Interruptions

Objectifs

- Définir une fonction d’interruption dans le traitement informatique
  - Définir les termes associés aux interruptions
  - Expliquer le processus en 3 étapes
- Implanter une fonction d’interruption
  - Mettre en œuvre une routine d’interruption
- Assurer la fiabilité d’une donnée en interruption
  - 3 mécanismes

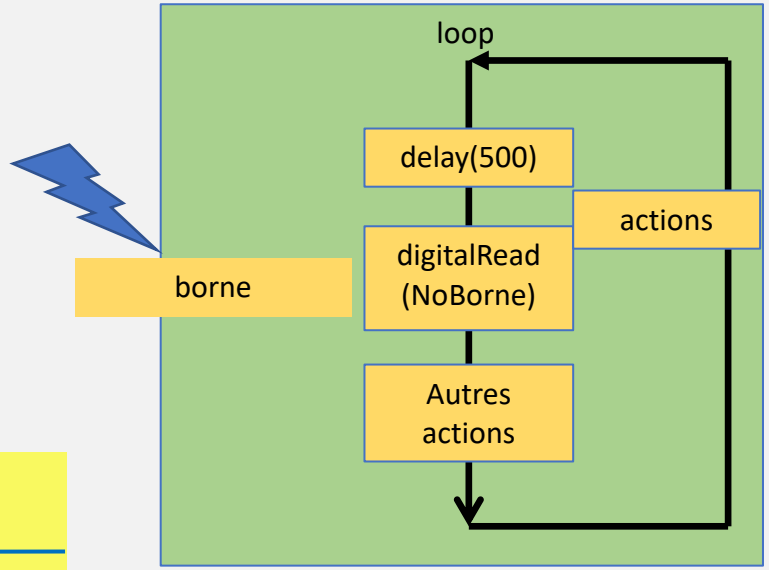
Interruptions

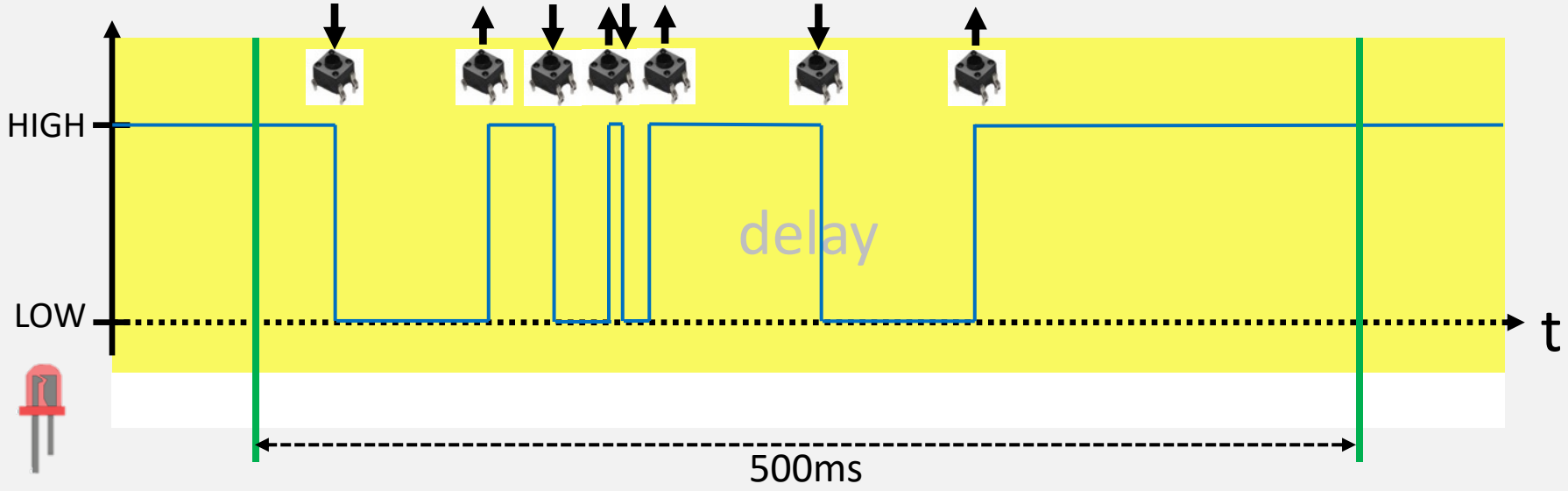
- Quand ?
  - Une action prioritaire doit « passer » avant le traitement en cours
    - Exemple Pop-up vous avez du courriel
  - Répondre à divers évènements, souvent en « parallèle »
- Comment ?
  - Utilisez des instructions désignées pour déclencher et gérer les traitements prioritaires

