dispositifs qui combinent la détection physique d'un signal (comme la température, la pression ou la lumière) avec des capacités de traitement et de communication intégrées. Ils peuvent analyser les données localement, filtrer les informations pertinentes et les transmettre à d'autres systèmes, améliorant ainsi la précision, la rapidité et l'autonomie des applications connectées.
- Smart actuators :
  - Les actionneurs intelligents (smart actuators) sont des dispositifs capables de réaliser une action (comme déplacer, ouvrir ou réguler) tout en intégrant des fonctions de traitement et de communication. Ils peuvent ainsi ajuster leur fonctionnement et échanger des informations avec d'autres systèmes pour plus d'efficacité et d'autonomie

Les systèmes considérés



- Systèmes temps réels
  - En charge du contrôle d'un processus
  - Liés à la dynamique du processus à contrôler
  - Soumis à des contraintes temporelles

Les systèmes considérés



- Systèmes temps réels
  - Pas forcément rapides (contrairement aux idées reçues)
  - Prédictibles
  - (Souvent) Fortement enfouis

Notions importantes

Prédictibilité

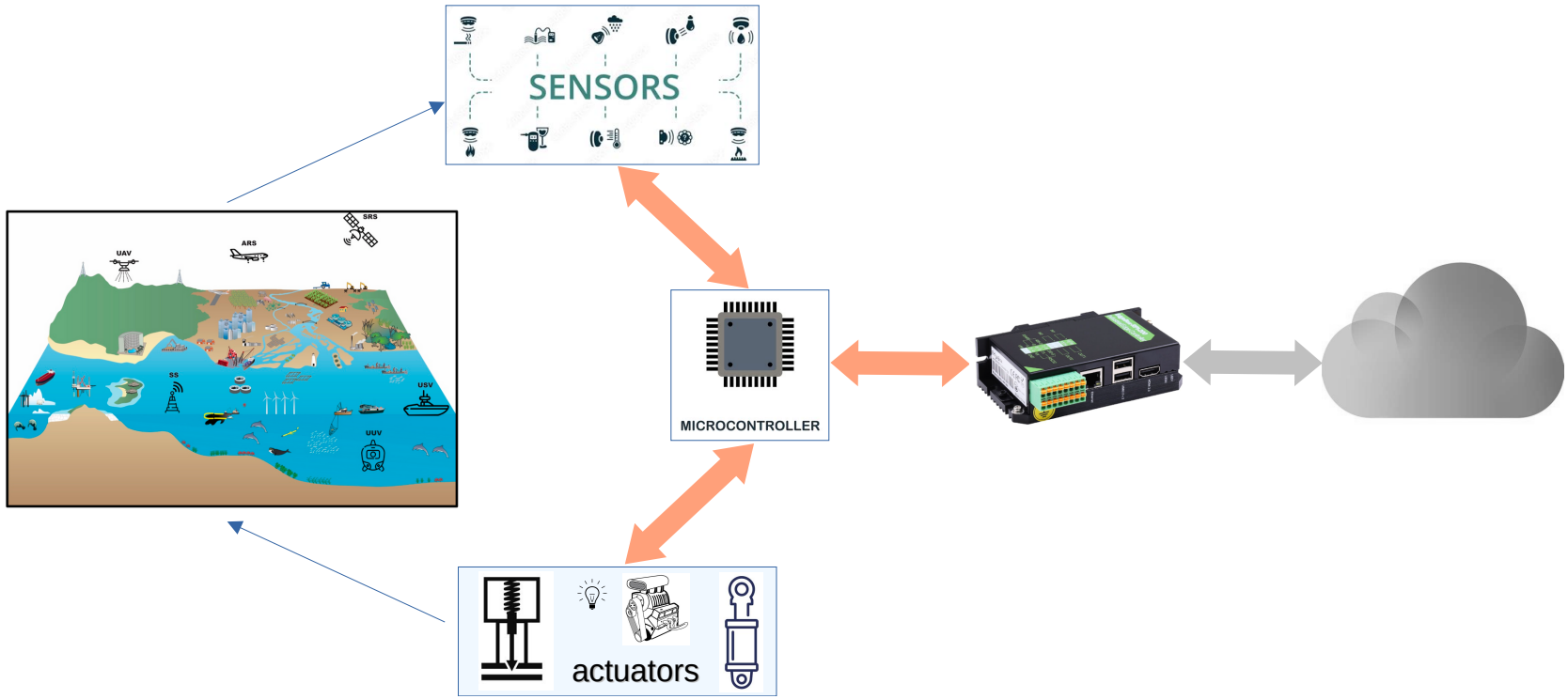
- On veut pouvoir prédire le comportement d'un système
  - Fonctionnel: le comportement est déterministe, i.e. un même jeu de donnée en entrée produit toujours la même sortie
  - Temporel: le calcul doit être fait avant des **échéances déterminées**

Notions importantes

concurrence

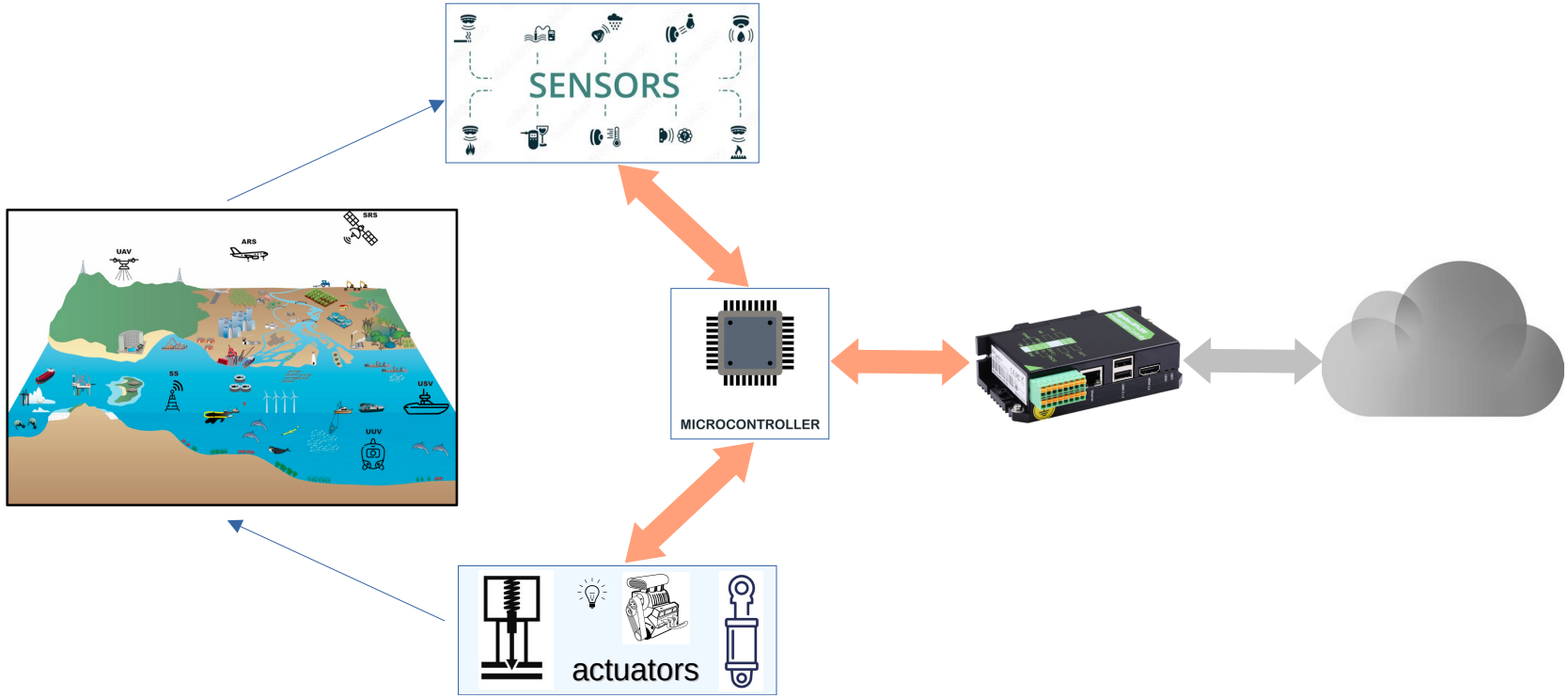
- La spécification du système est pensée de manière concurrente
  - Car l'environnement est parallèle (composés de processus indépendants ou faiblement couplés: le moteur, les essuies glace, l’opérateur, etc)
  - Car le système est composé de plusieurs activités plus ou moins critiques se faisant à des rythmes distincts.

Les systèmes considérés



- Systèmes temps réels
  - Contraintes matérielles fortes
    - Mémoire
    - Coût
    - Taille
    - Consommation énergétique

Les systèmes embarqués considérés

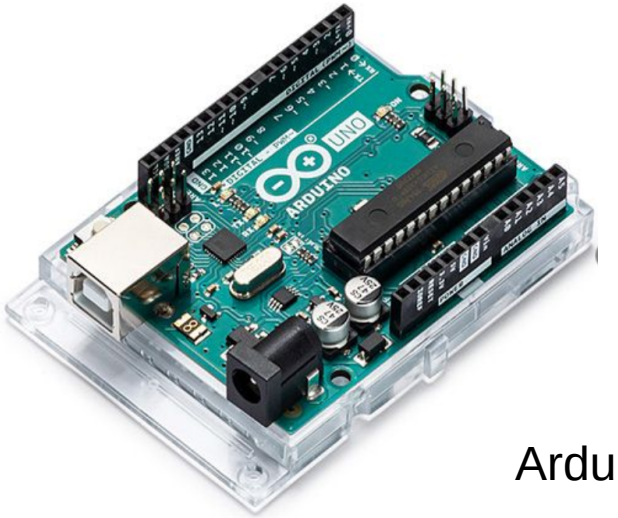


- Systèmes Critiques
  - Une faille peut mettre des vies en danger
    - Régulateur de vitesse
    - ABS
  - Prédictibilité accrue (déterminisme)

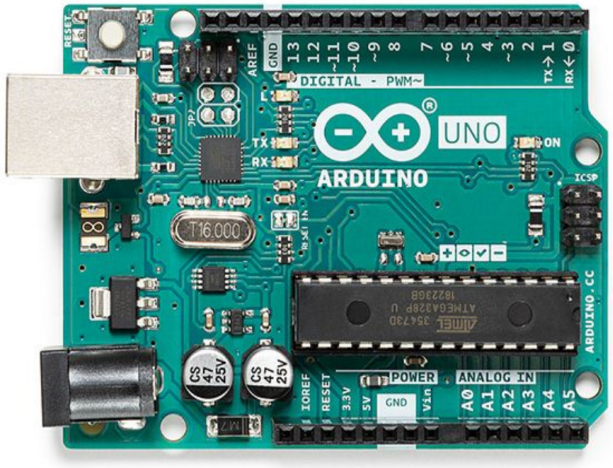
Contenu du cours

- Généralités
  - Les systèmes considérés
  - **Le développement de systèmes à base de micro contrôleurs**
- Programmation sans OS
  - micro-contrôleur sans OS, pourquoi, comment ?
  - Stratégie d'implémentation
    - programmation sans IT (Synchrone)
    - programmation avec IT (Asynchrone)
- Mise en oeuvre
- Programmation avec un OS temps réel...

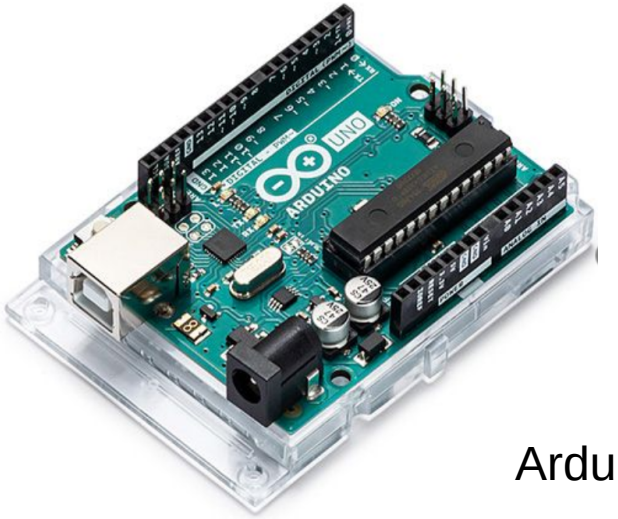
Matériel utilisé dans ce cours



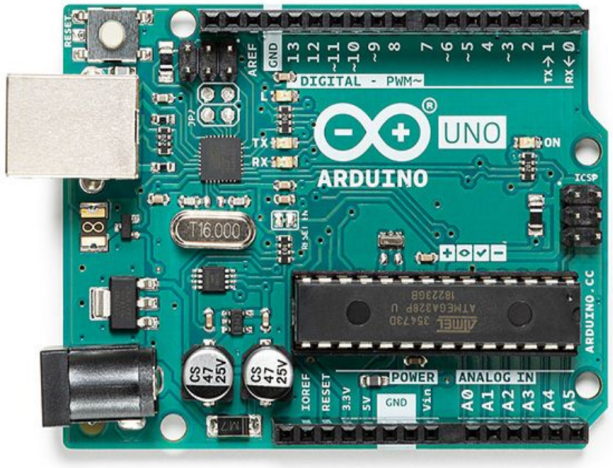
Arduino Uno !



Matériel utilisé dans ce cours



Arduino Uno !



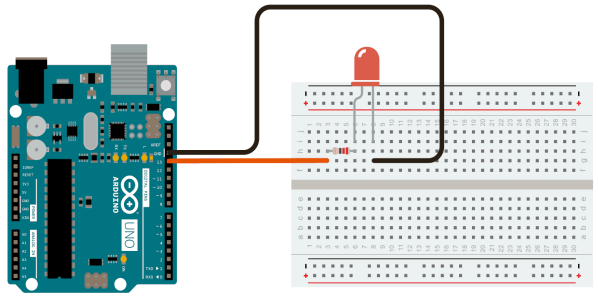
At what age can I introduce Arduino starter projects to kids?

You can start pretty early. Kids **between 8-10 years old** should be under adult supervision. First, make sure your kids understand all the basics. Start with small projects. Repeat the basics and evolve step-by-step. **So yes, Arduino is for kids!**

For them to be more independent, the ideal age is 12 years and up. This will always depend on your kid interests and the type of Arduino starter kit you buy; board and components are quite similar but not the manuals (some are easier to understand than others).

