

# **Gestion du temps : gérer les sorties**

## **Comprendre les Microcontrôleurs**

Jean-Daniel NICOUD et Pierre-Yves ROCHAT

- Durée d'une instruction
- Attente active
- Séquenceurs
- Multi-tâches
- Temps absolu

- Le processeur exécute en permanence des instructions
- La durée d'une instruction est courte ( $< 1 \mu\text{s}$ ), mais bien réelle !
- Cette durée peut être utilisée pour gérer le temps qui passe.
  - Horloge sur un **AVR** :
    - Horloge interne 8 MHz
    - Registre de calibrage
    - Calibrage d'usine
    - Quartz possible, jusqu'à 20 MHz
    - Division possible de la fréquence par 8
  - Horloge sur un **MSP430**:
    - Horloge interne 16 MHz
    - Choix de la fréquence par programme
    - Calibrage d'usine pour 1 et 16 MHz
    - Quartz 32 kHz pour un temps précis

# Prévoir la durée d'une instruction ?

- Instruction assembleur : un ou plusieurs cycles d'horloge
- Instruction en C : *il faudrait regarder quelles instructions assembleur le compilateur a généré !*
- Le temps d'une boucle est **répétitive** !
- Boucles d'attente active

# Boucle d'attente active

```
#define BaseTempsMs 460

void AttenteMs (int duree) {
    volatile int j; // variable de comptage
                    //pour l'unité de temps
    int i; // variable de comptage du nombre
           // d'unités de temps
    for (i=0; i<duree; i++) {
        for (j=0; j<BaseTempsMs; j++){
        }
    }
}
```

```
while (1) { // boucle de test, censée durer 10 secondes
    AttenteMs (10000);
    PORTB ^= (1<<0); // fait changer d'état une LED
}
```

- Ajustage de la constante `BaseTempsMs` pour obtenir les 10 secondes !
- Une précision limitée, suffisante dans beaucoup de cas
- Pour gérer la date et l'heure, il existe des circuits spécialisés

- L'Arduino - et Energia - offrent une procédure similaire : `delay (ms)`

... c'est ce qui nous avait permis d'écrire notre premier clignotant :

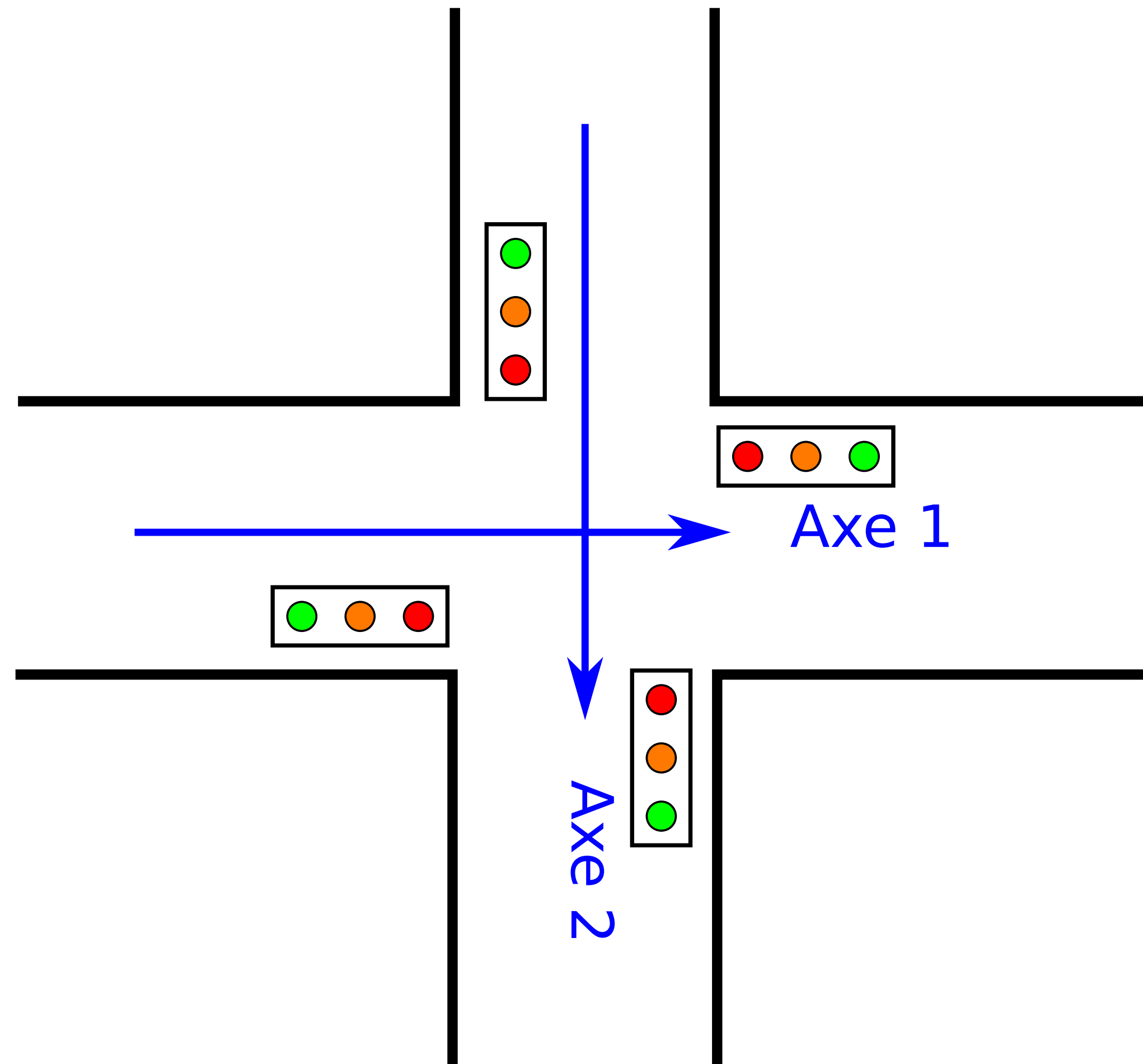
```
void loop () {  
    digitalWrite (LED_ROUGE, HIGH);  
    delay (500);  
    digitalWrite (LED_ROUGE, LOW);  
    delay (500);  
}
```

