

# Quelques éléments sur la programmation événementielle



Fabrice.Kordon@lip6.fr

# Vous avez dit «événement»?

## Les événements?

- Base de la programmation événementielle
- Donc la base pour des applications dites «réactives»

## Interruption

- «\*» provoquant un déroutement vers une adresse identifiée
  - ▶ Interruptions, trappes
  - ▶ Gestion au niveau matériel («vecteur d'interruptions»)

## Événement

- «\*» provoquant un déroutement vers un point de reprise
  - ▶ Niveau d'abstraction plus élevé
  - ▶ Gestion via l'OS
  - ▶ Possibilité d'«empilement»

# Vous avez dit «événement»?

## Les événements

- Base de la programmation concurrente
- Donc la base de l'OS

## Interrupteurs

- «(\*)» provoquant un déroutement vers une routine identifiée
  - ▶ Interruptions, trappes
  - ▶ Gestion au niveau matériel («vecteur d'interruptions»)

## Événement

- «(\*)» provoquant un déroutement vers un point de reprise
  - ▶ Niveau d'abstraction plus élevé
  - ▶ Gestion via l'OS
  - ▶ Possibilité d'«empilement»

### Differences

Mécanismes d'identification du traitement associé

Mécanisme de déroutement

▶ Interruptions, trappes

▶ Gestion au niveau matériel («vecteur d'interruptions»)

# Vous avez dit «événement»?

## Les événements

- Base de la programmation
- Donc la base de l'interfaçage

## Interrupteurs

- «(\*)» provoqués par l'environnement
- ▶ Mécanisme de déroulement
- ▶ Interruptions, trappes, ...
- ▶ Gestion au niveau matériel («vecteur d'interruption»)

## Événements

- «(\*)» provoqués par l'utilisateur
- ▶ Niveaux
- ▶ Gestion
- ▶ Possibilité d'attente

## Différences

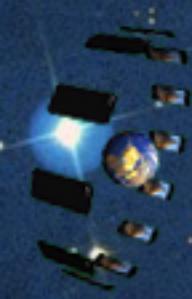
Mécanismes d'identification du traitement associé

Mécanisme de déroulement

Gestion au niveau matériel («vecteur d'interruption»)

## Interfaces graphiques

Un exemple typique depuis l'ancêtre de X (Xerox, années 1970)



# Gestion type par «rétro-appel»

3

## Rétro-Appel = «call-back»

## Mécanismes types offerts par l'OS

- Armer un événement (`enable_event`)
  - ▶ Associer une procédure de déroutement à un événement identifié)
- Désactiver un événement (`disable_event`)
  - ▶ Défaire l'association d'un événement à une procédure de déroutement
- Attendre un événement (`wait_event`)
  - ▶ Laisser l'OS se débrouiller
  - ▶ Utilisation optimale du CPU!!!

## Pour les événements non armés ou désarmés

- L'OS considère qu'il n'y a rien à exécuter
- L'événement sera perdu (pas de notification)

# Exemple sur une interface graphique simplifiée

- ▶ Amorçage des événements
- ▶ Fonctionnement type
- ▶ Masquage



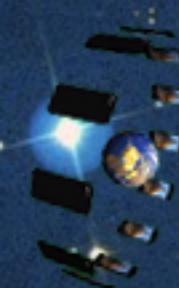
# Amorçage type d'un tel système...

5



## Poser les points de reprise...

```
begin
    enable_event (SIG_DEPL, @maj curseur)
    enable_event (SIG_G, @dessine menu)
    wait_event
end
```



# Amorçage type d'un tel système...

5

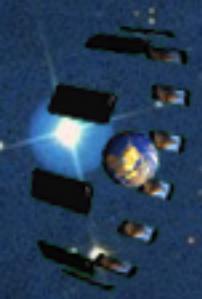
## Poser les points de reprise...

```
begin  
    enable_event (SIG_DEPL, @maj curseur)  
    enable_event (SIG_G, @dessine menu)  
    wait_event  
end
```

C'est tout?

Le gestionnaire  
d'événements gère  
le reste

Et en particulier  
les économies  
d'énergie...



# Mise à jour du déplacement du curseur

6

## Éléments mis à disposition

- Sauvegarde de la position **absolue** du curseur
  - ▶ **Variables globales H et V**
- **Variation** de la position depuis le dernier **wait\_event**
  - ▶ **Variables globales deltaH et deltaV**
- Primitives **efface curseur**
  - ▶ Prend une position passée en paramètre
- Primitive **dessine curseur**
  - ▶ Prend une position passée en paramètre

# Primitives maj curseur

```
begin  
    efface curseur (H, V)  
    H := H + deltaH  
    V := V + deltaV  
    dessine curseur (H, V)  
    wait event  
end
```