Arduino lib

```
1  const int ledPin =  LED_BUILTIN;   // the number of the LED pin
2  int ledState = LOW;                // ledState used to set the LED
3  unsigned long previousMillis = 0;  // will store last time LED was updated
4  const long interval = 1000;        // interval at which to blink (milliseconds)
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);           // set the digital pin as output
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (currentMillis - previousMillis >= interval) {
15     previousMillis = currentMillis;
16
17     if (ledState == LOW) {
18       ledState = HIGH;
19     } else {
20       ledState = LOW;
21     }
22
23     digitalWrite(ledPin, ledState);
24   }
25 }
```

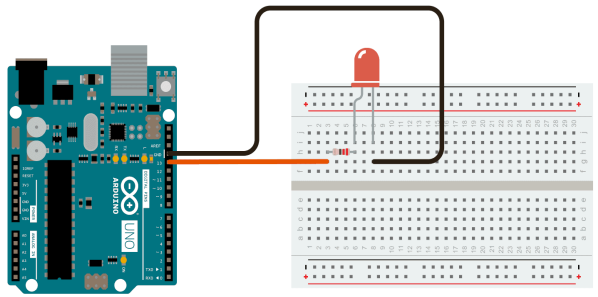


Arduino lib

hardware/arduino/avr/variants/standard/pins_arduino.h

```
54  #define LED_BUILTIN 13
```

```
1  const int ledPin =  LED_BUILTIN;   // the number of the LED pin
2  int ledState = LOW;                // ledState used to set the LED
3  unsigned long previousMillis = 0;  // will store last time LED was updated
4  const long interval = 1000;        // interval at which to blink (milliseconds)
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);           // set the digital pin as output
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (currentMillis - previousMillis >= interval) {
15     previousMillis = currentMillis;
16
17     if (ledState == LOW) {
18       ledState = HIGH;
19     } else {
20       ledState = LOW;
21     }
22
23     digitalWrite(ledPin, ledState);
24   }
25 }
```



Arduino lib

hardware/arduino/avr/variants/standard/pins_arduino.h

```
54  #define LED_BUILTIN 13
```

```
1  const int ledPin =  LED_BUILTIN;   // the number of the LED pin
2  int ledState = LOW;                // ledState used to se
3  unsigned long previousMillis = 0;  // will store last tim
4  const long interval = 1000;        // interval at which t
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);           // set the digital p
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (currentMillis - previousMillis >= interval) {
15     previousMillis = currentMillis;
16
17     if (ledState == LOW) {
18       ledState = HIGH;
19     } else {
20       ledState = LOW;
21     }
22
23     digitalWrite(ledPin, ledState);
24   }
25 }
```

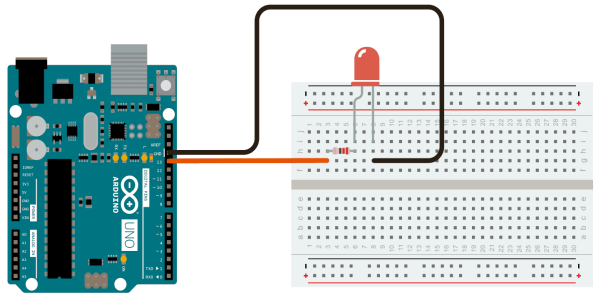
```
void pinMode(uint8_t pin, uint8_t mode)
{
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *reg, *out;

    if (port == NOT_A_PIN) return;

    // JWS: can I let the optimizer do this?
    reg = portModeRegister(port);
    out = portOutputRegister(port);

    if (mode == INPUT) {
        uint8_t oldSREG = SREG;
                cli();
        *reg &= ~bit;
        *out &= ~bit;
        SREG = oldSREG;
    } else if (mode == INPUT_PULLUP) {
        uint8_t oldSREG = SREG;
                cli();
        *reg &= ~bit;
        *out |= bit;
        SREG = oldSREG;
    } else {
        uint8_t oldSREG = SREG;
                cli();
        *reg |= bit;
        SREG = oldSREG;
    }
}
```

Arduino lib

hardware/arduino/avr/variants/standard/pins_arduino.h

```
54  #define LED_BUILTIN 13
```

```
1  const int ledPin =  LED_BUILTIN;   // the number of the LED pin
2  int ledState = LOW;                // ledState used to set the LED
3  unsigned long previousMillis = 0;  // will store last time LED was updated
4  const long interval = 1000;        // interval at which to blink (milliseconds)
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);           // set the digital pin as output
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (currentMillis - previousMillis >= interval) {
15     previousMillis = currentMillis;
16
17     if (ledState == LOW) {
18       ledState = HIGH;
19     } else {
20       ledState = LOW;
21     }
22
23     digitalWrite(ledPin, ledState);
24   }
25 }
```



Arduino lib

```
1  const int ledPin =  LED_BUILTIN;    // the
2  int ledState = LOW;                 // ledS
3  unsigned long previousMillis = 0;   // will
4  const long interval = 1000;         // inte
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);            // se
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (currentMillis - previousMillis >= in
15     previousMillis = currentMillis;
16
17     if (ledState == LOW) {
18       ledState = HIGH;
19     } else {
20       ledState = LOW;
21     }
22
23     digitalWrite(ledPin, ledState);
24   }
25 }
```

hardware/a
54 #def

```
// the prescaler is set so that timer0 ticks every 64 clock cycles, and the
// the overflow handler is called every 256 ticks.
#define MICROSECONDS_PER_TIMER0_OVERFLOW (clockCyclesToMicroseconds(64 * 256))

// the whole number of milliseconds per timer0 overflow
#define MILLIS_INC (MICROSECONDS_PER_TIMER0_OVERFLOW / 1000)

// the fractional number of milliseconds per timer0 overflow. we shift right
// by three to fit these numbers into a byte. (for the clock speeds we care
// about - 8 and 16 MHz - this doesn't lose precision.)
#define FRACT_INC ((MICROSECONDS_PER_TIMER0_OVERFLOW % 1000) >> 3)
#define FRACT_MAX (1000 >> 3)

volatile unsigned long timer0_overflow_count = 0;
volatile unsigned long timer0_millis = 0;
static unsigned char timer0_fract = 0;

#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
ISR(TIM0_OVF_vect)
#else
ISR(TIMER0_OVF_vect)
#endif
{
    // copy these to local variables so they can be stored in registers
    // (volatile variables must be read from memory on every access)
    unsigned long m = timer0_millis;
    unsigned char f = timer0_fract;

    m += MILLIS_INC;
    f += FRACT_INC;
    if (f >= FRACT_MAX) {
        f -= FRACT_MAX;
        m += 1;
    }

    timer0_fract = f;
    timer0_millis = m;
    timer0_overflow_count++;
}

unsigned long millis()
{
    unsigned long m;
    uint8_t oldSREG = SREG;

    // disable interrupts while we read timer0_millis or we might get an
    // inconsistent value (e.g. in the middle of a write to timer0_millis)
    cli();
    m = timer0_millis;
    SREG = oldSREG;

    return m;
}
```

Arduino lib

```
1  const int ledPin =  LED_BUILTIN;    // the
2  int ledState = LOW;                 // ledS
3  unsigned long previousMillis = 0;   // will
4  const long interval = 1000;         // inte
5
6  void setup() {
7    pinMode(ledPin, OUTPUT);            // se
8  }
9
10 void loop() {
11
12   unsigned long currentMillis = millis();
13
14   if (curre
15     previou
16
17     if (led
18       ledSt
       } else
         ledSt
       }

       digital
24   }
25 }
```

hardware/a
54 #def

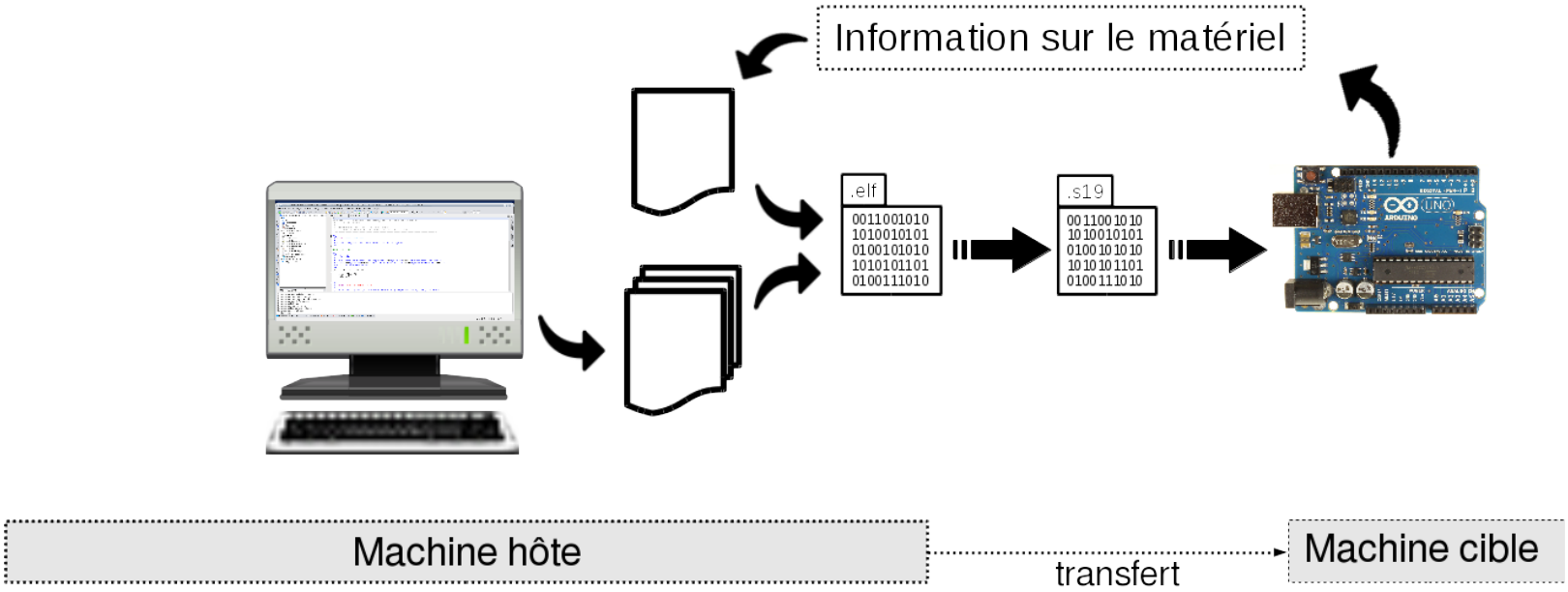
```
// the prescaler is set so that timer0 ticks every 64 clock cycles, and the
// the overflow handler is called every 256 ticks.
#define MICROSECONDS_PER_TIMER0_OVERFLOW (clockCyclesToMicroseconds(64 * 256))

// the whole number of milliseconds per timer0 overflow
#define MILLIS_INC (MICROSECONDS_PER_TIMER0_OVERFLOW / 1000)

// the fractional number of milliseconds per timer0 overflow. we shift right
// by three to fit these numbers into a byte. (for the clock speeds we care
// about - 8 and 16 MHz - this doesn't lose precision.)
#define FRACT_INC ((MICROSECONDS_PER_TIMER0_OVERFLOW % 1000) >> 3)
#define FRACT_MAX (1000 >> 3)

volatile unsigned long timer0_overflow_count = 0;
volatile unsigned long timer0_millis = 0;
static unsigned char timer0_fract = 0;

#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
ISR(TIM0_OVF_vect)
#else
ISR(TIMER0_OVF_vect)
#endif
{
    // copy these to local variables so they can be stored in registers
    // (volatile variables must be read from memory on every access)
    unsigned long m = timer0_millis;
    unsigned char f = timer0_fract;

    m += MILLIS_INC;
                                                    an
                                                    llis)
    m = timer0_millis;
    SREG = oldSREG;

    return m;
}
```

At what age can I introduce Arduino starter projects to kids?

You can start pretty early. Kids **between 8-10 years old** should be under adult supervision. First, make sure your kids understand all the basics. Start with small projects. Repeat the basics and evolve step-by-step. **So yes, Arduino is for kids!**

For them to be more independent, the ideal age is 12 years and up. This will always depend on your kid interests and the type of Arduino starter kit you buy; board and components are quite similar but not the manuals (some are easier to understand than others).

Le développement de systèmes temps réel version simple :-)

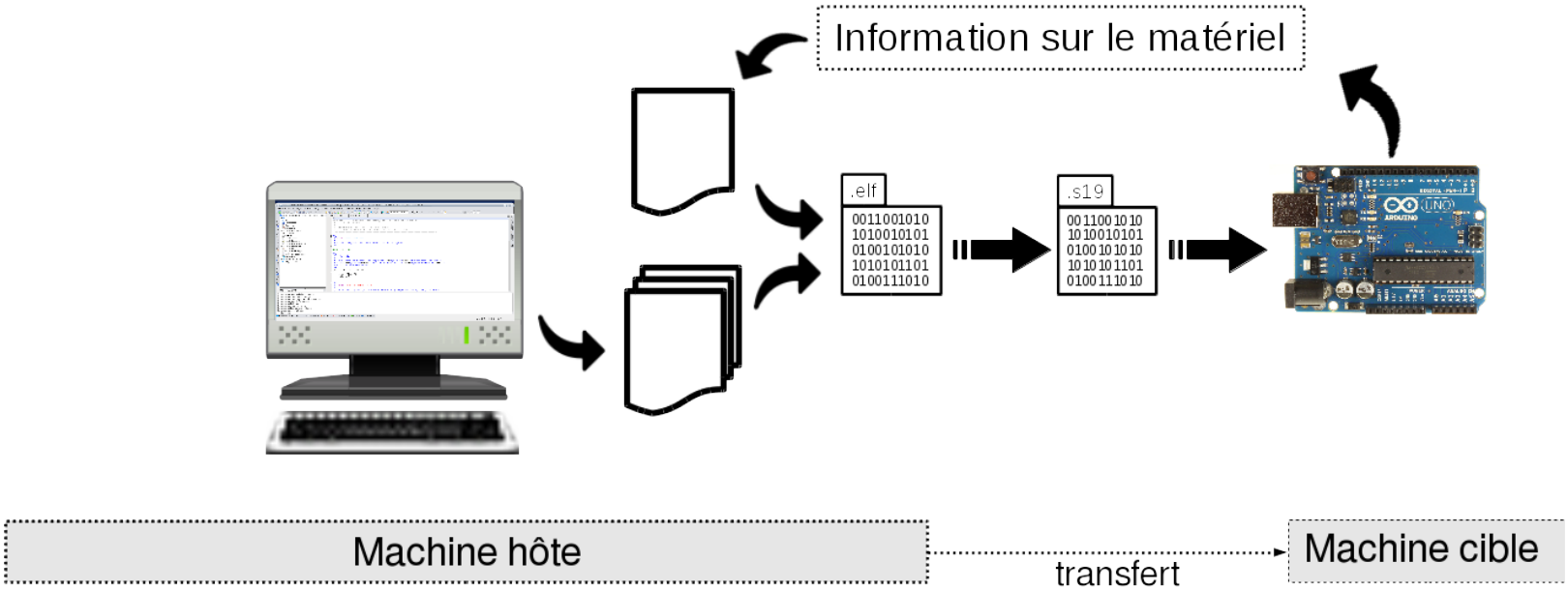


- **Différence forte machine hôte / machine cible**

Machine hôte :

- Entrées / Sortie Standard
- IDE (spécifiques ou pas)
- Simulation de la cible / environnement / processus

Le développement de systèmes temps réel version simple :-)

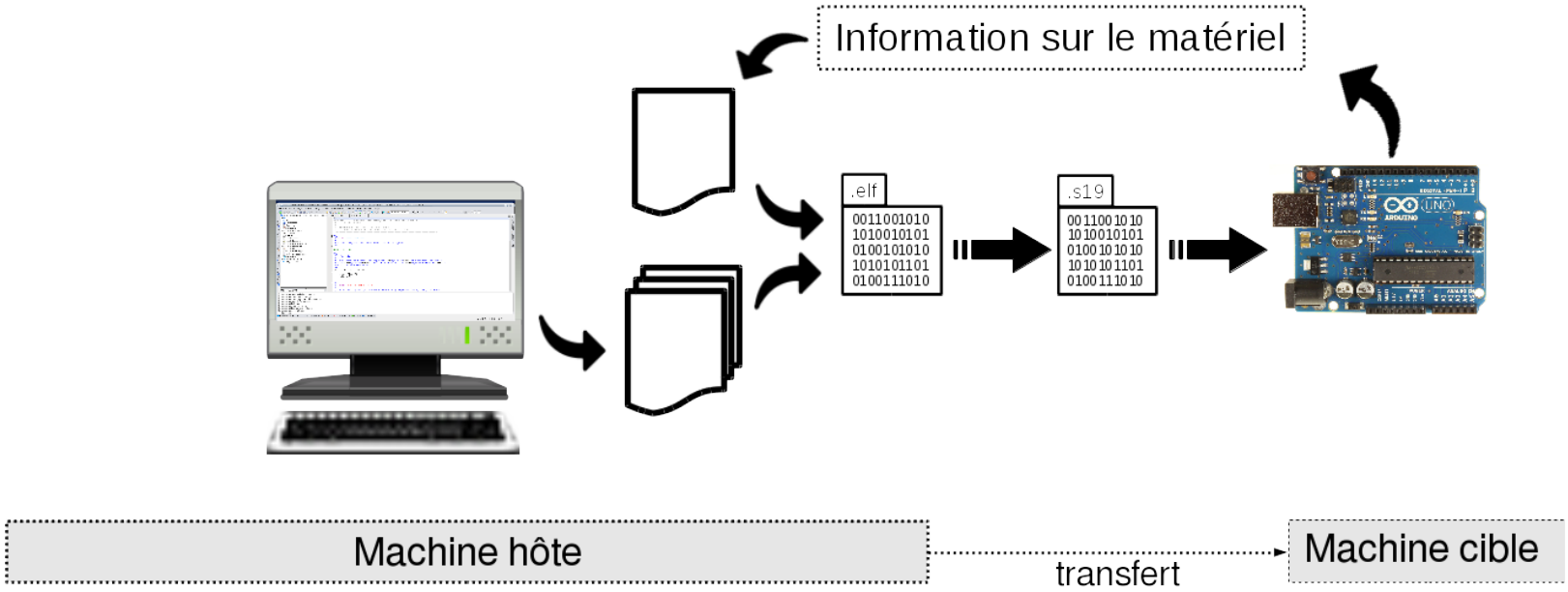


- **Différence forte machine hôte / machine cible**

  Machine cible :