- **Ne pas oublier** : cet appel est bloquant !

- Faire succéder :
  - des assignation de sorties
  - des attentes.
- Il existe des applications des microcontrôleurs qui n'utilisent pas d'entrées :
  - feu tricolores cycliques
  - boîtes à musique
  - enseignes lumineuses animées
  - journaux lumineux, etc

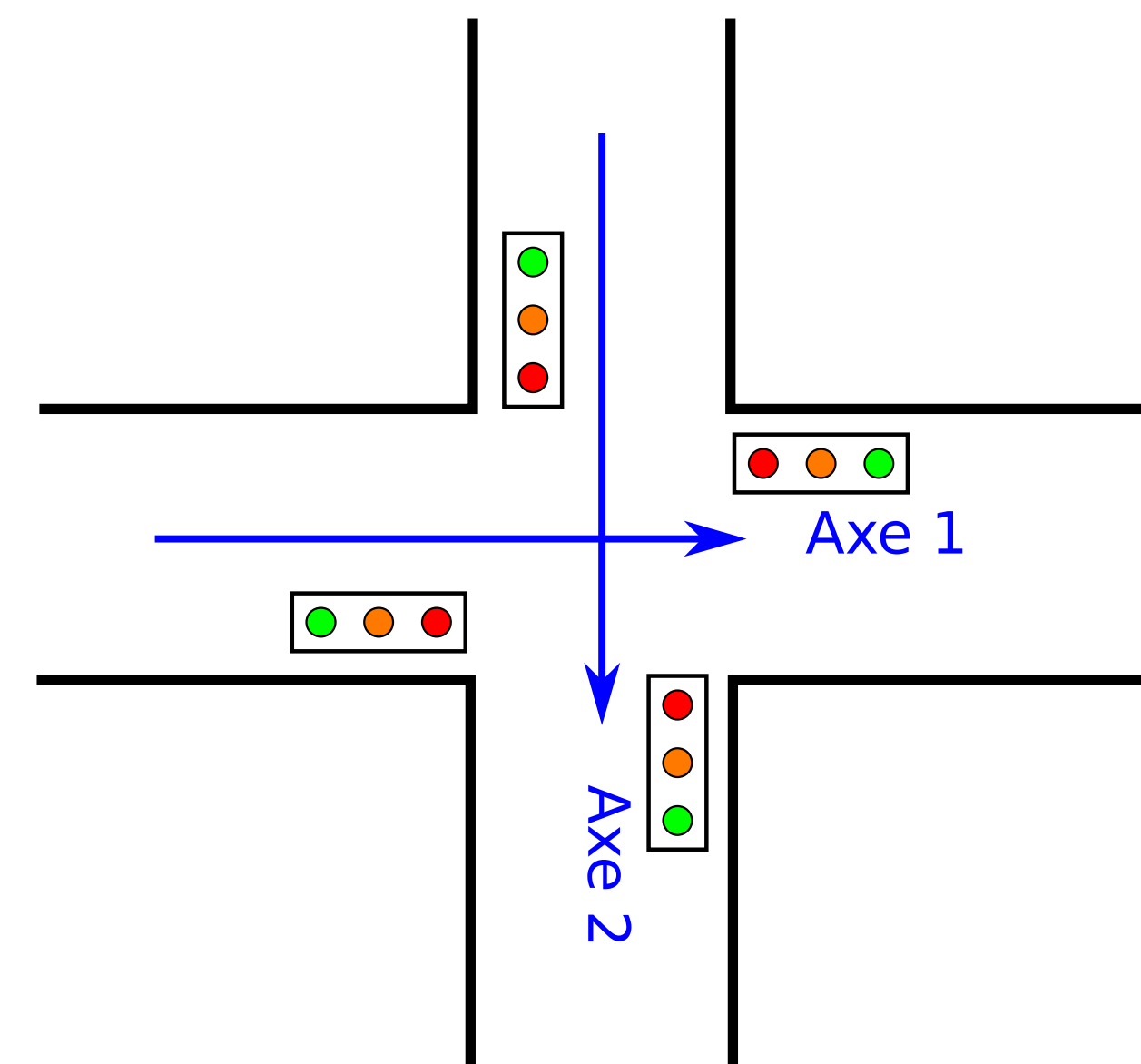


# Feu tricolore cyclique



# Feu tricolore cyclique

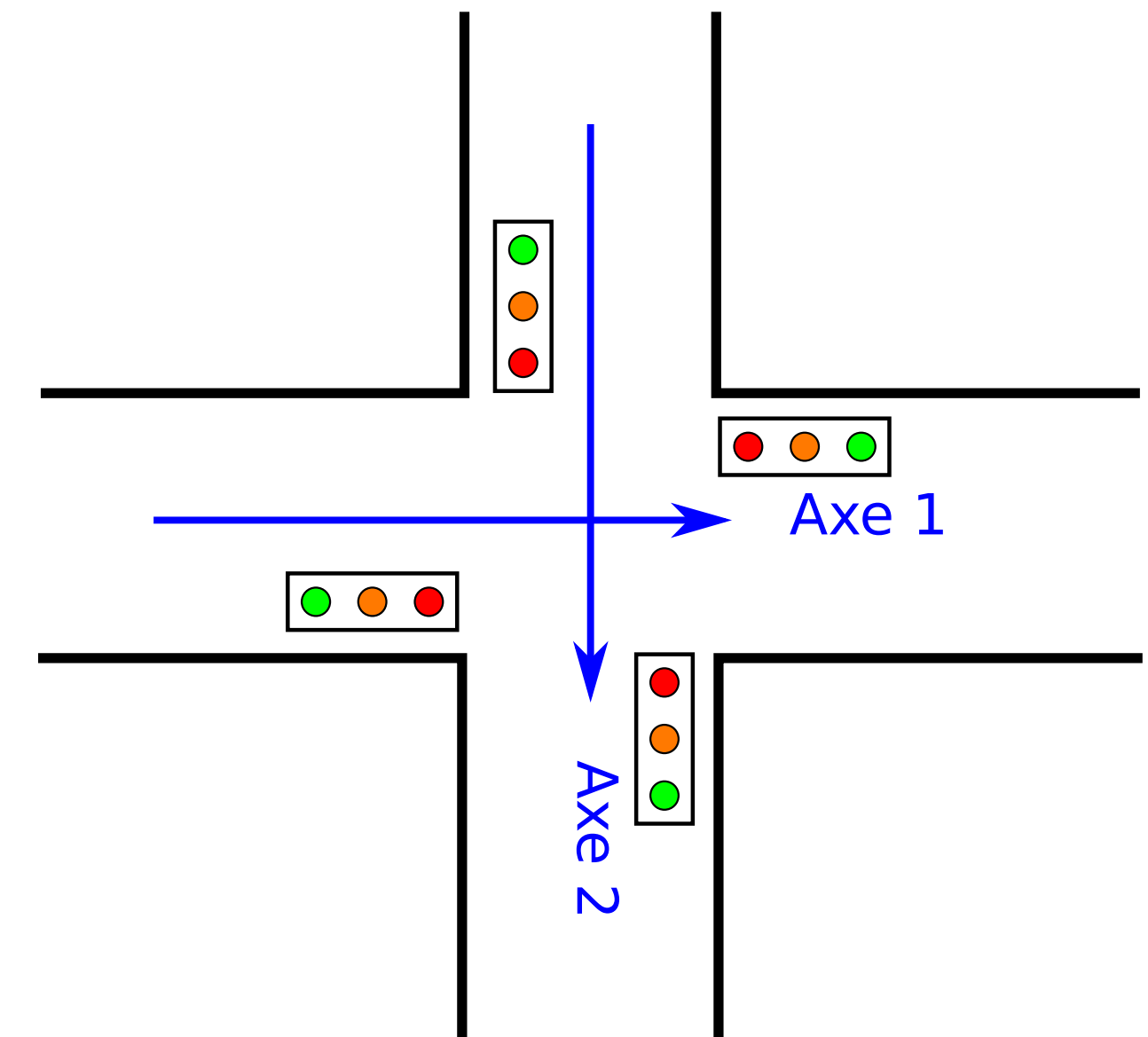
```
enum {Rouge, Orange, Vert}; // Définitions des couleurs
// Allume une couleur sur le feu de l'axe 1 :
void FeuAxe1 (int couleur) {
    digitalWrite(bitRouge1, LOW); // éteint les 3 feux
    digitalWrite(bitOrange1, LOW);
    digitalWrite(bitVert1, LOW);
    switch (couleur){ // ...allume la bonne :
        case Rouge :
            digitalWrite(bitRouge1, HIGH); break;
        case Orange :
            digitalWrite(bitOrange1, HIGH); break;
        case Vert :
            digitalWrite(bitVert1, HIGH); break;
    }
}
// Allume une couleur sur le feu de l'axe 2 :
void FeuAxe2 (int couleur) {
    ...
}
```