# Primitives maj curseur et dessine menu

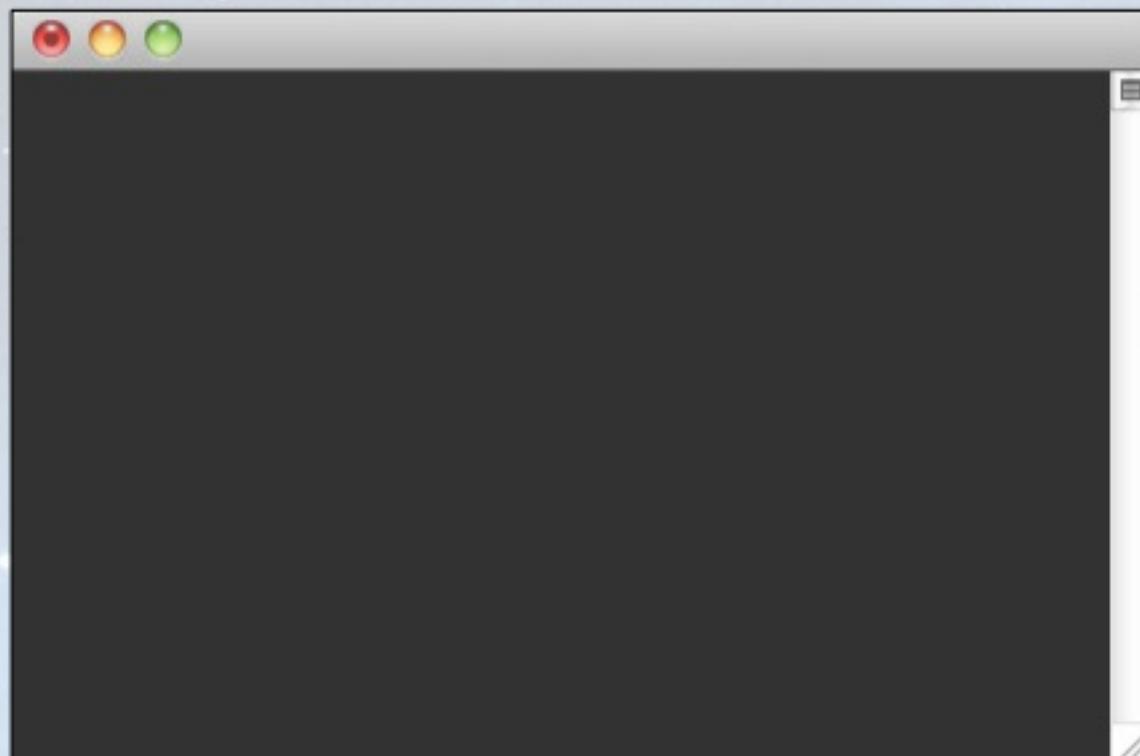
```
begin  
    efface curseur (H, V)  
    H := H + deltaH  
    V := V + deltaV  
    dessine curseur (H, V)  
    wait event  
end  
  
begin  
    efface curseur (H, V)  
    PMV := V  
    PMH := H  
    affiche menu (PMH, PMV)  
    dessine curseur (PMH, PMV)  
    enable event (SIG_M, @active item)  
    enable event (SIG_G, @efface menu)  
    wait event  
end
```