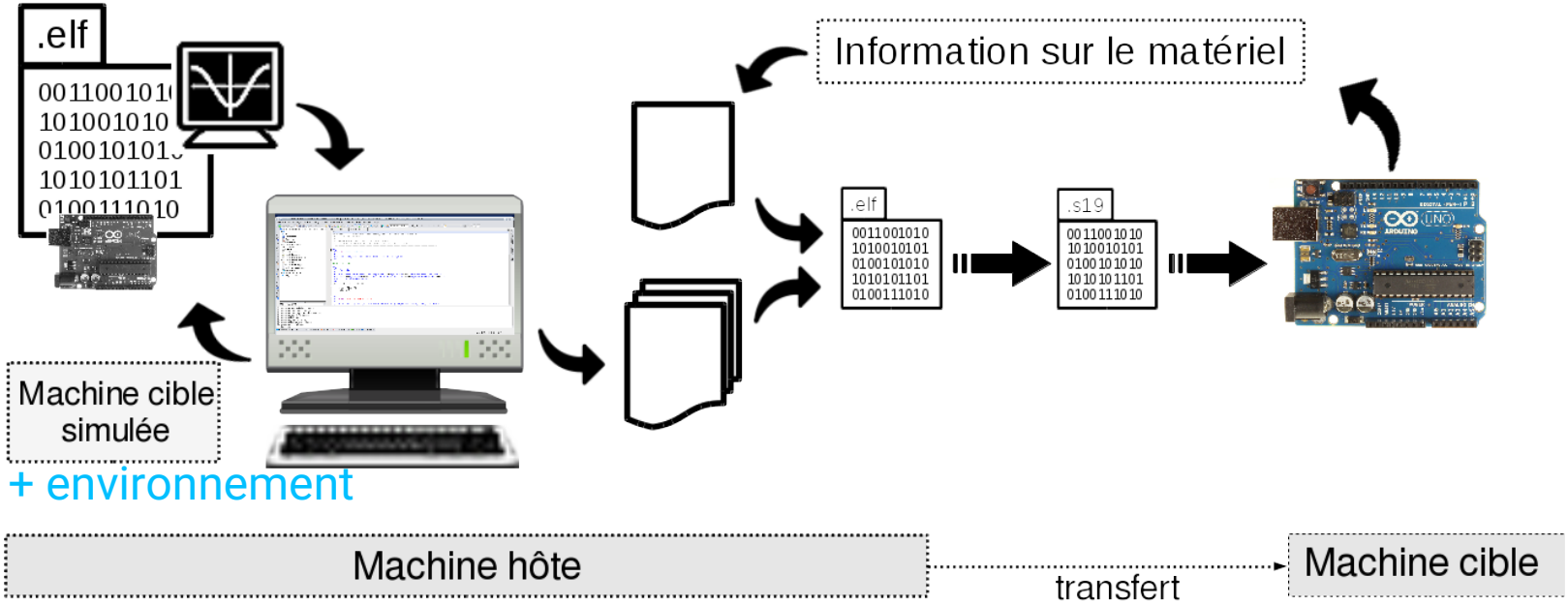
  - Entrées / Sorties ?
  - Difficilement utilisable hors de son **environnement**

Le développement de systèmes temps réel version simple :-)



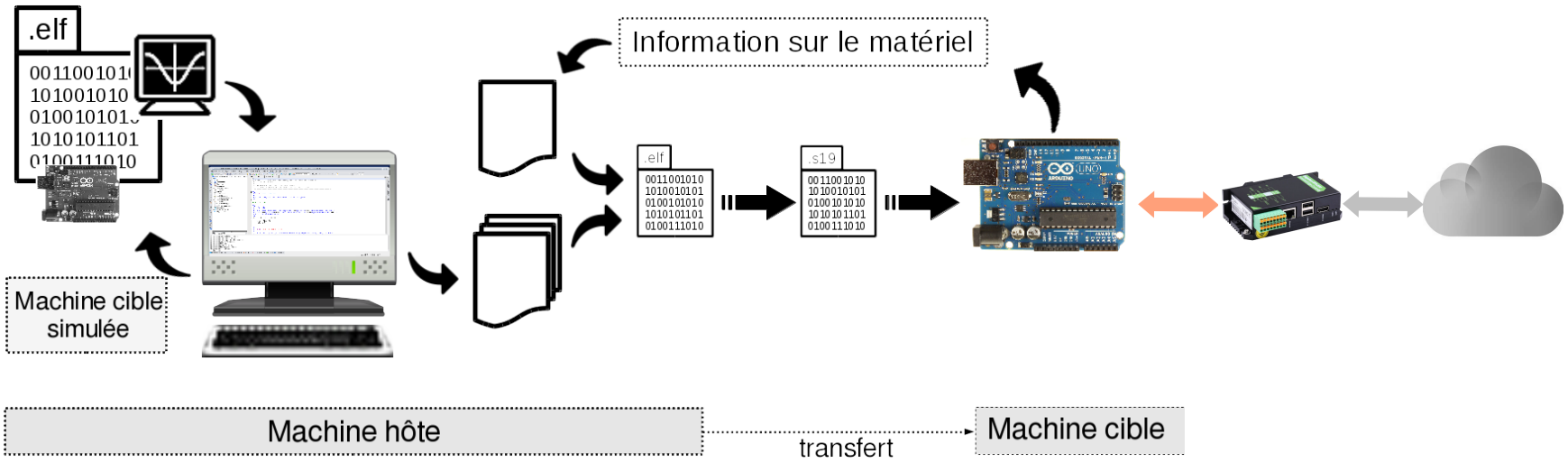
- Méthode de validation (fonction de la criticité)
  - Utilisation de méthodes formelles
  - Tests (critères de couverture)

Le développement de systèmes temps réel version simple :-)



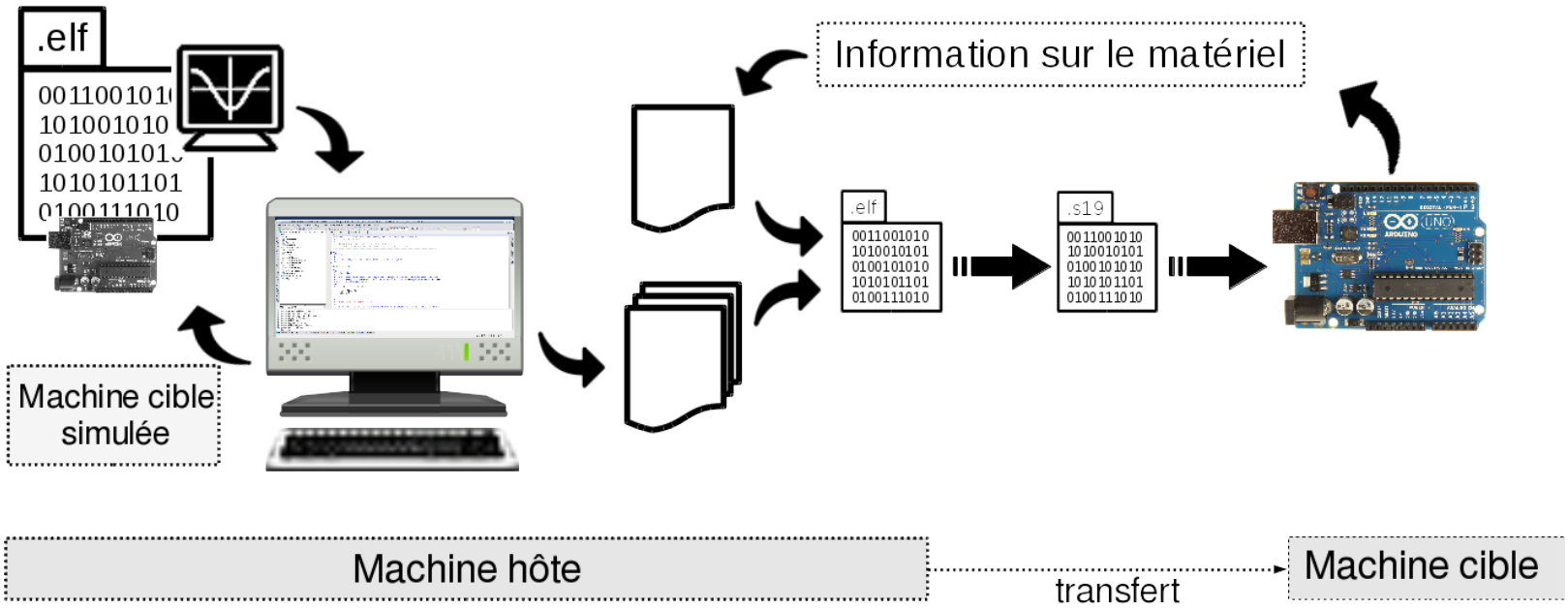
- Méthode de validation (fonction de la criticité)
  - Utilisation de méthodes formelles
  - Tests (critères de couverture)
  - Simulation fonctionnelle (exhaustive ou non) sur machine hôte

Le développement de systèmes temps réel version simple :-)



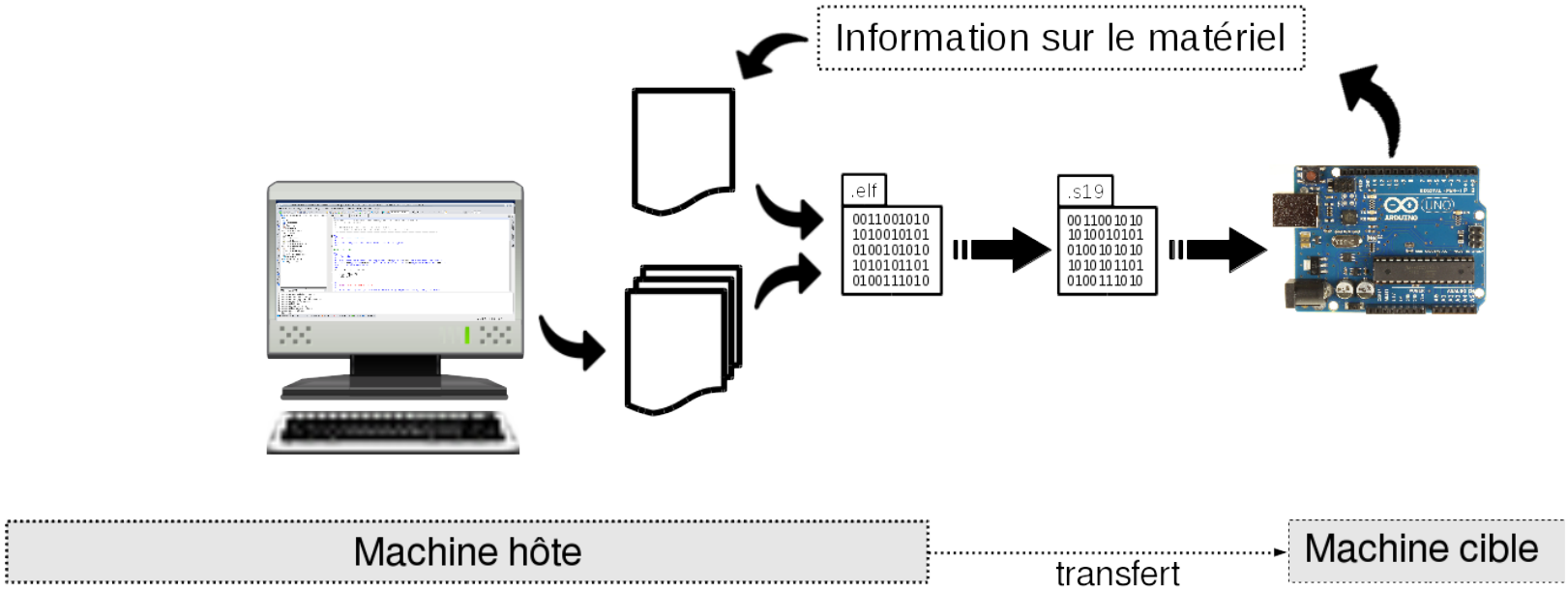
- Méthode de validation (fonction de la criticité)
  - Utilisation de méthodes formelles
  - Tests (critères de couverture)
  - Simulation fonctionnelle (exhaustive ou non) sur machine hôte
  - Digital Twin ?!

Le développement de systèmes temps réel version simple :-)



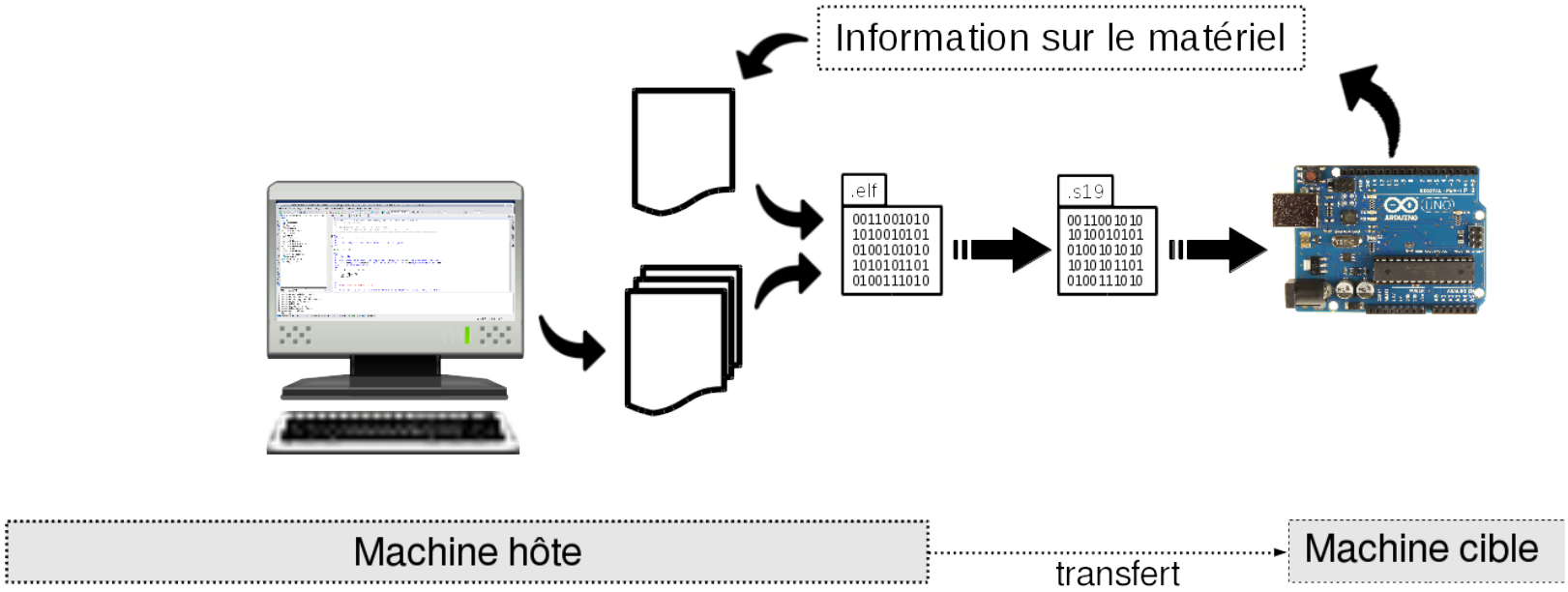
- Critères de validation
  - Temporelle
  - Énergétique
  - Empreinte mémoire
  - ...

Le développement de systèmes temps réel version simple :-)



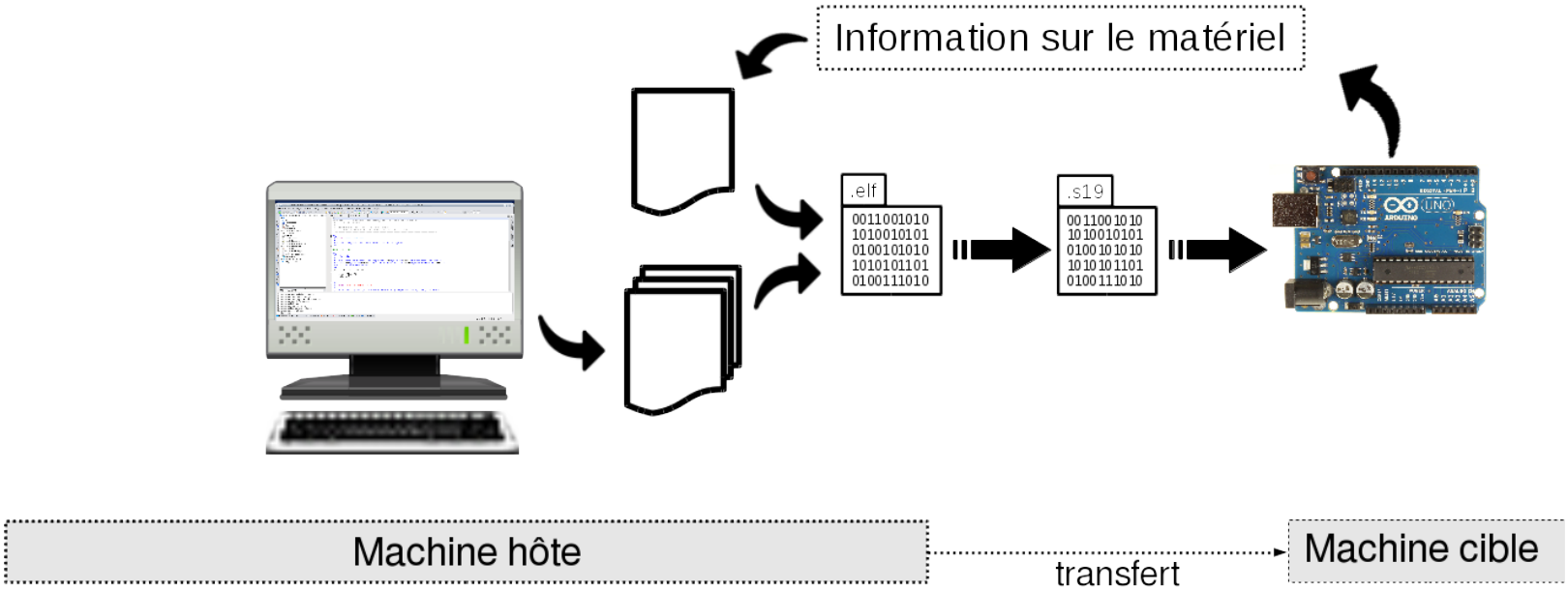
- Cross compilation (compilation croisée)
  - Jeu d'instruction spécifique
  - Configuration matérielle spécifique
    - En particulier mapping mémoire
  - Avec information de débuggage (elf) ou non (s19)

Le développement de systèmes temps réel version simple :-)



- Cross compilation et validation ?
  - Re-validation du code généré
  - Compilateurs certifiés
    - Répondent à des critères stricts de génération

Le développement de systèmes temps réel version simple :-)



- Transfert (jtag / SPI / ...)
  - Simple upload
  - Transfert et pilotage
    - Permet le debug
    - Mais l'environnement et le process ?