Interruptions

- Utilisées lorsque des évènements rapides doivent être détectés
  - Perte de temps
    - L'instruction delay() est proscrite
  - Perte d'évènements
    - Exemple: plusieurs passages d'une borne de HIGH à LOW

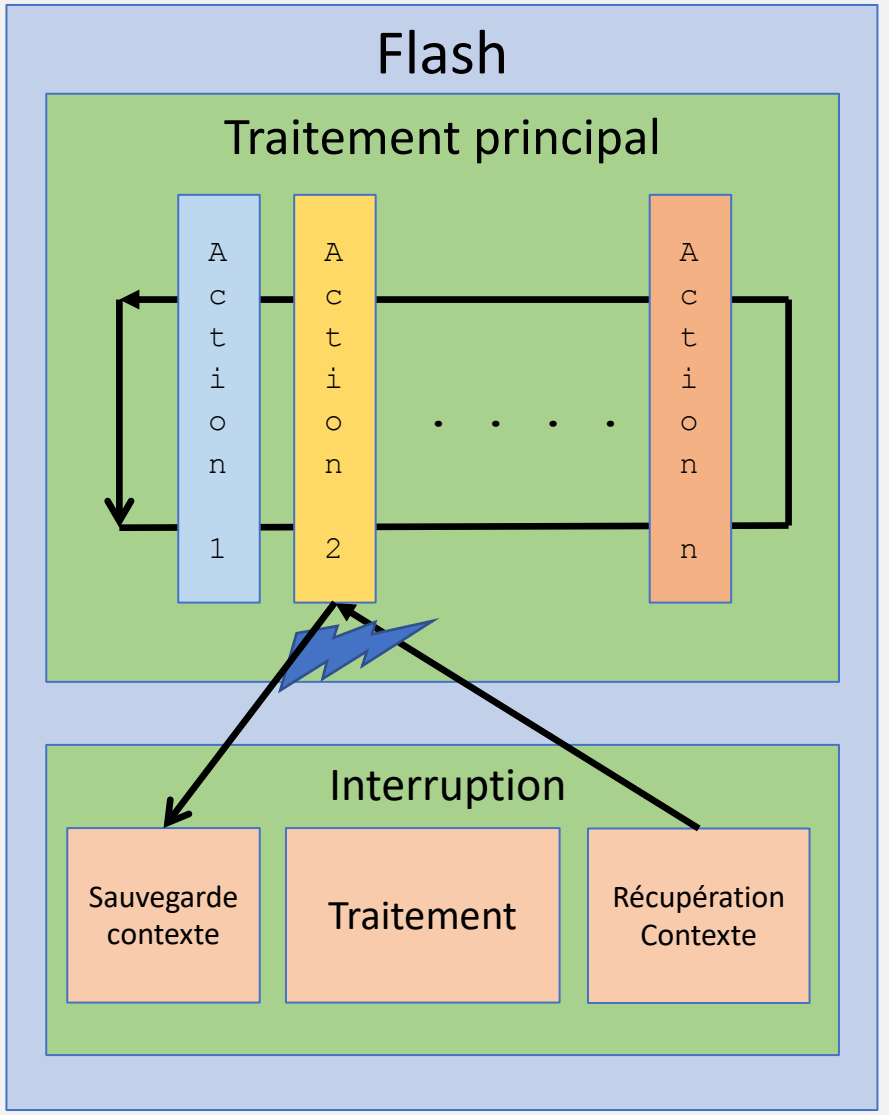
    Non détectés durant les « autres actions »