# Mais attention...

8

## Que se passe-t-il si les événements sont préemptifs?



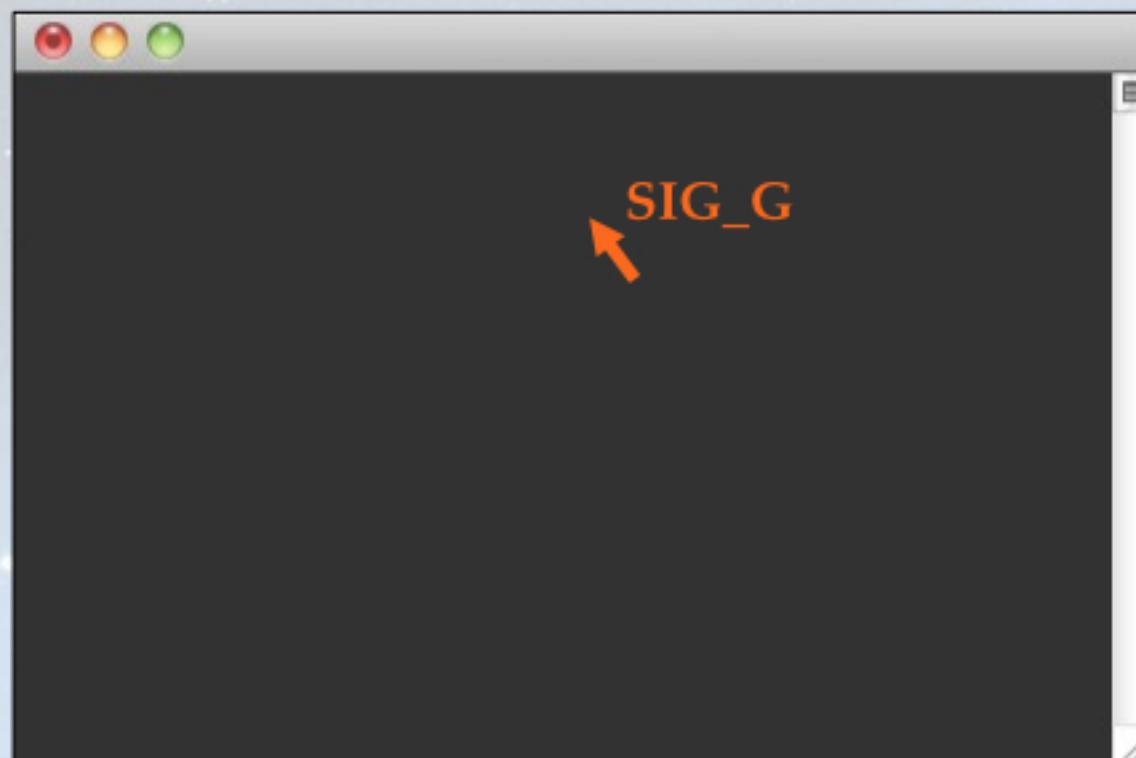
Mon  
programme



# Mais attention...

8

## Que se passe-t-il si les événements sont préemptifs?

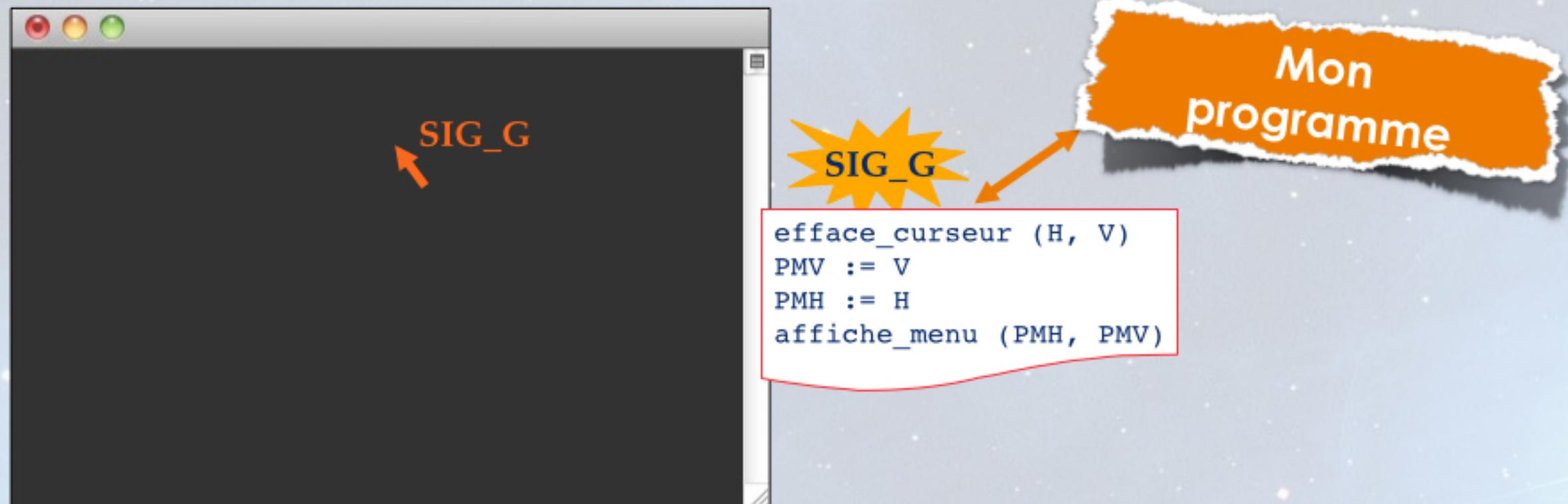