Contenu du cours

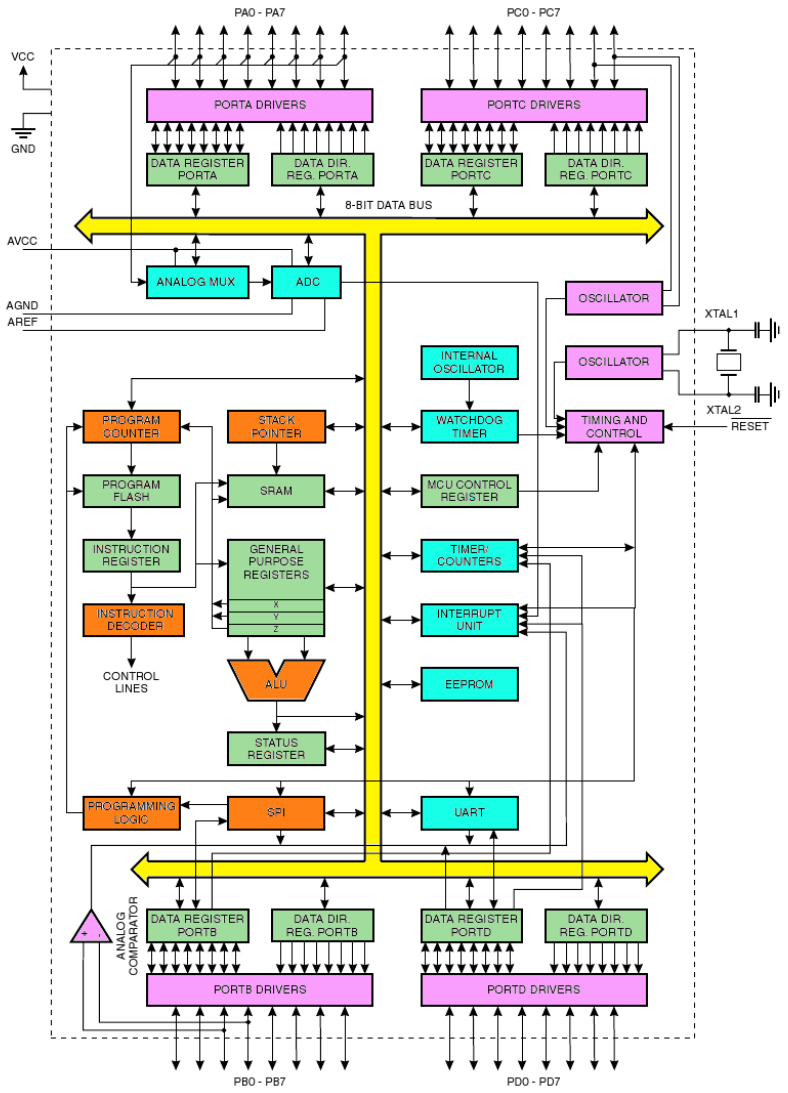
- Généralités
  - Les sytèmes considérés
  - Le développement de systèmes à base de micro contrôleurs
- Programmation sans OS
  - **Micro-contrôleur** sans OS, pourquoi ? comment ?
  - Stratégie d'implémentation
    - Programmation sans IT (Synchrone)
    - Programmation avec IT (Asynchrone)
- Mise en oeuvre
- Programmation avec un OS temps réel...

Micro-contrôleur ?

- Processeur
- Mémoire
- Périphériques
- Bus de communication
- Entrées / Sorties

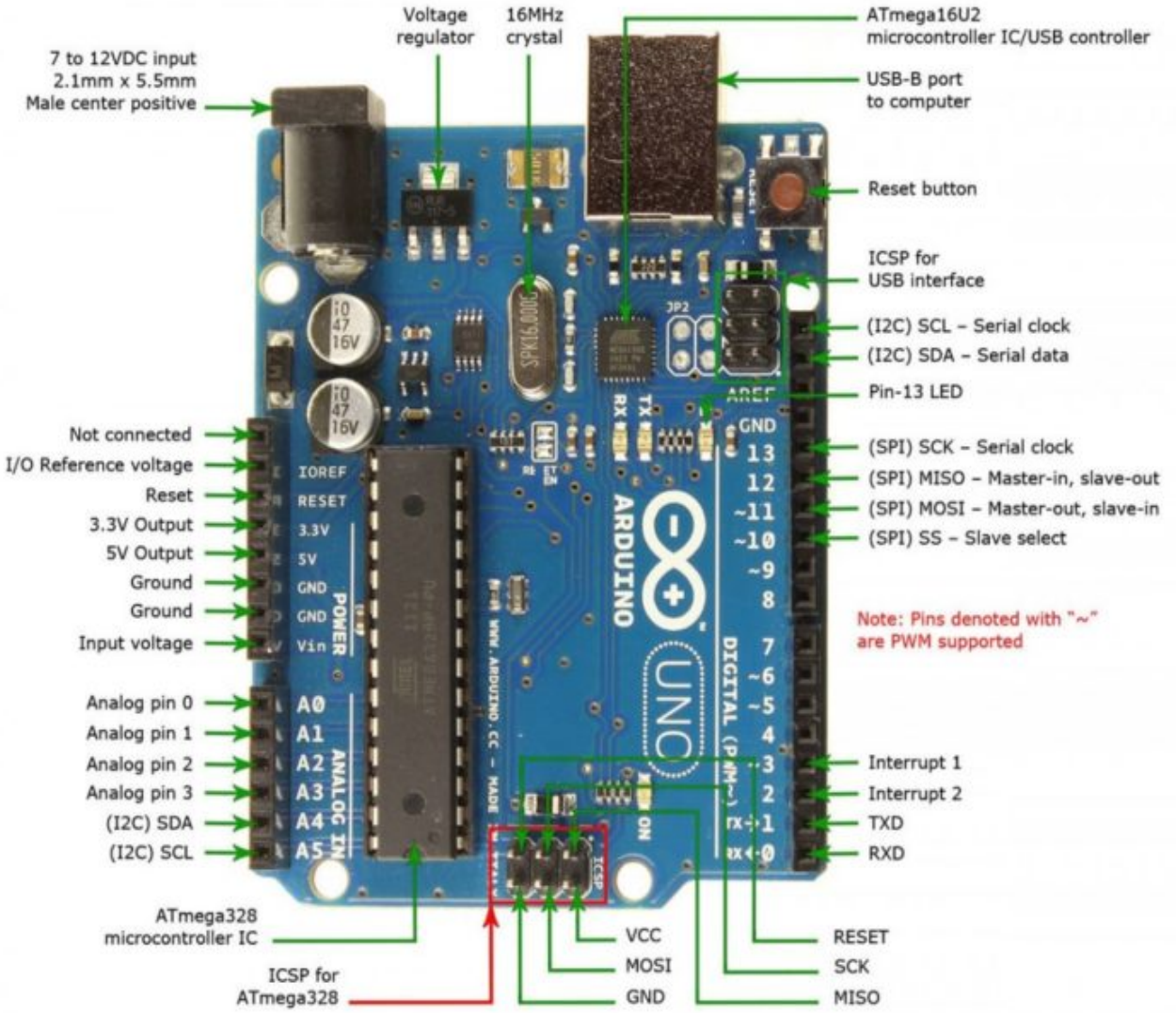
→ **Dans le même boitier**

Figure 1. The AT90S8535 Block Diagram (From Atmel DataSheet)



AT90S8535

Matériel utilisé dans ce cours



Contenu du cours

- Généralités
  - Les systèmes considérés
  - Le développement de systèmes à base de micro contrôleur
- Programmation sans OS
  - **Micro-contrôleur sans OS, pourquoi ?** comment ?
  - Stratégie d'implémentation
    - Programmation sans IT (Synchrone)
    - Programmation avec IT (Asynchrone)
- Mise en oeuvre
- Programmation avec un OS temps réel...

Micro-contrôleur sans OS : pourquoi ?

- Un OS prend de la place et du temps de calcul
  - ➔ Perte financière (nombre d'exemplaire ?)
- Un OS complique la conception / validation
  - L'OS prend la main ?
  - Combien de temps ?
  - Quand mon code est-il exécuté ?
  - Est-il interrompu ?
- L'OS utilise-t'il toutes les possibilités matérielles ?
  - Ex.: le ATmega328p a 5 modes de mise en veille

Micro-contrôleur sans OS : pourquoi ?

	Active Clock Domains					Oscillators		Wake-Up Sources							
Sleep Mode	clk CPU	clk FLASH	clk IO	clk ADC	clk ASY	Main Clock Source Enabled	Timer Oscillator Enabled	INT and PCINT	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X	X	X(2)	X	X	X		
Power-Down								X	X				X		X
Power-Save					X		X(2)	X	X	X			X		X
Standby(1)						X		X	X				X		X
Extended Standby					X(2)	X	X(2)	X	X	X			X		X

Note:

1. Only recommended with an external crystal or resonator selected as the clock source.
2. If Timer/Counter2 is running in Asynchronous mode.

- L'OS utilise-t'il toutes les possibilités matérielles ?
  - Ex.: le ATmega328p a 6 modes de mise en veille

Micro-contrôleur sans OS : pourquoi ?

- Un OS n'est pas disponible !
- Vous êtes en train de développer un OS

Micro-contrôleur sans OS : pourquoi ?

- Sans OS
  - Programmation mono-tâche
  - Prédictibilité forte
  - Programmation proche du matériel
    - Optimisation possible
    - Configuration fine et adaptée
  - Gain de place
  - Gain de performance

Contenu du cours

- Généralités
  - Les sytèmes considérés
  - Le développement de systèmes à base de micro contrôleur
- Programmation sans OS
  - **Micro-contrôleur sans OS, pourquoi ? comment ?**
  - Stratégie d'implémentation
    - Programmation sans IT (Synchrone)
    - Programmation avec IT (Asynchrone)
- Mise en oeuvre
- Programmation avec un OS temps réel...

Micro-contrôleur sans OS : comment ?

- Mettre en place votre environnement de développement
  - Choisir un langage de développement
    - Assembleur
    - C / C++
    - Ada
    - Rust
    - ...
  - Trouver / choisir un compilateur adapté
  - Trouver un *linker* (pour faire le transfert)
  - Trouver / choisir un simulateur si disponible
  - Se procurer les *datasheets* du micro-contrôleur

Micro-contrôleur sans OS : comment ?

- Les datasheets en quelques mots
  - C'est la documentation du micro-contrôleur
  - Sont généralement très conséquentes (567 pages pour le micro contrôleur des cartes arduino !)
  - Ne contiennent pas que des choses utiles pour les informaticiens
  - Réverbatif si on ne sait pas ce que l'on cherche

Ce n'est pas un roman donc à moins de vouloir devenir expert des moindres fonctionnalités, ne le lisez pas séquentiellement

Micro-contrôleur sans OS : comment ?

- Informations importantes des datasheets
  - Les ports d'Entrées / Sorties (et Brochages des pattes physiques)
  - La description des périphériques intégrés
    - Timer ? Convertisseurs A/D ? liaison série ?
  - L'organisation de la mémoire
    - Intégrée ou non
  - Les registres

- Informations moins importantes
  - Caractéristiques éléctriques (sauf les consos dans les différents mode de veille)
  - Jeux d'instructions assembleur (a moins que...)

Micro-contrôleur sans OS : comment ?

- Les ports d'Entrées / Sorties
  - Permettent de communiquer avec l'électronique de la machine (notamment les capteurs et les actionneurs)
  - Plus ou moins nombreux
  - Multi fonctionnalités
  - Multi Directionnels

Table 6-1. Input/Output Ports

Port	Input Pins	Output Pins	Bidirectional Pins	Shared Functions
Port A	3	3	2	Timer
Port B	—	8	—	High-order address
Port C	—	—	8	Low-order address and data bus
Port D	—	—	6	Serial communications interface (SCI) and serial peripheral interface (SPI)
Port E	8	—	—	Analog-to-digital (A/D) converter

68HC11

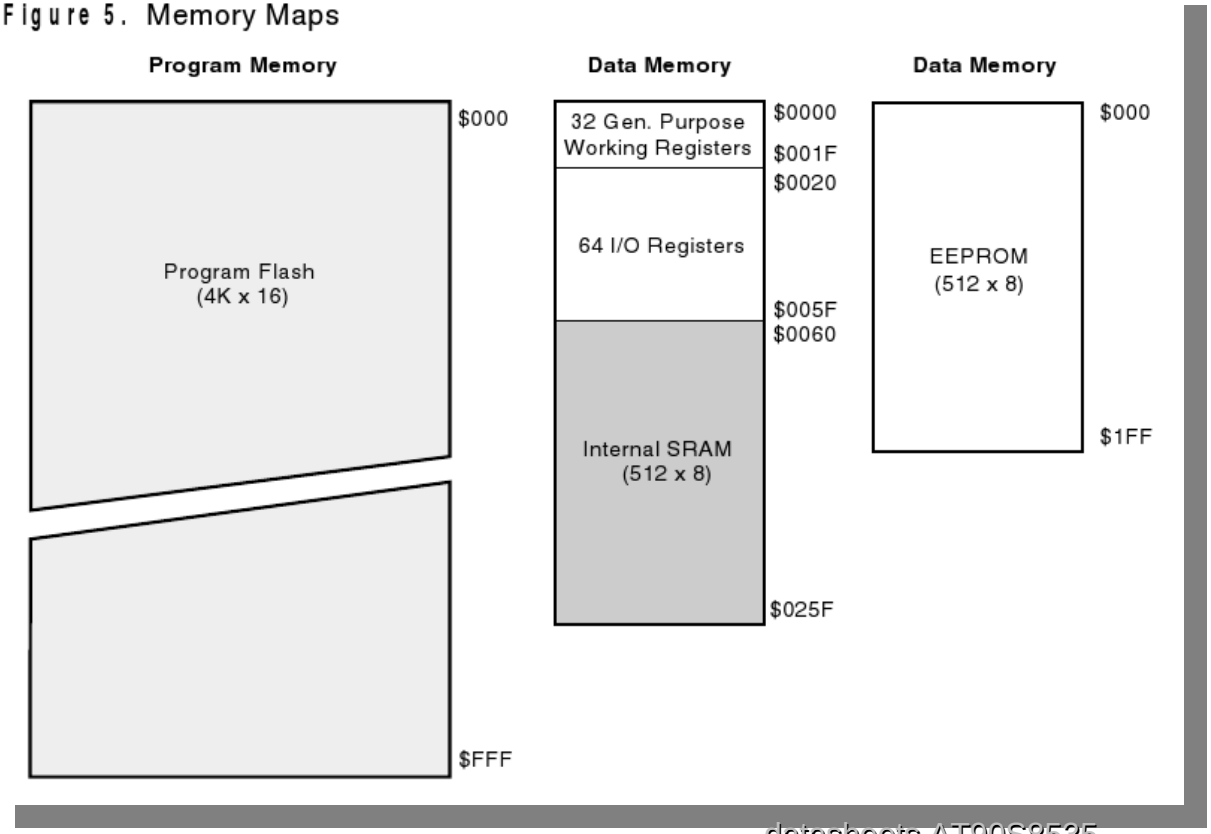
Micro-contrôleur sans OS : comment ?

