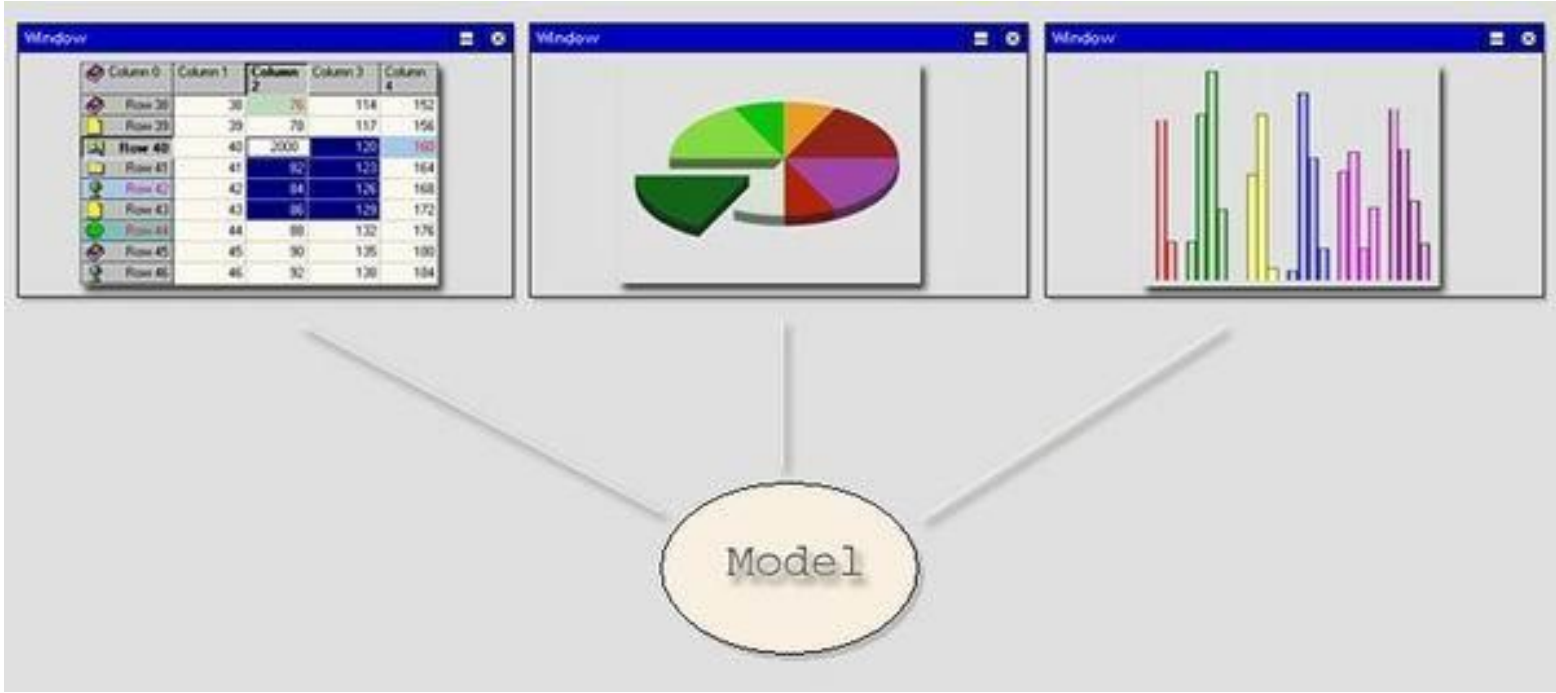
I H-M A
Complémént
MVC

Interactions avec le système : modèle et vue

- L'interface utilisateur est chargée de représenter, sous une forme interprétable par un humain, les informations internes de l'application.
- Ces informations constituent le **modèle *(model)*** de l'application (les données représentant son état actuel).
- On peut créer plusieurs **vues** distinctes d'un même modèle.



Interactions avec le système : contrôleur

- Lorsque l'utilisateur interagit avec le système
  - Déplacement de la souris, clic, double-clic, …
  - Frappe sur les touches du clavier (caractères, touches de fonctions, …)