Mon  
programme

# Mais attention...

8

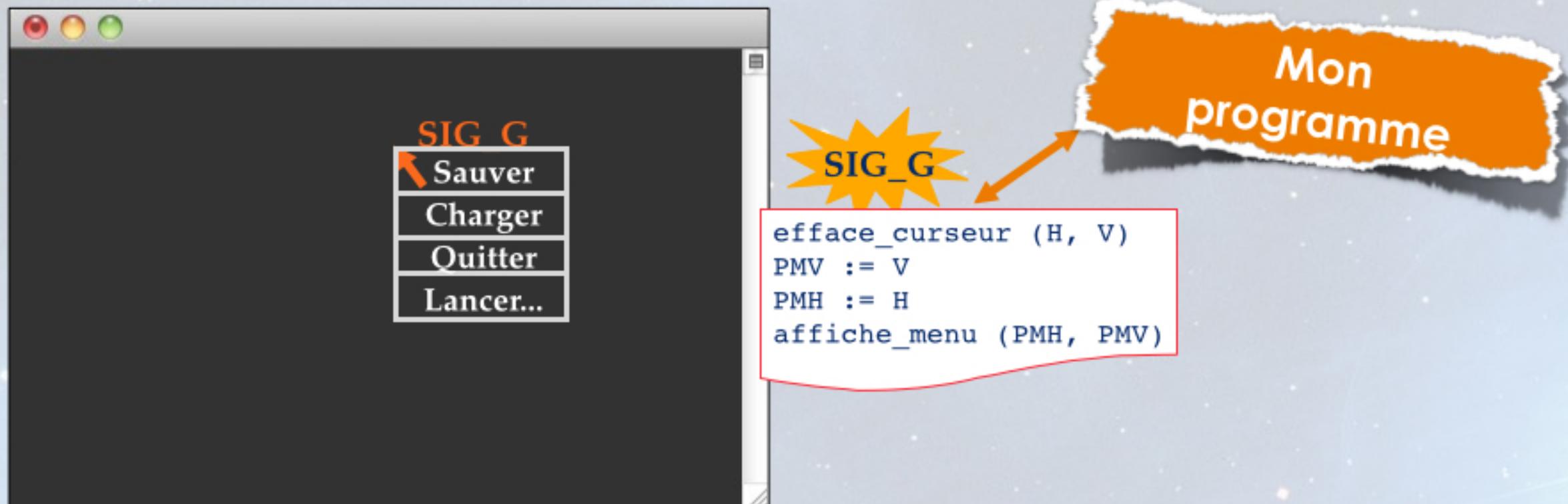
## Que se passe-t-il si les événements sont préemptifs?



# Mais attention...

8

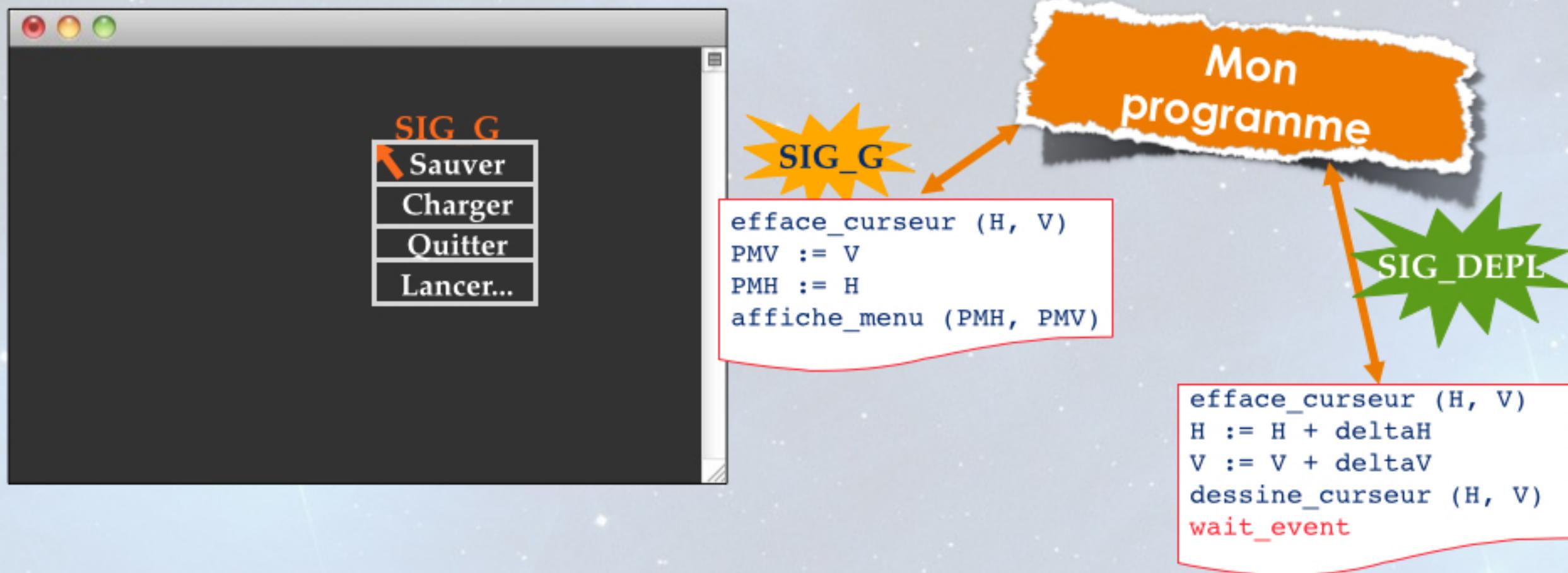
## Que se passe-t-il si les événements sont préemptifs?