- Les périphériques classiques
  - Timer
    - Permettent de mesurer le temps physique
    - Sont liés à la fréquence physique d'oscillation
    - Programmables et sources d'interruptions ou non
  - Liaison série
    - SPI (Serial Peripheral Interface) – JTAG
    - UART (universal asynchronous receiver transmitter)– RS232
    - ...
    - ➔ Permet souvent le 'transfert' et le debuggage "*in situ*"
  - Convertisseur Analogique numérique
    - Interface avec des capteurs analogiques

Micro-contrôleur sans OS : la mémoire

- L'organisation de la mémoire ; un exemple

Figure 5. Memory Maps



The RAM is mapped to $0000 after reset.

datasheets AT90S8535

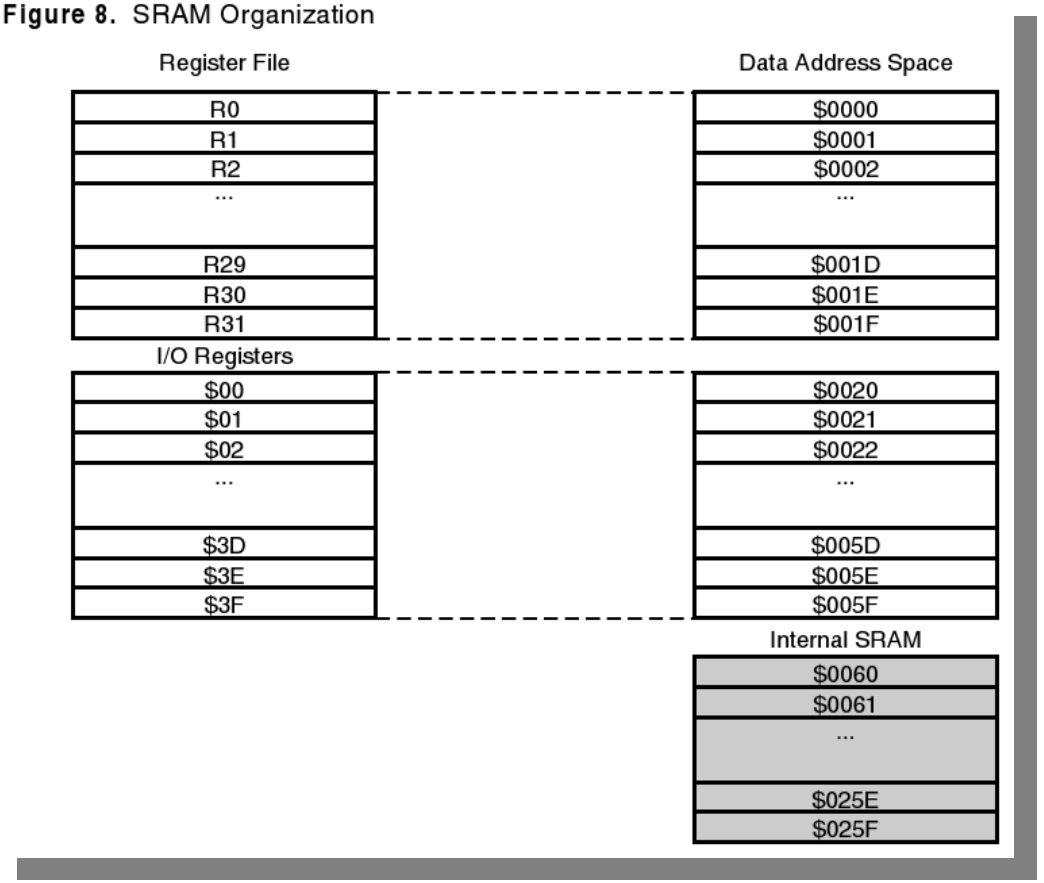
Micro-contrôleur sans OS : la mémoire

- Choisir son mapping
  - Une place pour le programme
    - En RAM pour les premiers tests
    - En EEPROM (ou Flash) pour les tests plus poussés
    - En ROM une fois en production
  - Une place pour les variables
    - En RAM bien sûr
  - Une place pour la pile
    - En RAM mais pas n'importe où (début ou fin ?)
      - Dépend des micro-contrôleurs (implémentation matérielle de push / pop)

Micro-contrôleur sans OS : les registres

- Stockés en RAM
- 2 types
  - Configuration
  - Utilisateurs
- *Peuvent être redondants*

Figure 8. SRAM Organization



Datasheets AT90S8535

Micro-contrôleur sans OS : les registres

- Les registres de configuration
  - Souvent nombreux
  - 8 / 16 / 32 bits
  - À une adresse mémoire particulière
  - À considérer bit par bit

TABLE 4-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

bit 4 **INTE**: RB0/INT External Interrupt Enable bit

1 = Enables the RB0/INT external interrupt

0 = Disables the RB0/INT external interrupt

Datasheets PIC16F84A

Micro-contrôleur sans OS : les registres

- Les registres 'utilisateur'
  - 8 / 16 / 32 bits
  - À une adresse mémoire particulière
  - Permettent de stocker les données sur lesquelles travailler (~ variables)
  - Sont utilisés par les instructions de calcul assembleurs

Contenu du cours

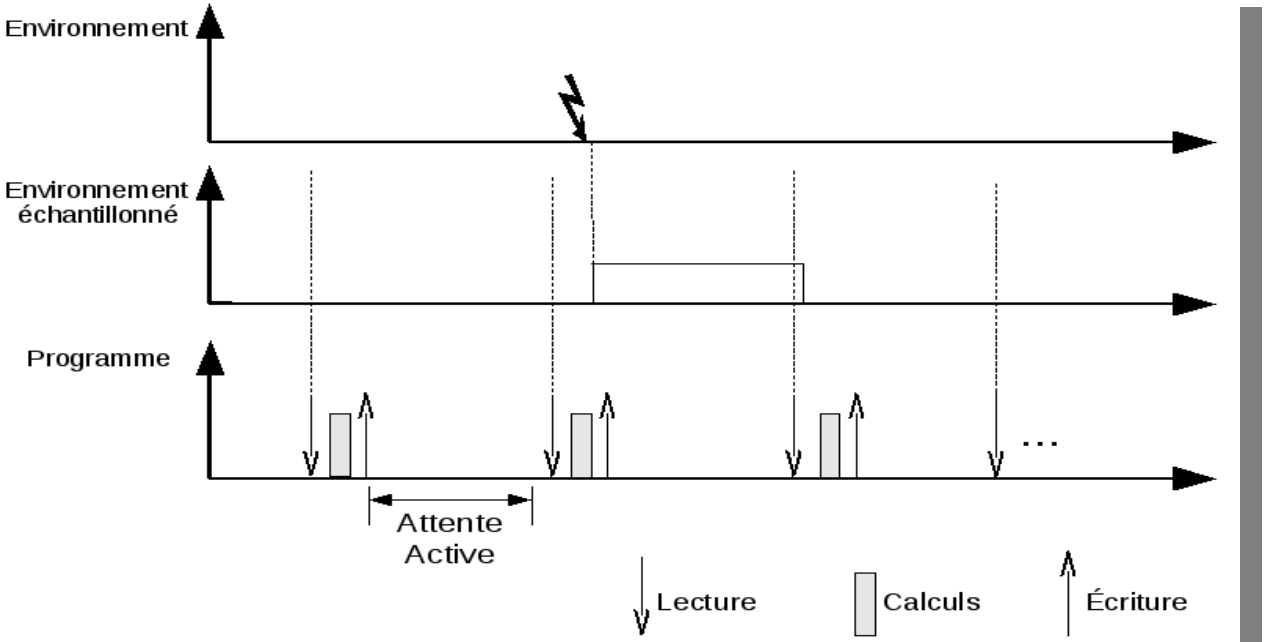
- Généralités
  - Les sytèmes considérés
  - Le développement de systèmes à base de micro contrôleur
- Programmation sans OS
  - Micro-contrôleur sans OS, pourquoi ? comment ?
  - **Stratégie d'implémentation**
    - **Programmation sans IT (Synchrone)**
    - Programmation avec IT (Asynchrone)
- Mise en oeuvre
- Programmation avec un OS temps réel...

Micro-contrôleur sans OS : sans IT

- Stratégie d'implémentation
  1) Sans interruption (Synchrone)
     - Boucle infinie
       - Lecture de tous les capteurs + état système (scrutation)
       - Calculs
       - Écriture actionneurs + état système
       - (attente)

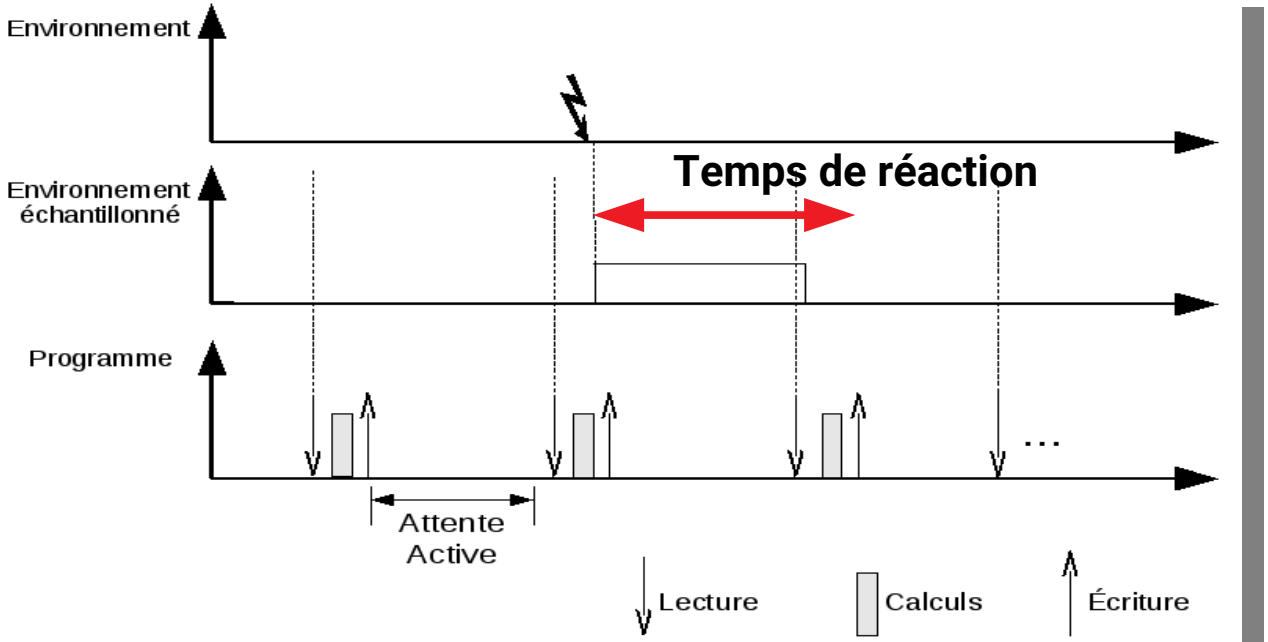
Micro-contrôleur sans OS : sans IT

- Boucle infinie
  - Lecture de tous les capteurs + état système (scrutation)
  - Calculs
  - Écriture actionneurs + état système
  - (attente)



Micro-contrôleur sans OS : sans IT

- Boucle infinie
  - Lecture de tous les capteurs + état système (scrutation)
  - Calculs
  - Écriture actionneurs + état système
  - (attente)



Micro-contrôleur sans OS : sans IT

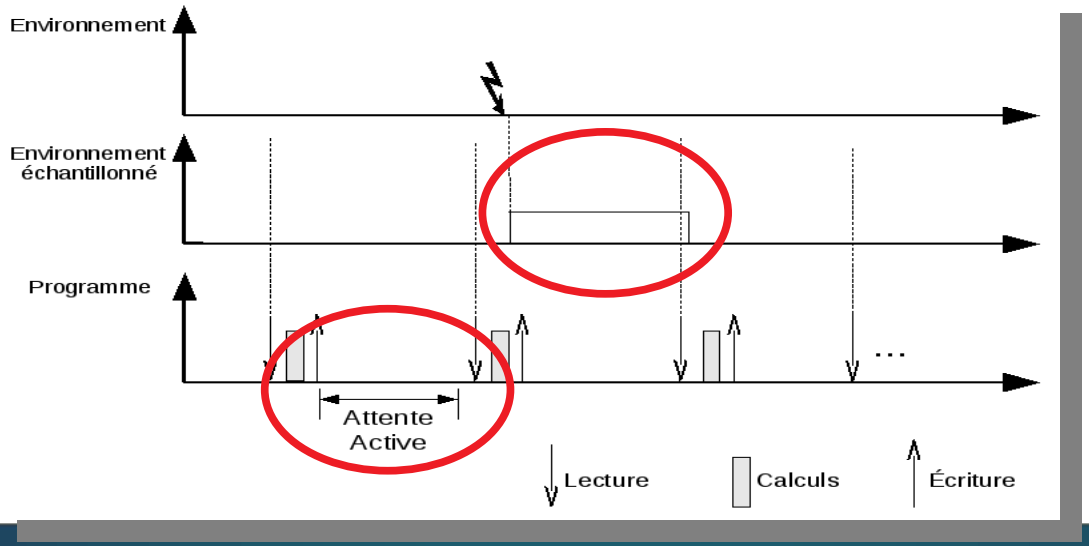
- Stratégie d'implémentation

  1) Sans interruption (Synchrone)

  - Avantages :
    - Très simple à mettre en oeuvre
    - Calculs des temps de réactions simplifiés
    - Pas / peu de gestion de la pile

Micro-contrôleur sans OS : sans IT

- Stratégie d'implémentation
  1) Sans interruption (Synchrone)
     - Inconvénients :
       - Peu réactif, dépend beaucoup de la durée des calculs
       - Difficile de gérer la consommation électrique (Attente active)

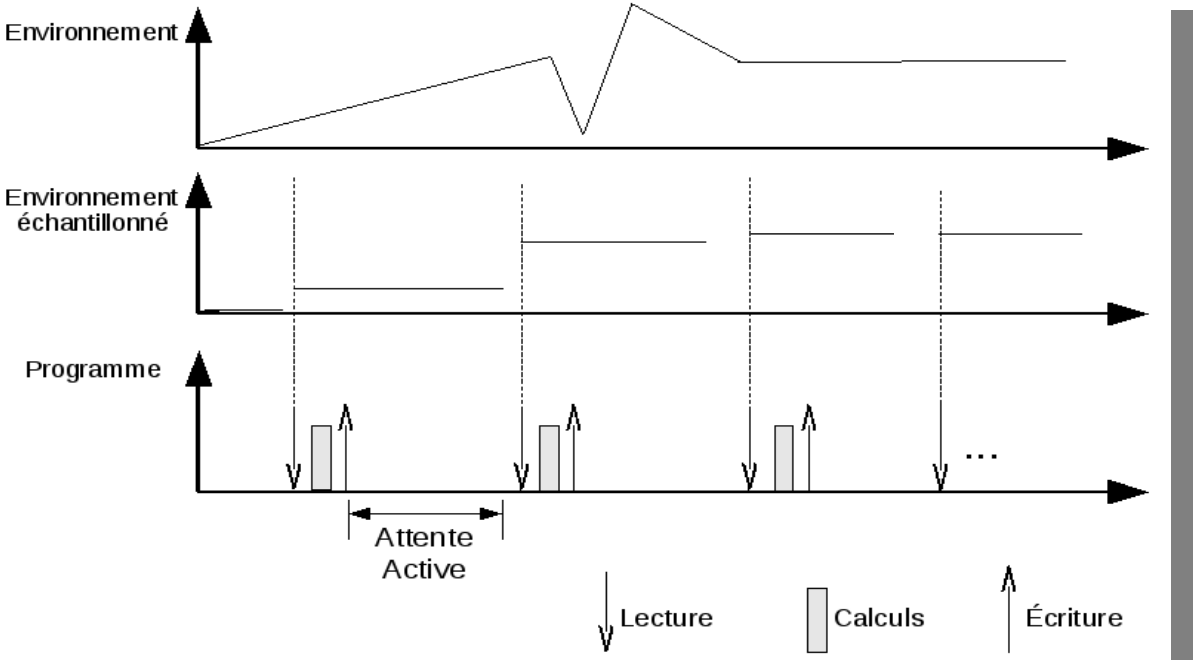


Micro-contrôleur sans OS : sans IT

- Stratégie d'implémentation

1) Sans interruption (Synchrone)

- Inconvénients :
  - Peu réactif, dépend beaucoup de la durée des calculs
  - Difficile de gérer la consommation électrique (Attente active)