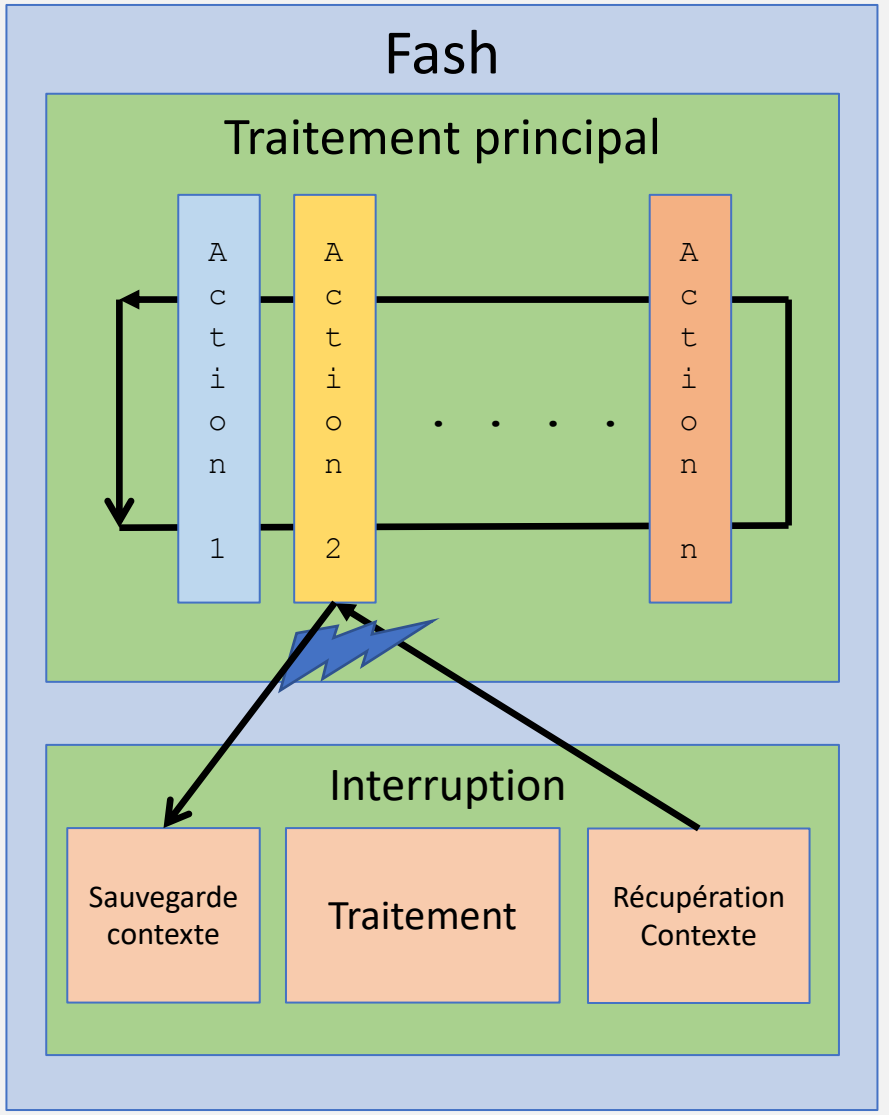
Caractéristiques

- Qualités :
  - Brèves : réduire la latence
  - Fiabilité : instructions de protection des valeurs durant l'interruption
- Types :
  - Externes, provenant de périphériques
    - Exemple : appui d'un bouton-poussoir
  - Internes, provenant de routines
    - Exemple : minuteries