# Mais attention...

8

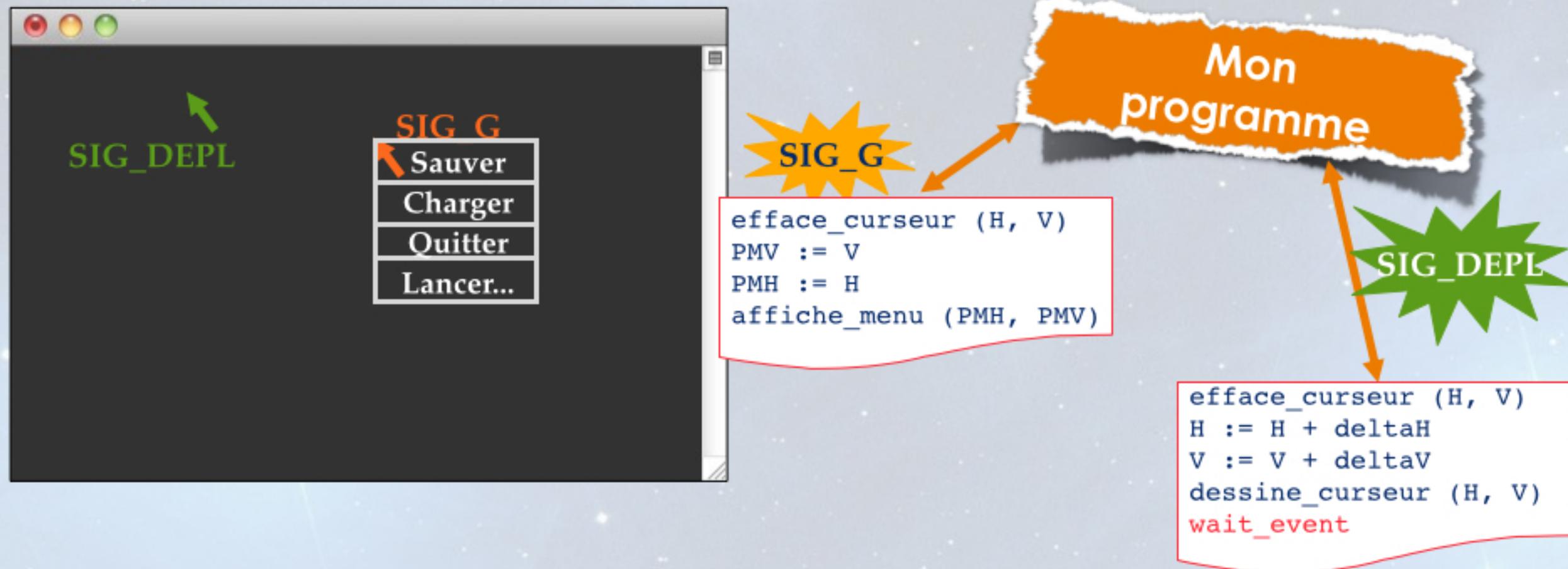
## Que se passe-t-il si les événements sont préemptifs?