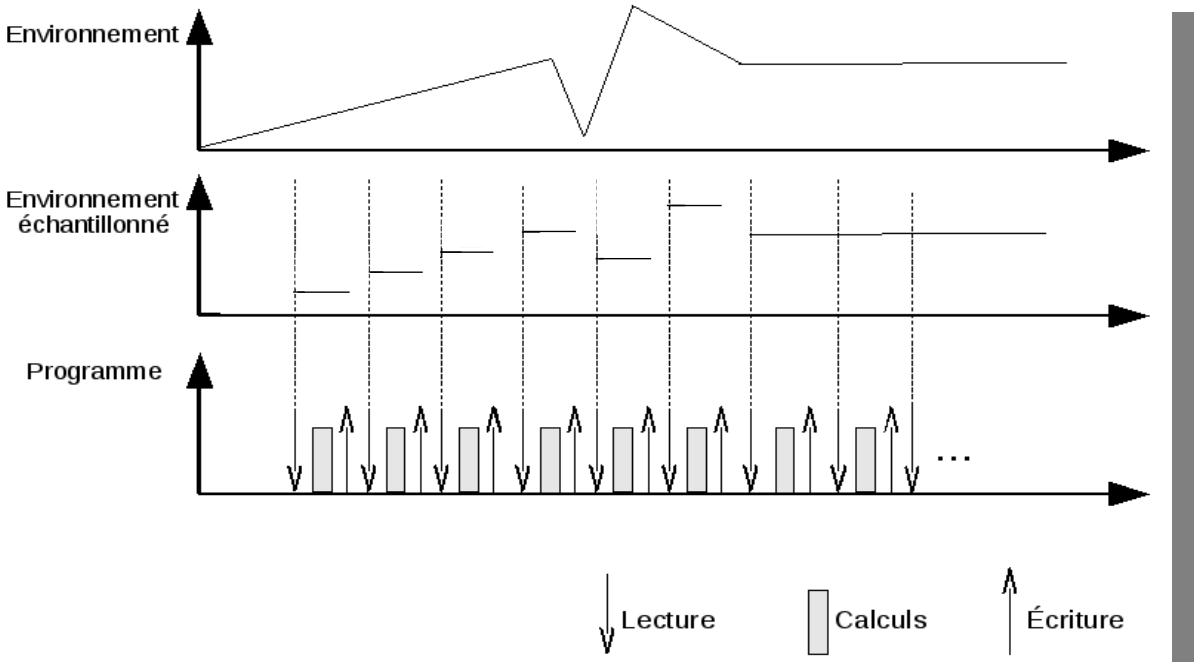


Micro-contrôleur sans OS : sans IT

- Stratégie d'implémentation

1) Sans interruption (Synchrone)

- Inconvénients :
  - Peu réactif, dépend beaucoup de la durée des calculs
  - Difficile de gérer la consommation électrique (Attente active)



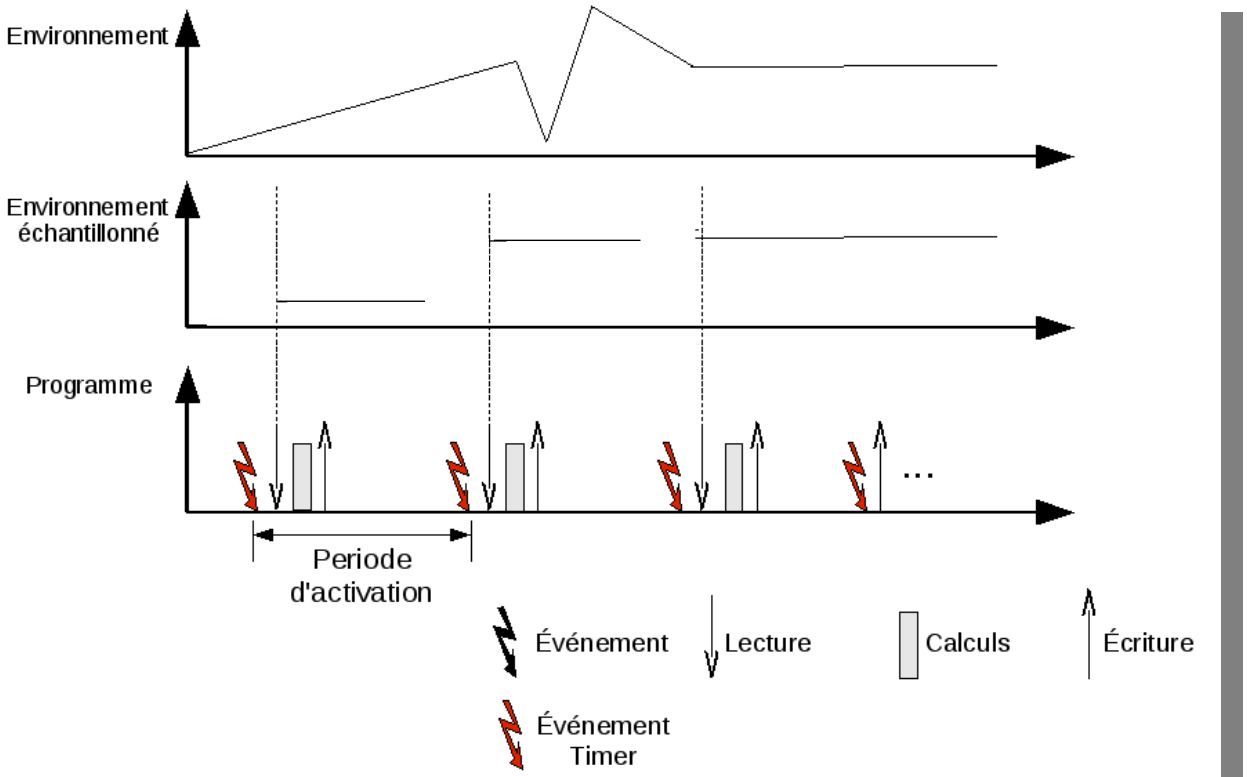
Micro-contrôleur sans OS : avec IT

- Choisir sa stratégie d'implémentation

2) Avec interruption (asynchrone)
   - Idem que sans interruption +
   - Traitement urgence
   - Changement d'états
   - Attente non active

Micro-contrôleur sans OS : avec IT

- Idem que sans interruption +
- Traitement urgence
- Changement d'états
- Attente non active



Micro-contrôleur sans OS : avec IT

- Choisir sa stratégie d'implémentation

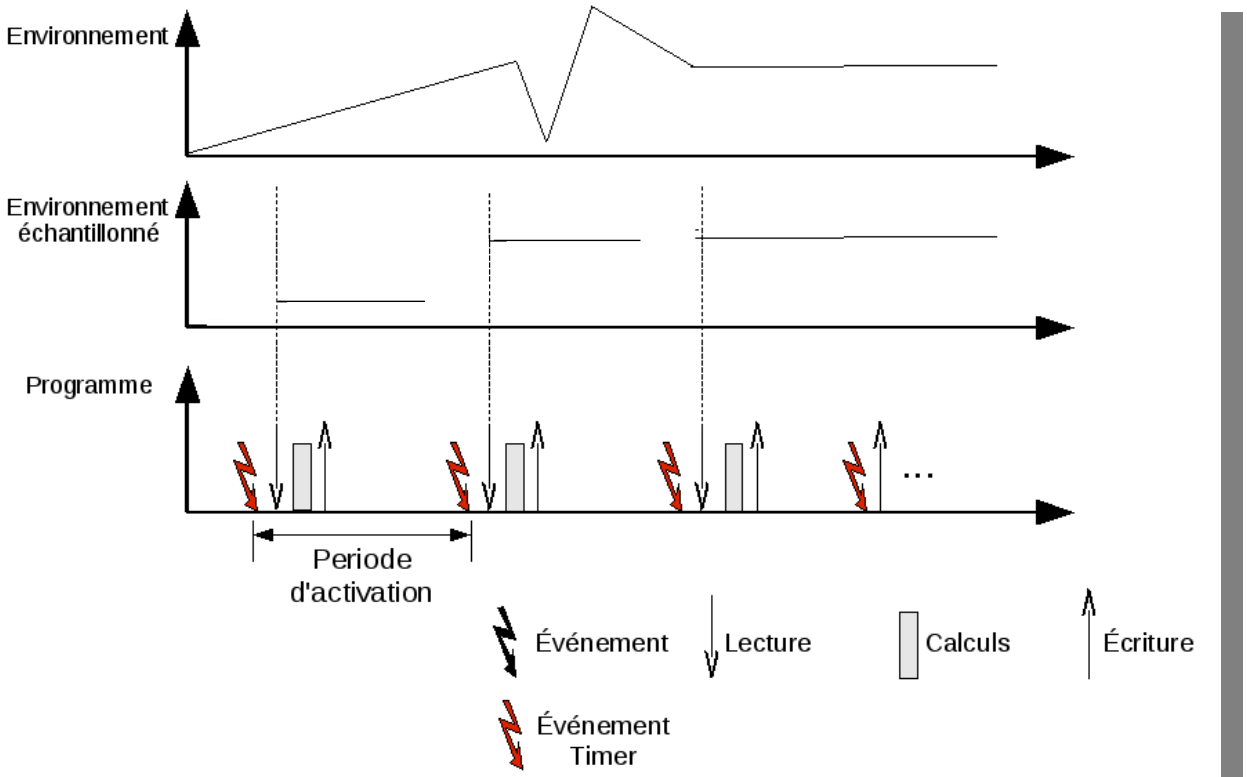
2) Avec interruption
   - Idem que sans interruption +
   - Traitement urgence
   - Changement d'états
   - Attente non active

- Avantages :
  - Purement réactif
  - Permet l'économie d'énergie
  - Pas de scrutation non nécessaire

Micro-contrôleur sans OS : avec IT

2) Avec interruption

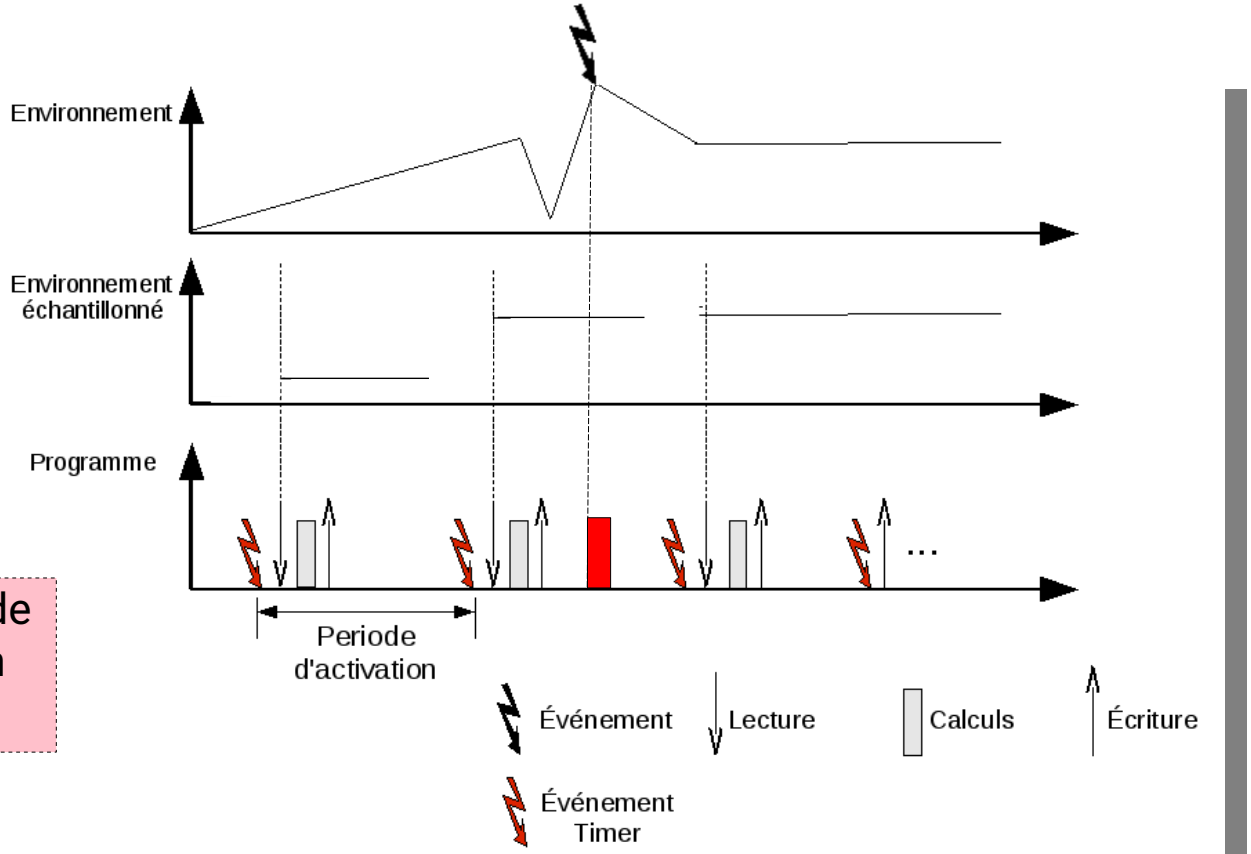
- Avantages :
  - Purement réactif



Micro-contrôleur sans OS : avec IT

2) Avec interruption

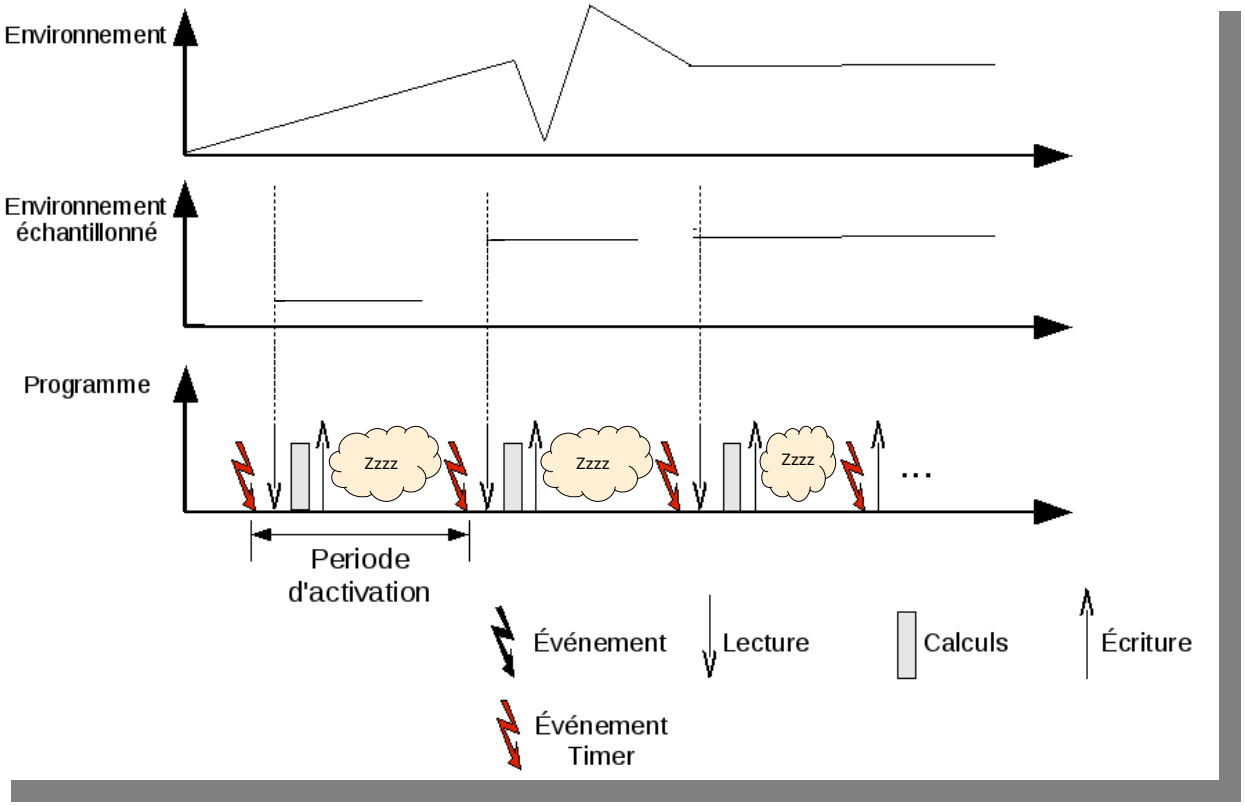
- Avantages :
  - Purement réactif



La fontion exécutée lors de l'arrivée d'une interruption est appelée ***handler***