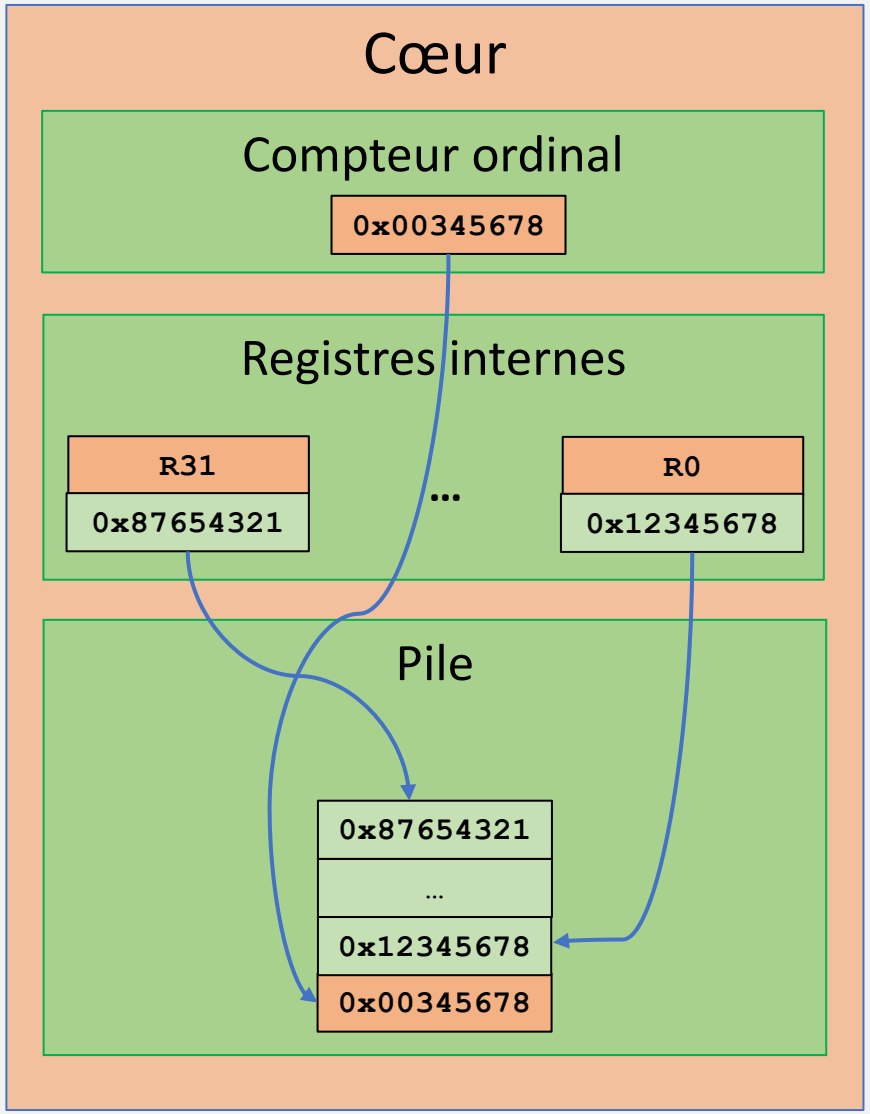
Principe d’interruption

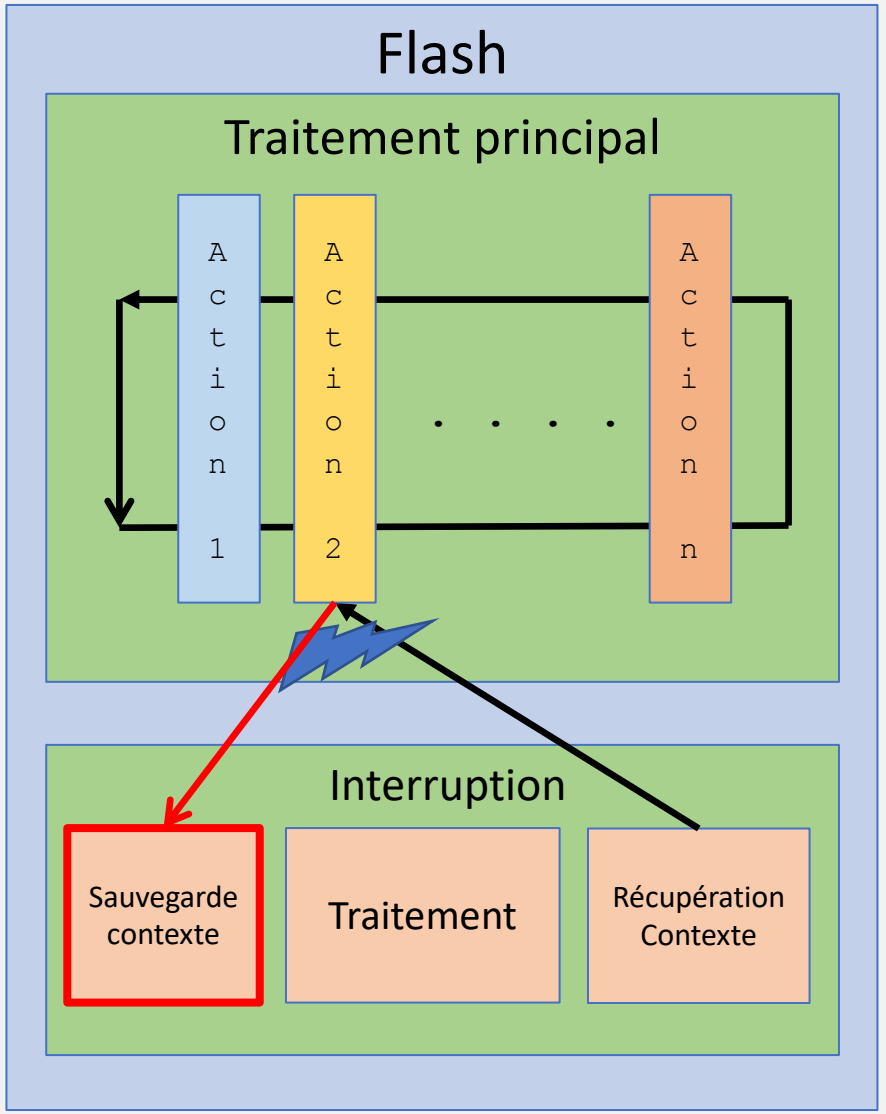
- À tout moment, lorsqu’un évènement survient, le traitement en cours est interrompu
- Le « contexte » est sauvegardé
- Le traitement d’interruption s’exécute
- Le « contexte » est récupéré
- Le traitement principal reprend