# Mais attention...

8

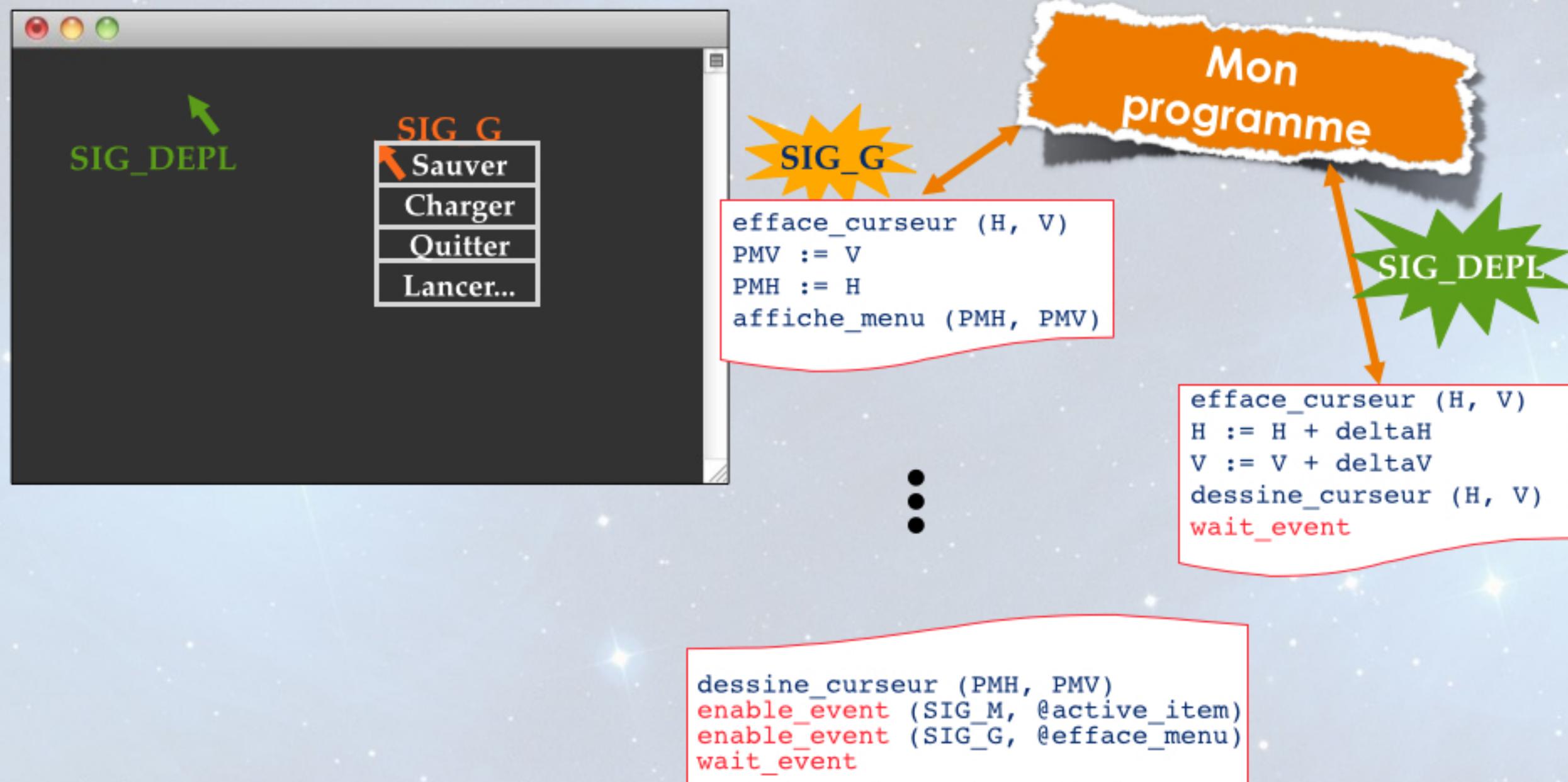
## Que se passe-t-il si les événements sont préemptifs?



# Mais attention...

8

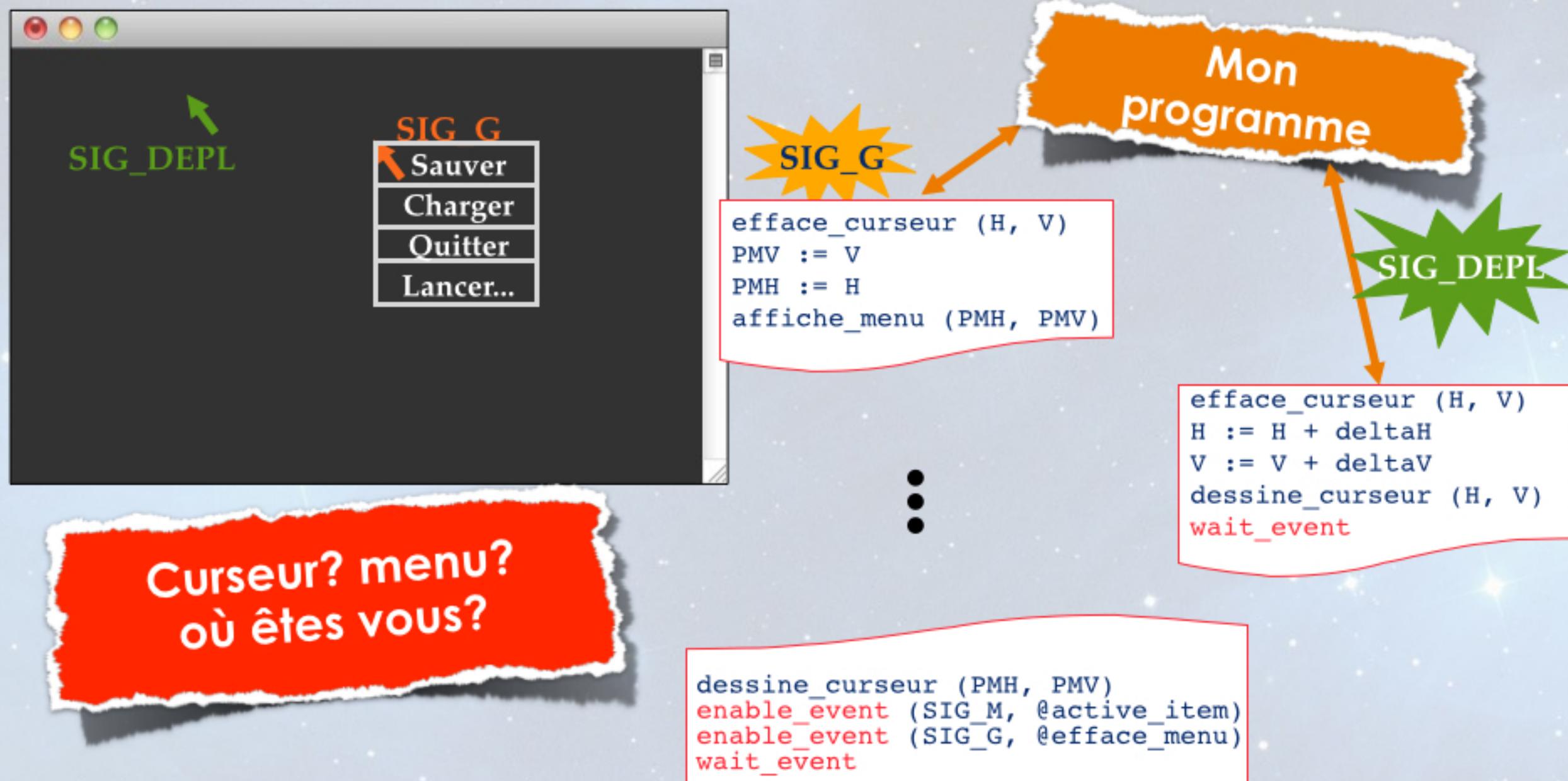
## Que se passe-t-il si les événements sont préemptifs?