2) Avec interruption

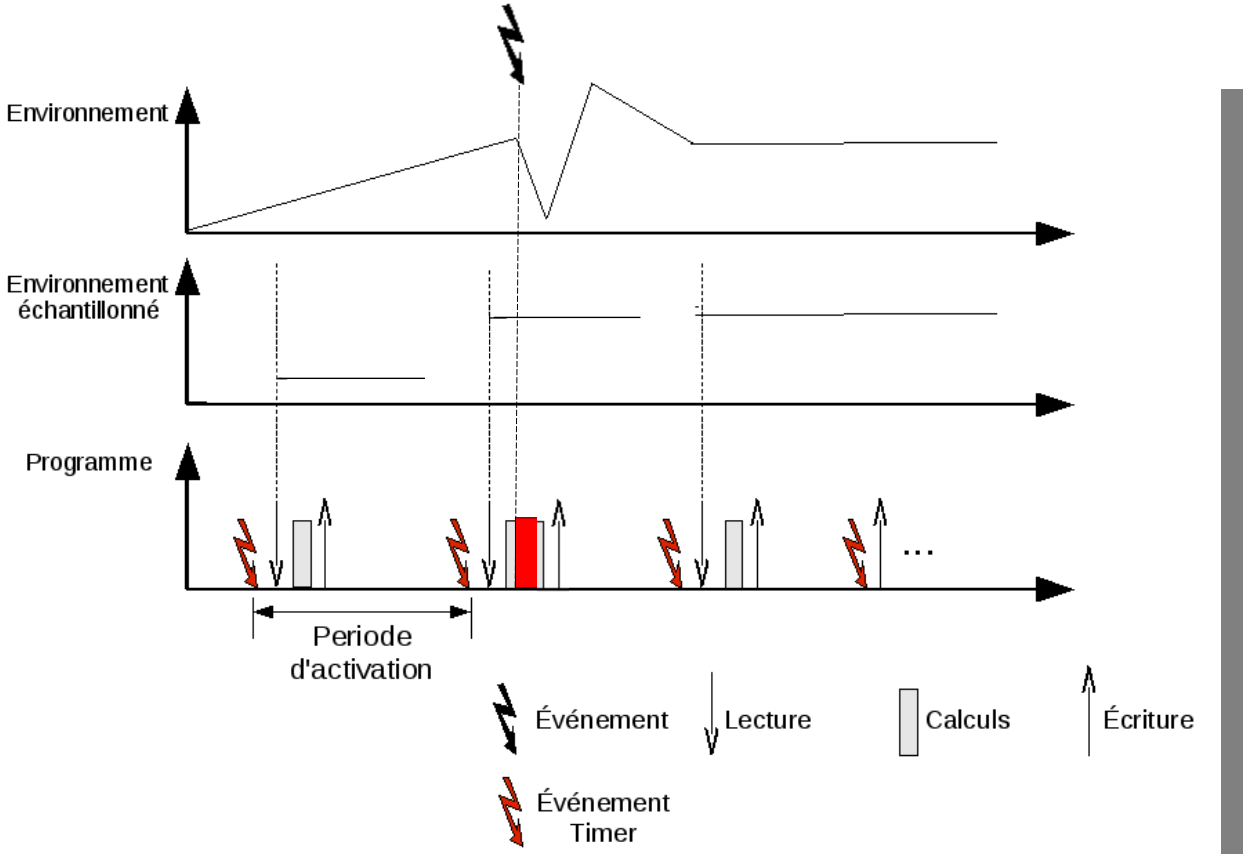
- Avantages :
  - Permet l'économie d'énergie



Micro-contrôleur sans OS : avec IT

2) Avec interruption

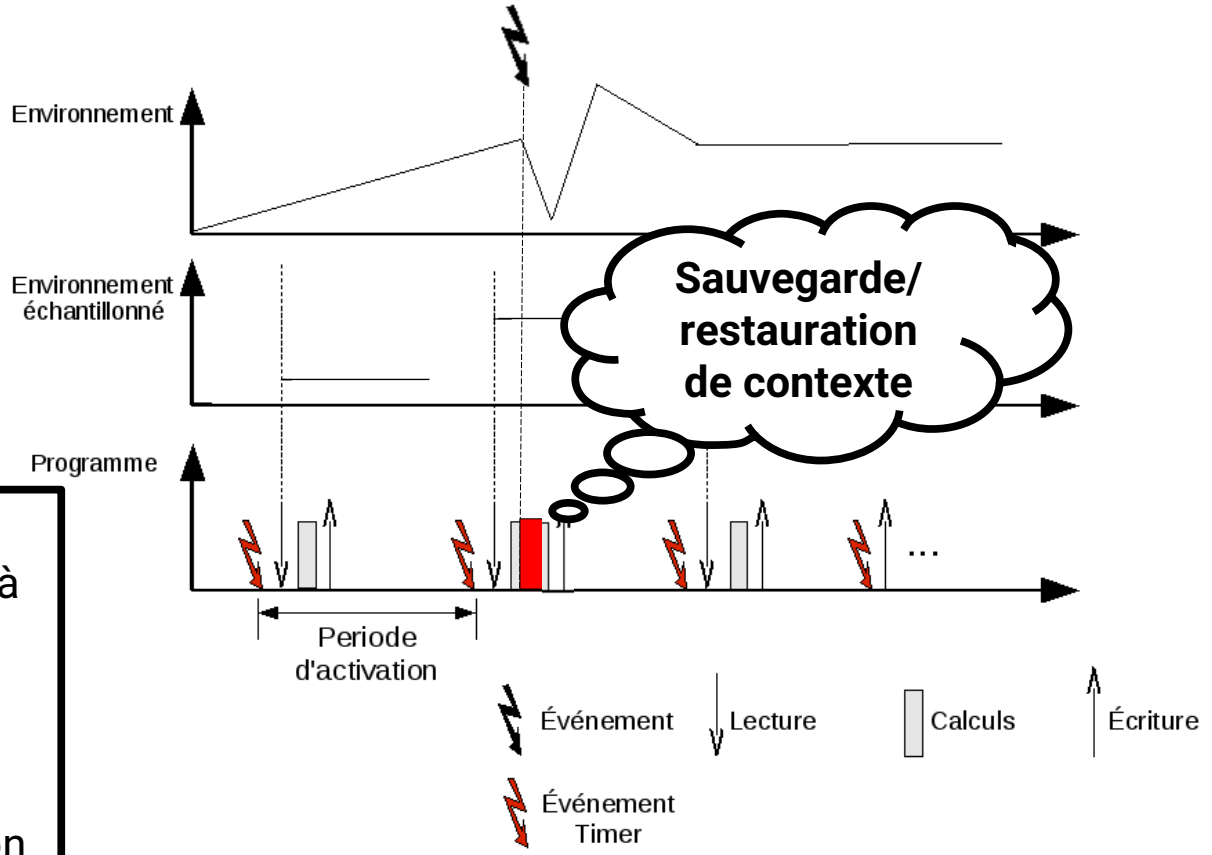
- Inconvénient :
  - Demande un peu de technique



Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique



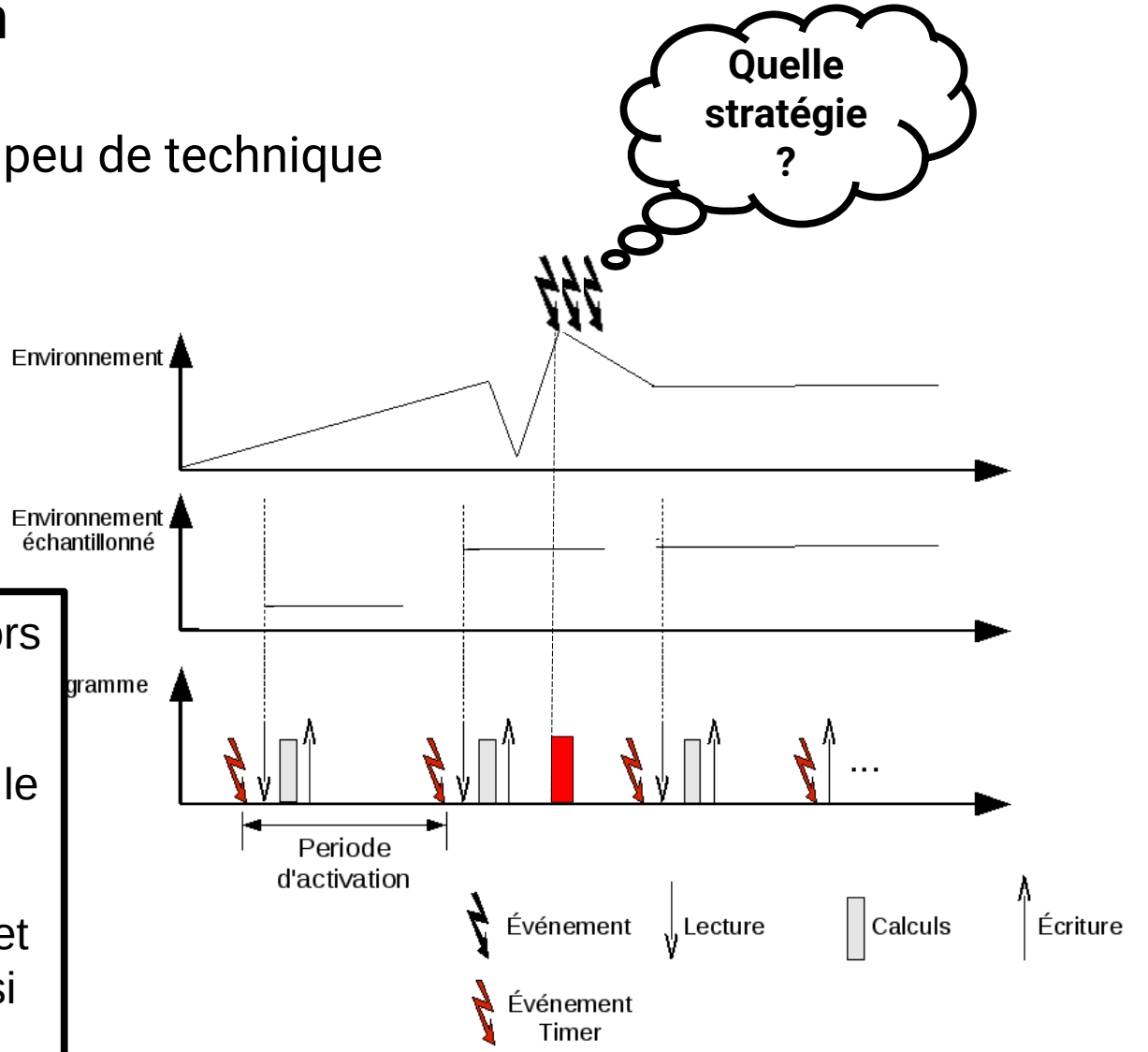
1.Mettre tous les valeurs des registres utilisateurs dans la pile à l'entrée dans l'interruption

2.Récupérer les valeurs des registres et les réassigner à la sortie du programme d'interruption

Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique



1.On ignore les interruptions lors du traitement d'interruption

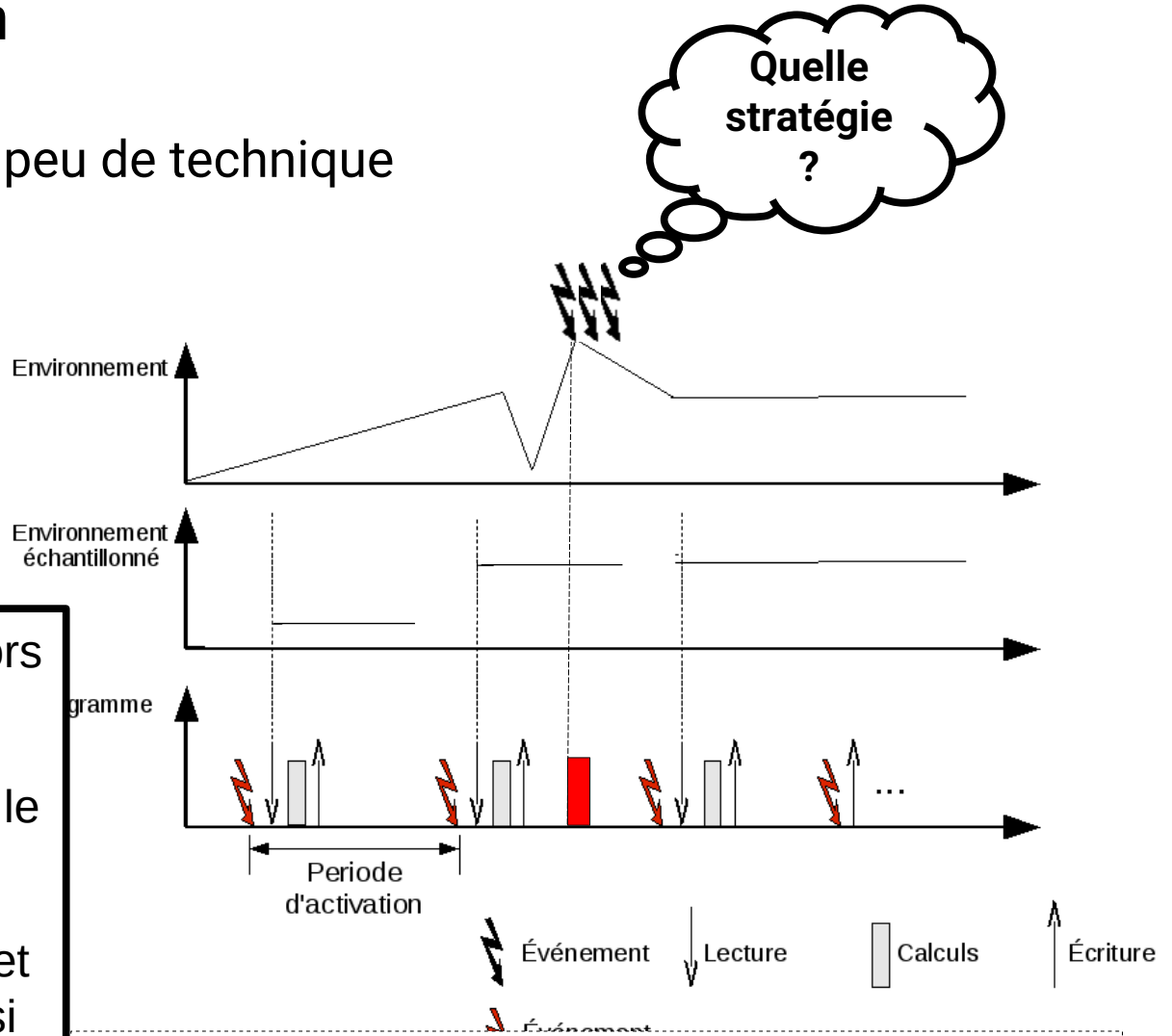
2.On masque les interruptions le temps du traitement

3.On compte les interruptions et on les traite successivement (si possible par le matériel)

Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique



1. On ignore les interruptions lors du traitement d'interruption

2. On masque les interruptions le temps du traitement

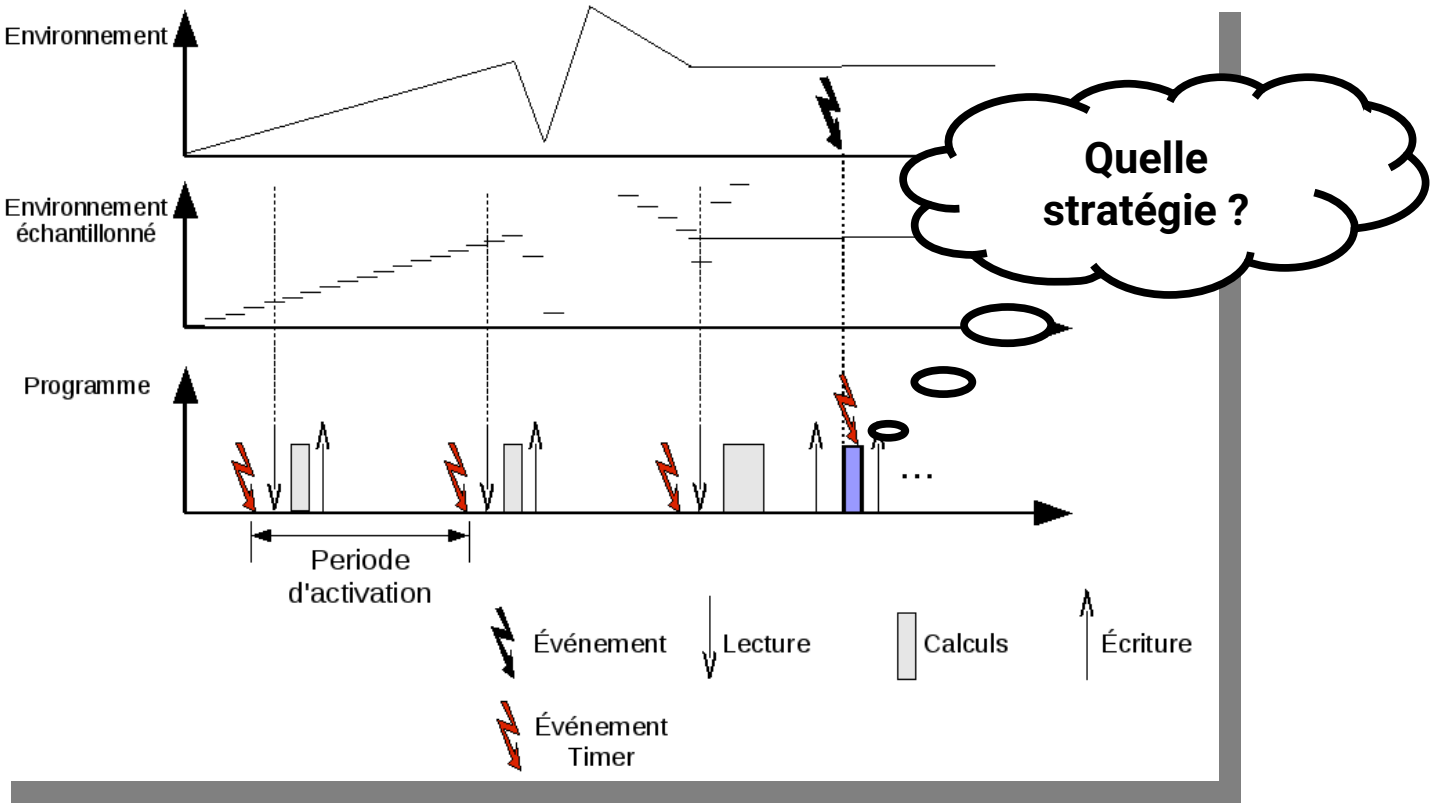
3. On compte les interruptions et on les traite successivement (si possible par le matériel)