Composants impliqués

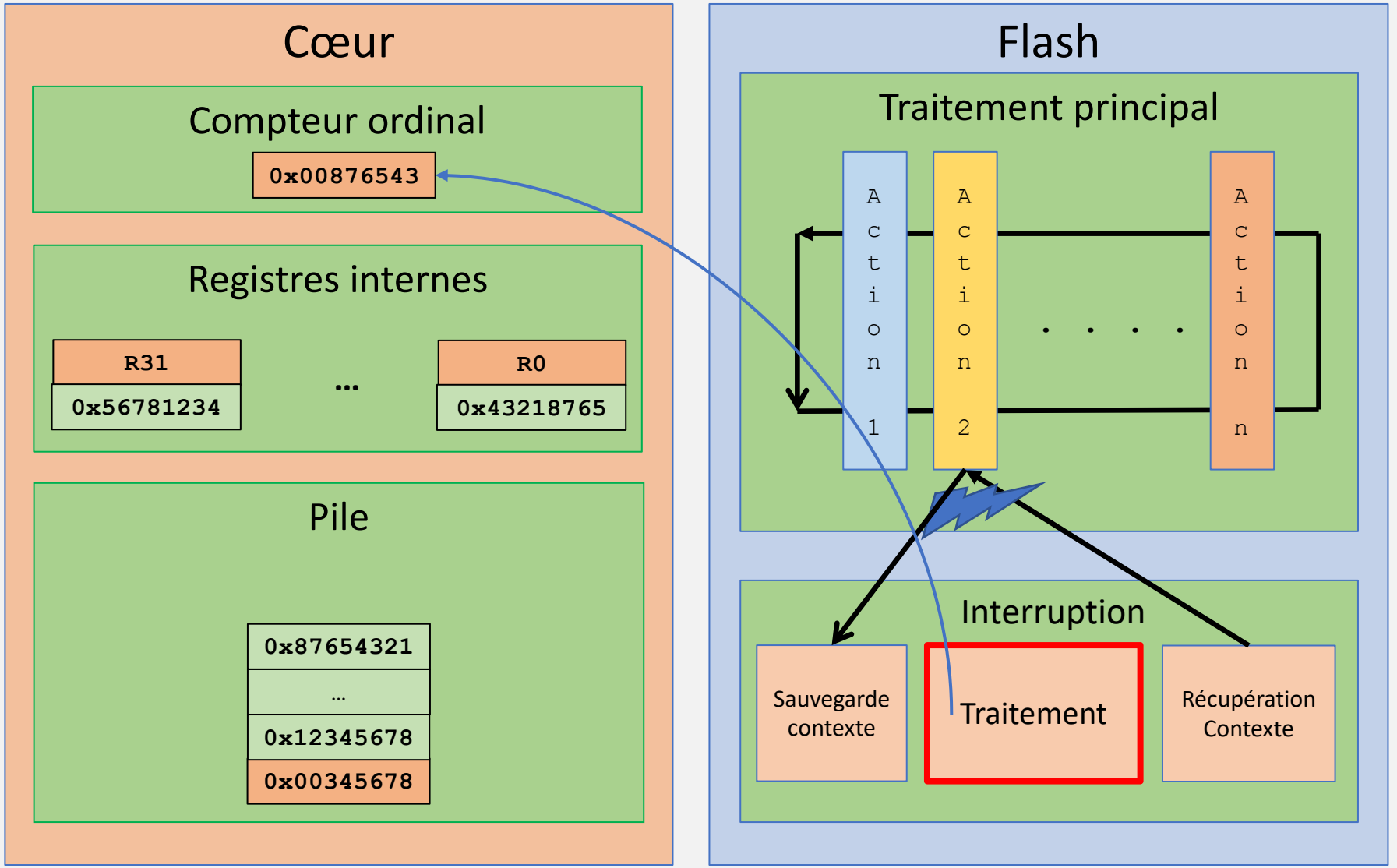
- « contexte » sauvegardé sur la pile pendant l'interruption
  - Compteur ordinal
  - Registres
  - . . .