# Mais attention...

8

## Que se passe-t-il si les événements sont préemptifs?





# Solution, la séquentialisation

9



## I retarder la prise en compte des événements

- Désarmement



# Solution, la séquentialisation

9

## I retarder la prise en compte des événements

- Désarmement

```
begin
    disarm_event
    efface curseur (H, V)
    PMV := V
    PMH := H
    affiche menu (PMH, PMV)
    dessine curseur (PMH, PMV)
    enable_event (SIG_M, @active_item)
    enable_event (SIG_G, @efface_menu)
    retrieve_event
    wait_event
end
```

```
begin
    disarm_event
    efface curseur (H, V)
    H := H + deltaH
    V := V + deltaV
    dessine curseur (H, V)
    retrieve_event
    wait_event
end
```

# La «boucle événementielle» (exemple type, la ligne série)

- ▶ **Structure d'une ligne série**
- ▶ **Programmation du gestionnaire**

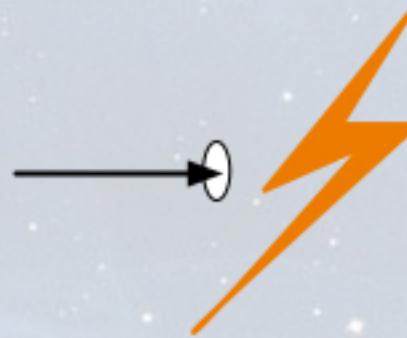
## Structure du système



## Modalités

- **cpt** initialisé à 0
- Remplissage progressif à chaque interruption
- Recopie vers l'espace du destinataire quand le buffer est plein
  - ▶ i.e. **cpt = 512**