# Feu tricolore cyclique

```
while (1) { // boucle principale
  // passage sur l'axe routier 1 :
  FeuAxe2(Rouge); FeuAxe1(Vert); AttenteSec(20);
  // fin du passage :
  FeuAxe1(Orange); AttenteSec(3);
  // blocage de l'axe 1 :
  FeuAxe1(Rouge); AttenteSec(1);


  // passage sur l'axe routier 2 :
  FeuAxe2(Vert); AttenteSec(20);
  // fin du passage
  FeuAxe2(Orange); AttenteSec(3);
  // blocage l'axe 2
  FeuAxe2(Rouge); AttenteSec(1);
}
```



# Gérer plusieurs tâches ?

- Les attentes sont « bloquantes »
- Facile de faire clignoter une LED à 2 Hz
- ... mais difficile d'ajouter une seconde LED qui clignote à 3 Hz !

# Gérer plusieurs tâches ?

- Les attentes sont « bloquantes »
- Facile de faire clignoter une LED à 2 Hz
- ... mais difficile d'ajouter une seconde LED qui clignote à 3 Hz !
-  Idée : - avoir une boucle principale à **durée constante**  
- et ne pas utiliser de boucles à l'intérieur

# Double clignotant à fréquences inégales

```
#define ToggleRedLed {P1OUT ^= (1<<0);} // inverse la LED rouge
#define ToggleGreenLed {P1OUT ^= (1<<6);} // inverse la LED verte
int compteur1=0; int compteur2=0;

while (1) { // boucle principale
    if (compteur1==0) ToggleRedLed;
    compteur1++;
    if (compteur1==250) compteur1=0; // 2Hz, demi période de 250ms

    if (compteur2==0) ToggleGreenLed;
    compteur2++;
    if (compteur2==166) compteur2=0; // 3Hz, demi période de 166ms

    AttenteMs(1);
}
```

- Temps écoulé depuis le début du programme

```
unsigned long TempsMs=0;
while (1) { // boucle principale

    AttenteMs (1);
    TempsMs++;
}
```

# Double clignotant, autre version

```
unsigned long TempsMs=0;

while (1) { // boucle principale
    if ((temps%250) ==0) ToggleRedLed;
    if ((temps%166) ==0) ToggleGreenLed;
    AttenteMs(1); TempsMs++;
}
```



# Double clignotant, plus simple ?

```
unsigned long TempsMs=0;

while (1) { // boucle principale
    if ((temps%250) ==0){ ToggleRedLed;}
    if ((temps%166) ==0){ToggleGreenLed;}
    AttenteMs(1); TempsMs++;
}
```

```
int compteur1=0;
int compteur2=0;

while (1) { // boucle principale
    if (compteur1==0) {ToggleRedLed;}
    compteur1++;
    if (compteur1==250) compteur1=0;

    if (compteur2==0) {ToggleGreenLed;}
    compteur2++;
    if (compteur2==166) {compteur2=0;}

    AttenteMs(1);
}
```

- De nouveau, Arduino – et Energia - offre une procédure similaire :  
`unsigned long millis ( )`
- Mais Arduino offre aussi :  
`unsigned long micros ( )` *en  $\mu$ s*
- Ils ont «un truc» ... les Timers et les interruptions !
- *Ne pas oublier* : ces appels ne sont pas bloquants !

- Durée d'une instruction
- Attente active
- Séquenceurs
- Multi-tâches
- Temps absolu
- Procédures Arduino correspondantes :  
`delay`, `millis` et `micros`