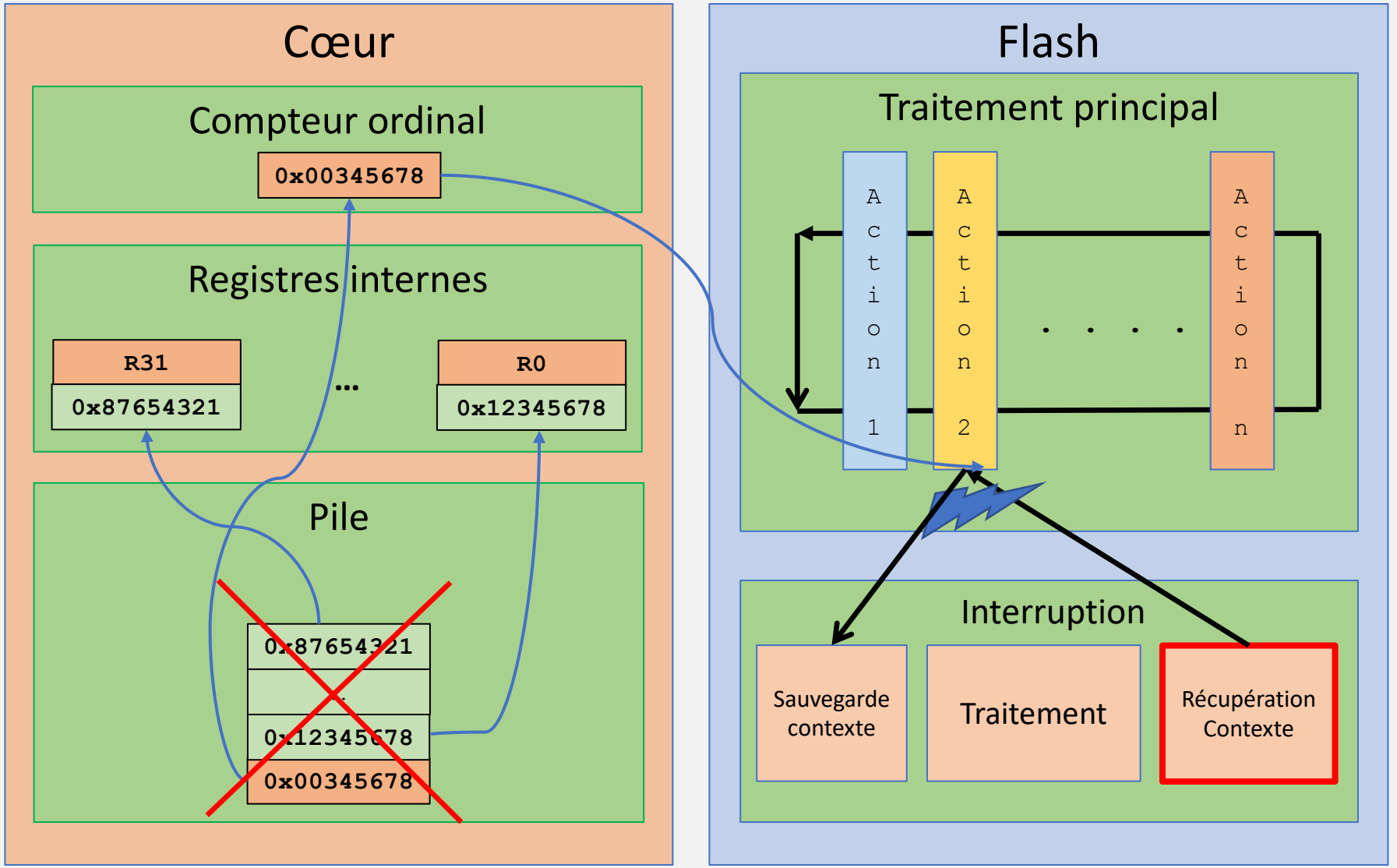
Composants impliqués

- Traitement de l'interruption



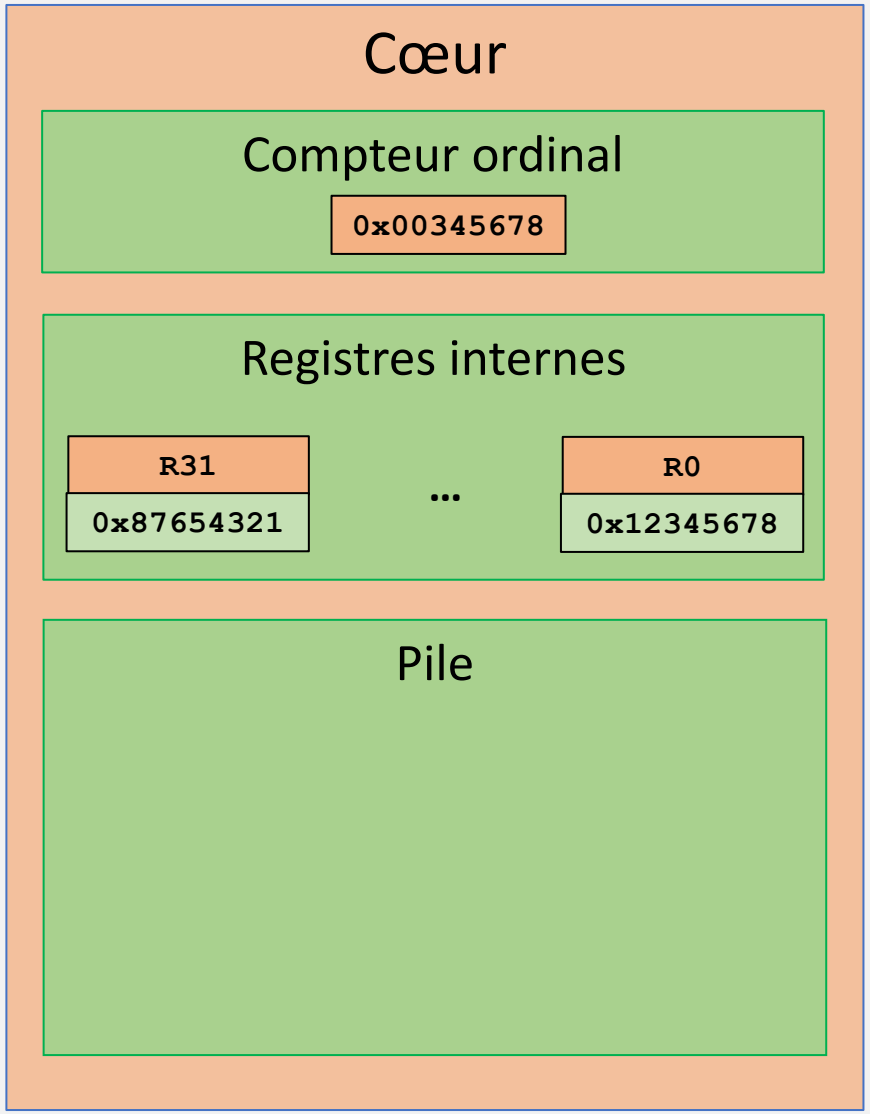
Composants impliqués

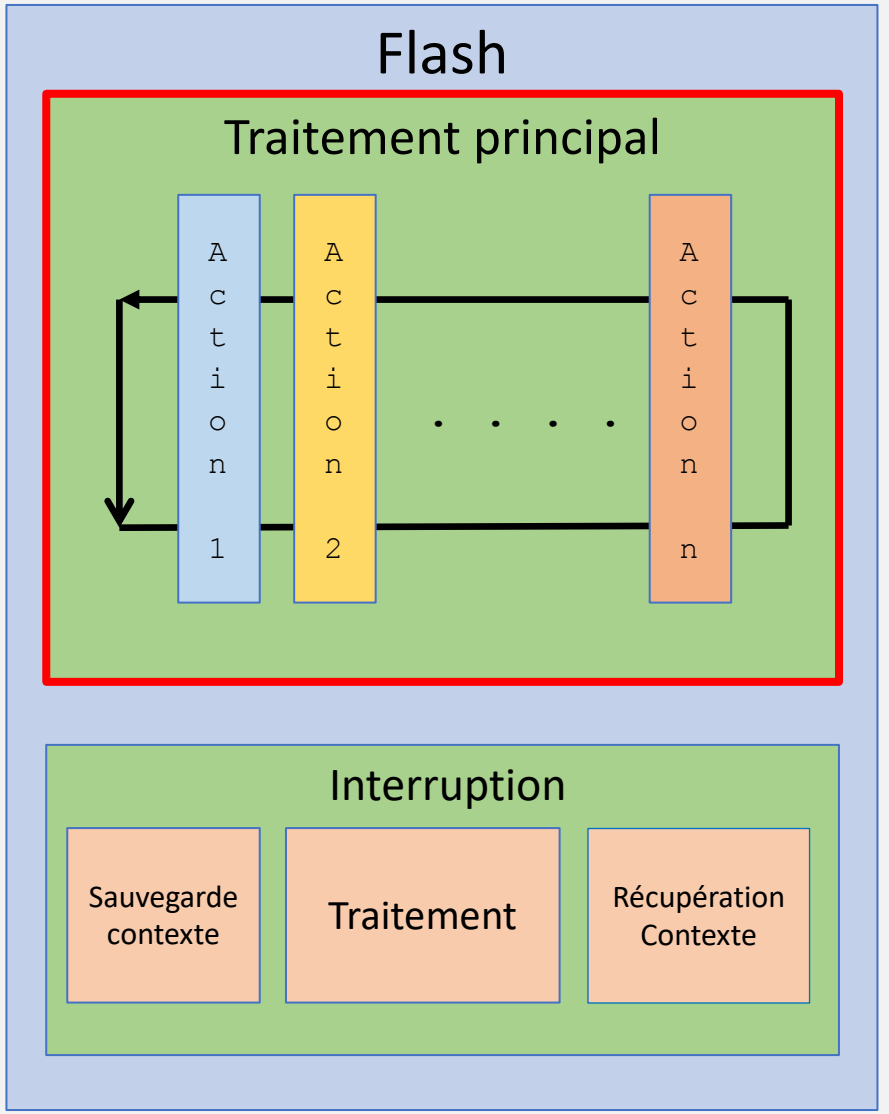
- Restauration du contexte



Composants impliqués

- Reprise du traitement principal
- . . .
- jusqu'à la prochaine interruption





Mise en œuvre sur ESP32

- La fonction attachInterrupt a été spécialement développée pour faciliter la programmation
  - Demande 3 paramètres
    - <*NuméroBorne*>
      - Toutes les bornes du ESP32 sont autorisées, mais…
    - IRAM_ATTR <*fonction d'interruption*>
      - Sans paramètre, sans valeur de retour!
    - <*propriété de déclenchement*>
      - FALLING, CHANGE, RISING, ….