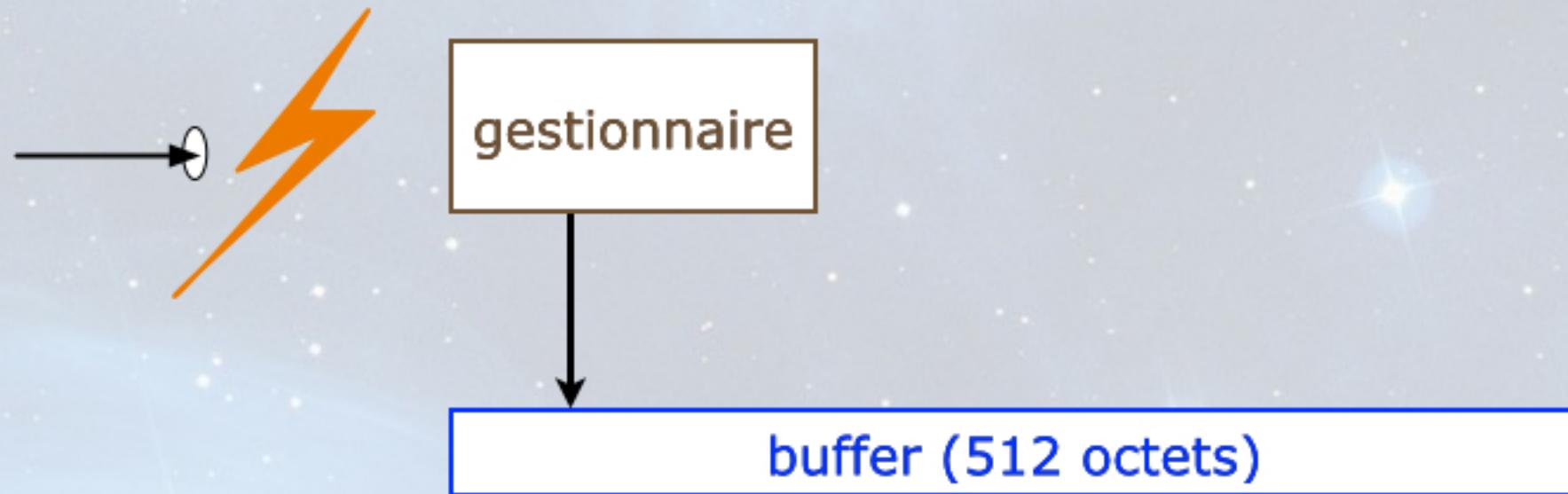
## Structure du système



## Modalités

- **cpt** initialisé à 0
- Remplissage progressif à chaque interruption
- Recopie vers l'espace du destinataire quand le buffer est plein
  - ▶ i.e. **cpt = 512**

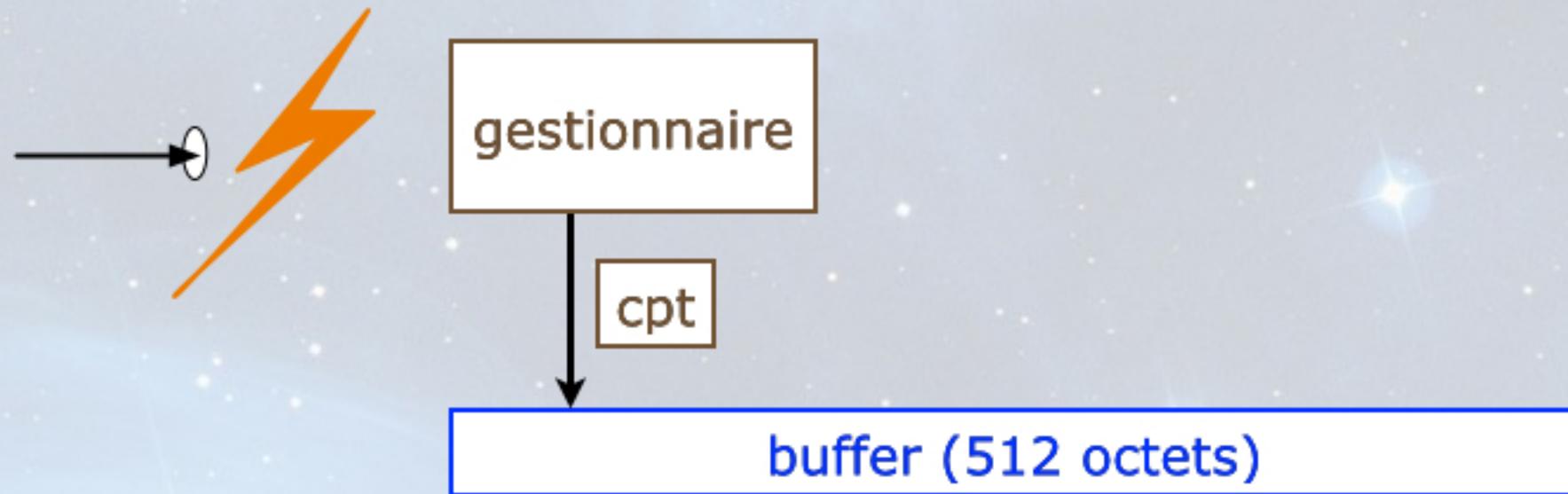
## Structure du système



## Modalités

- **cpt** initialisé à 0
- Remplissage progressif à chaque interruption
- Recopie vers l'espace du destinataire quand le buffer est plein
  - ▶ i.e. **cpt = 512**

## Structure du système



## Modalités

- **cpt** initialisé à 0
- Remplissage progressif à chaque interruption
- Recopie vers l'espace du destinataire quand le buffer est plein
  - ▶ i.e. **cpt = 512**



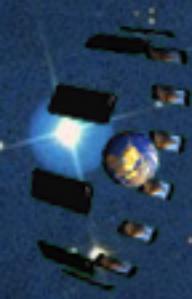
# Programmation du gestionnaire («handler»)

12

## Masquer les interruptions

- Pas les désarmer

```
begin
    mask (SER_LINE)
    buffer(cpt) := (REG_SER_LINE)
    cpt++
    if cpt = 512 then
        recopier le buffer dans l'espace de données
        du récipiendaire
        cpt := 0;
    end if
    unmask (SER_LINE)
end
```



# En guise de conclusion

13

## ■ Smartphones/tablettes, des terminaux centrés sur leur interface utilisateur

- Programmation réactive (comme les interfaces graphiques)
- mécanismes de programmation événementielle

## ■ Événements : gestion délicate

- Une «certaine forme» de parallélisme
- Manipulation par effets de bords (éléments partagés)
  - ▶ Cf la boucle événementielle
- La gestion d'événements s'appuie maintenant sur des langages objets
  - ▶ e.g. Objective C/Swift pour iOS, Java pour Android