➔ **Dépend de la sémantique de l'interruption**

Micro-contrôleur sans OS : avec IT

2) Avec interruption

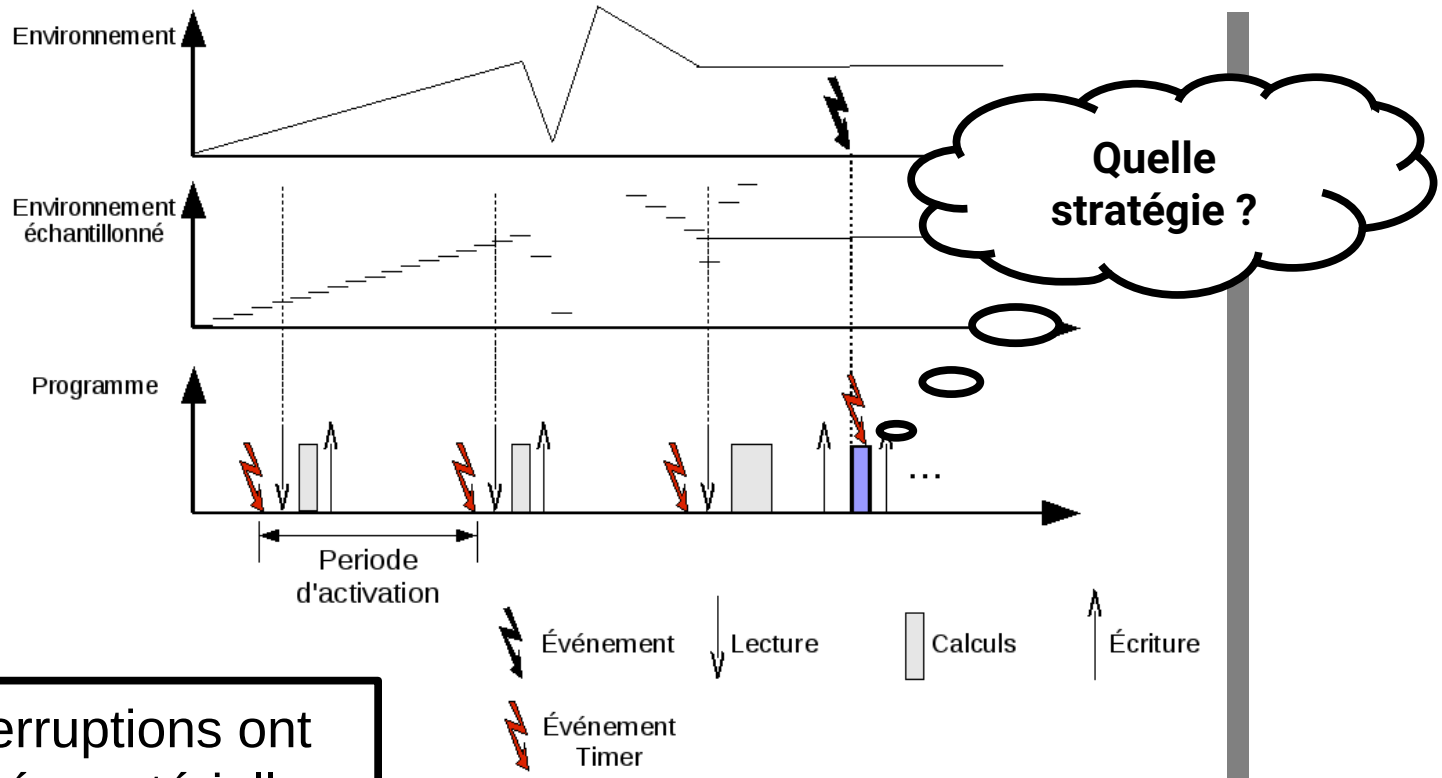
- Inconvénient :
  - Demande un peu de technique



Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique

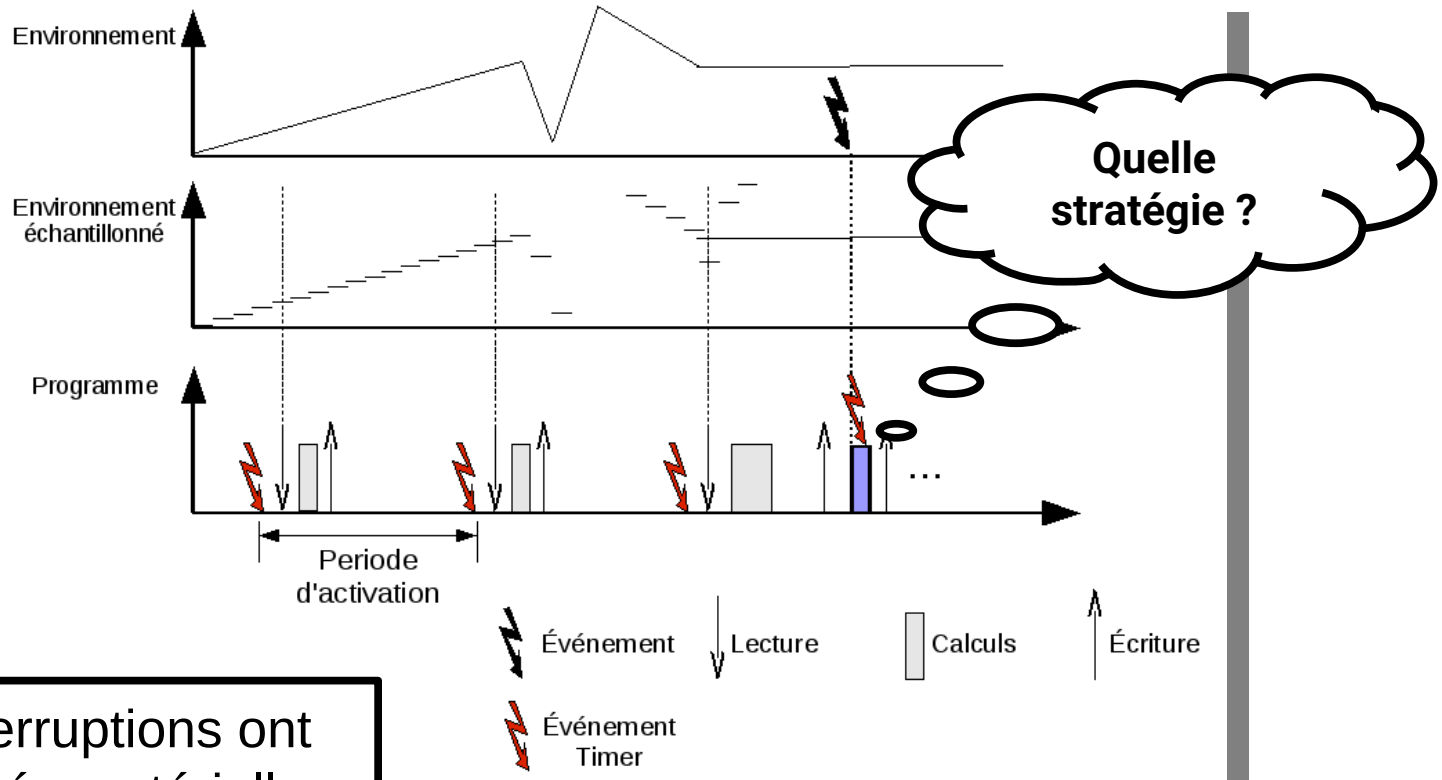


✔ Les interruptions ont des priorités matérielles non modifiables

Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique



✔ Les interruptions ont des priorités matérielles non modifiables

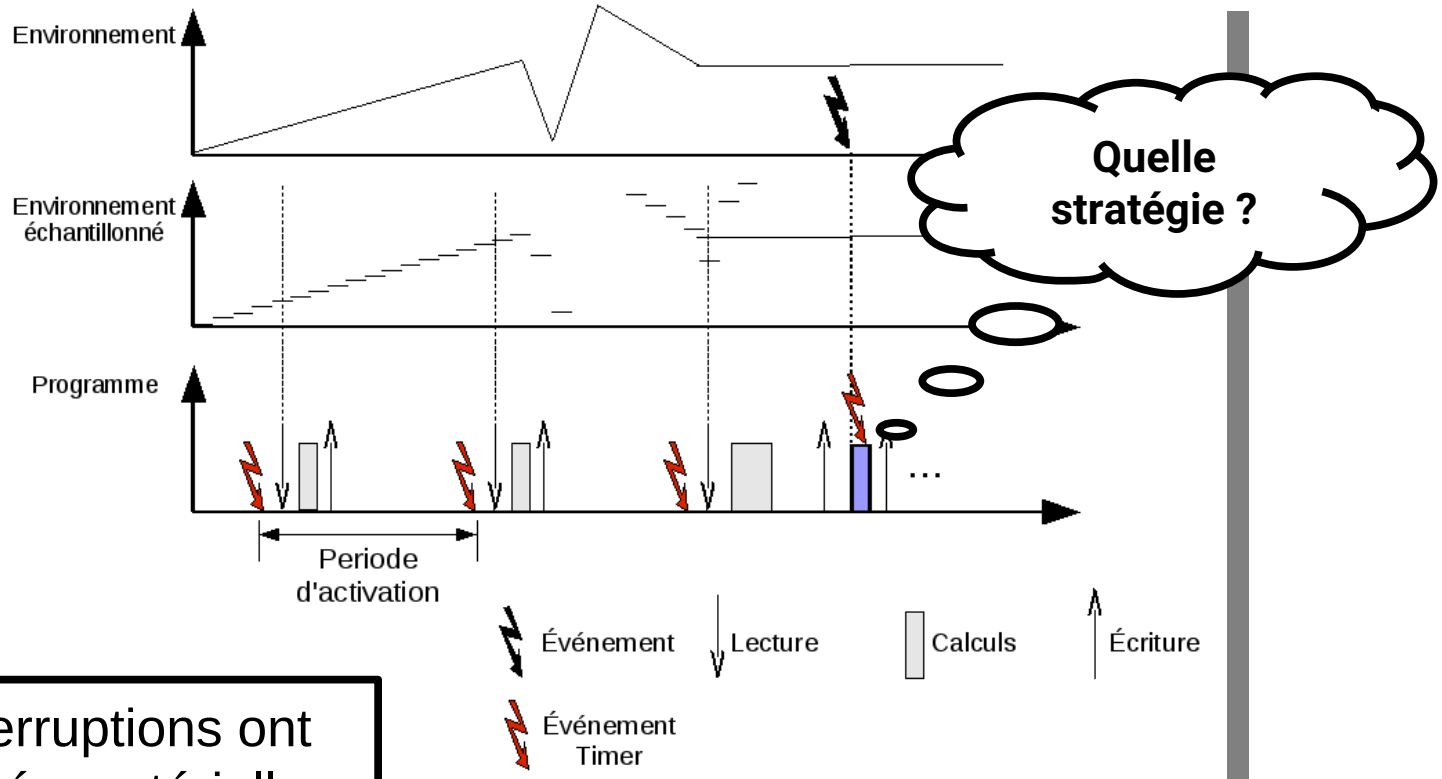
➔ **Dépend de la sémantique de l'interruption**

Micro-contrôleur sans OS : avec IT

2) Avec interruption

- Inconvénient :
  - Demande un peu de technique

Limite du sans OS ?



✔ Les interruptions ont des priorités matérielles non modifiables

➔ **Dépend de la sémantique de l'interruption**

Contenu du cours

- Généralités
  - Les sytèmes considérés
  - Le développement de systèmes à base de micro contrôleur
- Programmation sans OS
  - Micro-contrôleur sans OS, pourquoi ? comment ?
  - Stratégie d'implémentation
    - Programmation sans IT (Synchrone)
    - Programmation avec IT (Asynchrone)
- **Mise en oeuvre**
- Programmation avec un OS temps réel...

Mise en oeuvre: Le micro-contrôleur

- Atmel Atmega328P (16MHz)
  - Architecture RISC
  - 32Ko de Flash / 1Ko d'EEPROM, 2Ko de RAM
  - 23 entrées/sorties programmables
  - Communication UART, SPI et I2C
  - 2 timers 8 bits et 1 timer 16 bits
  - 6 modes de veilles
  - ... (voir datasheet)

Mise en oeuvre: La chaîne de compilation

- AVR-GCC and co
  - Cross compilateur basé sur gcc
  - Linker (avrudude) , librairies pour architecture spécifiques (avr-libc)
  - Windows / Linux

➔ https://github.com/m3y54m/start-avr?tab=readme-ov-file

Mise en oeuvre: La chaîne de compilation

- Définition des handlers différentes selon le compilateurs / linkers
  - AVR-gcc

```
//handler de l'interruption nommées INT_COMPA

ISR(INT_COMPA_vect)
{
    // traiter l'interruption
}
```

  - ICCAVR

```
//handler de l'interruption numéro NUM

#pragma nom_fonction_handler ISR:NUM

[...]

nom_fonction_handler
{
    // traiter l'interruption
}
```

Mise en oeuvre: Simuler

- SimulAVR
  - Simul le jeux d'instruction RISC AVR
  - Permet de visualiser l'état de la mémoire et des registres en pas à pas...ou pas
  - Windows / Linux

➔ https://reprap.org/wiki/SimulAVR

➔ http://savannah.nongnu.org/projects/simulavr/

- AVRStudio
  - Idem SimulAVR MAIS Solution professionnelle
  - Windows

Mise en oeuvre: *Linker* et transférer

- Avr-gcc (avr-objcopy)
  - Avr-gcc produit le mapping seul d'après le type de micro-contrôleur renseigné lors de la compilation
  - Avr-objcopy traduit le binaire (.elf) en format atmel , i.e. intel hexadecimal (.hex)

- AVRDude : https://github.com/avrdudes/avrdude
  - AVRDUDE - AVR Downloader Uploader - is a program for downloading and uploading the on-chip memories of Microchip's AVR microcontrollers.
  - For instance, to program an Arduino Uno connected to the serial port COM1 with a HEX file called blink.hex, you would run the following command:
    - avrdude -c arduino -P COM1 -b 115200 -p atmega328p -D -U flash:w:objs/blink.hex:i

Mise en oeuvre: Le code

- Code C classique
  - Arrêt des interruptions
  - **Configuration des registres**
  - Démarrage des interruptions
  - Boucle sans fin
- Forte utilisation des nombres hexadécimaux / binaire

Bit	7	6	5	4	3	2	1	0	
$1B ($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

binaire	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
hexa	**D**				**3**			
Décimal	**211**							

Mise en oeuvre: Le code

- Utilisation de types de données simples
  - Unsigned char = 8bits
  - Unsigned int = 16 bits

➔ Les multiplications et divisions par des puissances de 2 se font par un simple décalage à gauche ou à droite

- Une page avec quelques rappels sympa :
  - https://lucidar.me/fr/c-class/lesson-07-01-bit-masking/

Travaux Pratiques

À vous de ~~jouer~~...

travailler