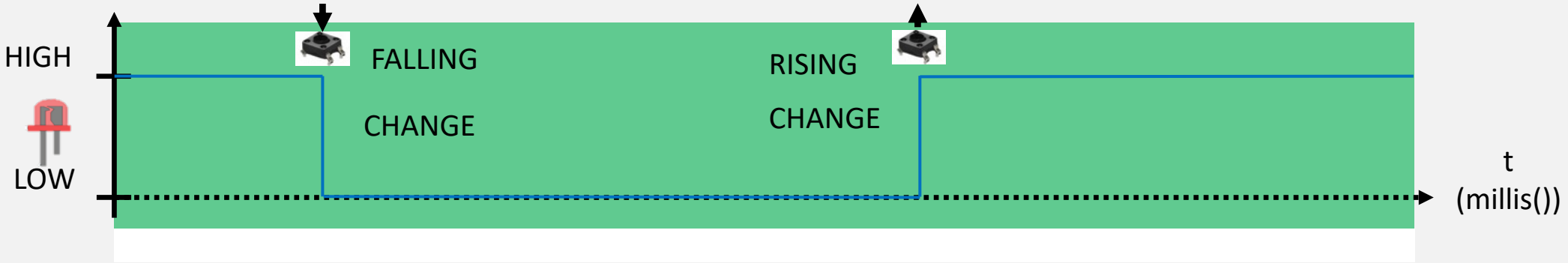
Mise en œuvre sur ESP32

```
void IRAM_ATTR AppuieBouton() {
  compteur++;

}
```

```
void setup() {
  Serial.begin(115200);

  pinMode(PIN_BUTTON, INPUT);
  attachInterrupt(PIN_BUTTON, AppuieBouton , CHANGE);
```



Mise en œuvre sur ESP32

- 1) Fiabilité des données de la routine d'interruption
- Option : **volatile**
  - Toute variable traitée dans la fonction d'interruption et utilisée ailleurs doit être déclarée **volatile**

```
volatile int compteur = 0;

#define PIN_BUTTON 14

#define PIN_LED 2
#define DELAY_LED 2000

void IRAM_ATTR AppuieBouton() {
  compteur++;

}
```

```
void loop() {
  Serial.print("compteur  ");
  Serial.println( compteur);

    digitalWrite(PIN_LED, HIGH);
    delay(DELAY_LED);
    digitalWrite(PIN_LED, LOW);
    delay(DELAY_LED);
}
```

Les fonction AppuieBouton et loop ne sont pas liées entre elles; loop n'appelle pas AppuieBouton et AppuieBouton n'appelle pas loop . Dans ce cas-là, le compilateur « PEUT » utiliser deux références différentes pour la variable compteur, induisant des erreurs de données

Mise en œuvre sur ESP32

- 2)Fiabilité des données de la routine d'interruption
  - Séquence de traitement
    - **nointerrupt()** (1)
    - Utiliser la valeur de la < *variable* >
    - **Interrupt()**

```
void loop() {

  noInterrupts();
  Serial.print("compteur :");
  Serial.println( compteur );
  interrupts();
```

Cette séquence prévient l'interruption durant la « longue période » d'écriture à la console

(1): ces instructions sont des redéfinitions des instruction cli() et sei()

Mise en œuvre sur ESP32

- 3)Fiabilité des données de la routine d'interruption: mutex
- ESP32 est muni d'une protection même de l'écriture de la variable durant l'interruption
- Ainsi, l'interruption se protège elle-même au cas où le bouton serait appuyé durant l'interruption!

```
portMUX_TYPE synchro = portMUX_INITIALIZER_UNLOCKED ;

void IRAM_ATTR AppuieBouton() {
    portENTER_CRITICAL(&synchro);
    compteur++;
    portEXIT_CRITICAL (&synchro);
}
```

- Nous convenons que notre bouton-poussoir est un peu lent pour cette situation. Par contre, certains périphériques ultra-rapides pourraient provoquer cette situation

Références

- https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf : Datasheet
- https://www.youtube.com/watch?v=CJhWlfkf-5M : ESP32 secrets : interrupts and deep-sleep under the hood
- https://www.youtube.com/watch?v=xoASnOYhQ14 : Explications plus détaillées des interruptions sur ATMega328P