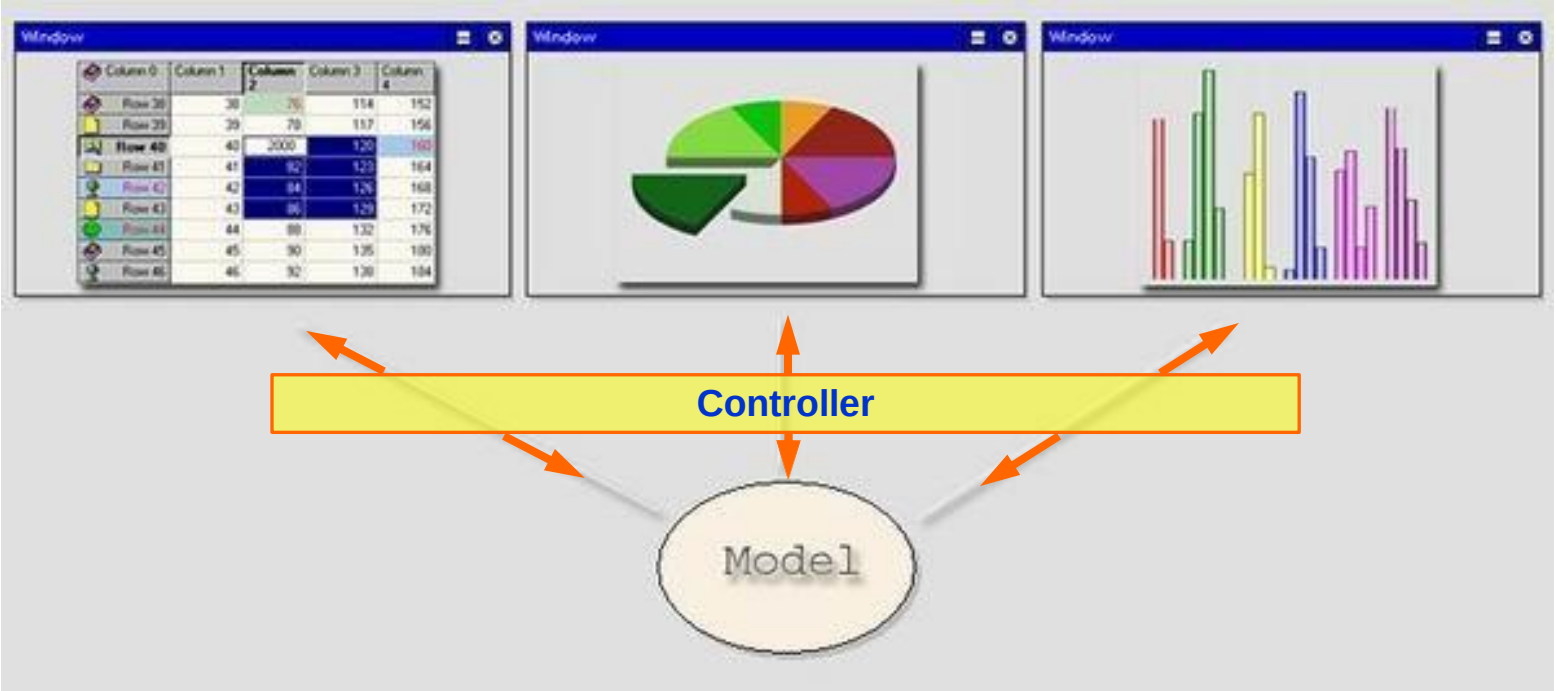
le composant logiciel qui gère ces actions est nommé le **contrôleur** (***controller***) du système.

- Le rôle du contrôleur est donc de **réagir aux actions de l'utilisateurs**

- Si l'utilisateur agit au moyen de la souris (ou touchpad, trackpad, joystick, tablette graphique, etc.), l'interaction se fait au travers de la vue
- Lors de la saisie, au clavier, d'un champ de texte (ou d'une zone de texte, ou lors de la navigation à l'aide du clavier), la vue est également impliquée dans l'interaction
- Il y a donc souvent une **relation assez forte entre le contrôleur et la vue**

Synchronisation modèles ⇔ vues

- Qui se charge d'assurer la cohérence entre les données du modèle et leur représentation dans les différentes vues ?
- Dans la variante simple (nommée synchrone) de l'architecture MVC, c'est le rôle du **contrôleur** qui agit comme un intermédiaire et qui a la charge de synchroniser les informations entre les vues et le modèle.



Architecture MVC

- L'**architecture MVC** *(Model-View-Controller)* est un **modèle de conception** (***Design Pattern***) très classique qui a été introduit avec le langage *Smalltalk-80*.
- Le principe de base de l'architecture MVC est relativement simple, on divise le système interactif en **trois parties distinctes** :
  - le **modèle** (***Model***) qui offre l'accès et permet la gestion des données (état du système)
  - la **vue** (***View***) qui a pour tâche l'affichage des informations (visualisation) et qui participe à la détection de certaines actions de l'utilisateur
  - le **contrôleur** (***Controller***) qui est chargé de réagir aux actions de l'utilisateur (clavier, souris) et à d'autres événements internes et externes
- Ce modèle de conception simplifie le développement et la maintenance des applications en répartissant et en découplant les activités dans différents sous-systèmes (plus ou moins) indépendants.

Interactions MVC

La vue consulte les données du modèle

Model

View

Le contrôleur consulte et met à jour le modèle en fonction des événements

La vue détermine quels événements sont passés au contrôleur.

Le contrôleur peut modifier les éléments affichés

Controller

Flux de données

Événements

MVC / Le modèle (Model)

- **Le Modèle** *(**Model**)* est responsable de la gestion de l'**état du système** (son contenu actuel, la valeur de ses données).
- Il offre également les **méthodes** et **fonctions** permettant de gérer, transformer et manipuler ces données (logique de l'application).
- Le modèle peut informer les vues des changements intervenus dans ses données. La communication de ces changements intervient en général sous la forme d'événements qui seront gérés par des contrôleurs (les vues s'enregistrent auprès du modèle pour être notifiées lors des changements dans les données)
- Les informations gérées par le modèle sont indépendantes de la manière dont elles seront affichées. En fait, le modèle doit pouvoir exister indépendamment de la représentation visuelle des données.
- Dans certaines situations (simples) le **modèle** peut contenir lui-même les données, mais la plupart du temps, il **agit** comme un **intermédiaire** *(proxy)* vers les données qui sont stockées dans une base de données ou un serveur d'informations (en *Java*, le modèle est souvent défini par une **interface**).

MVC / La vue (View)

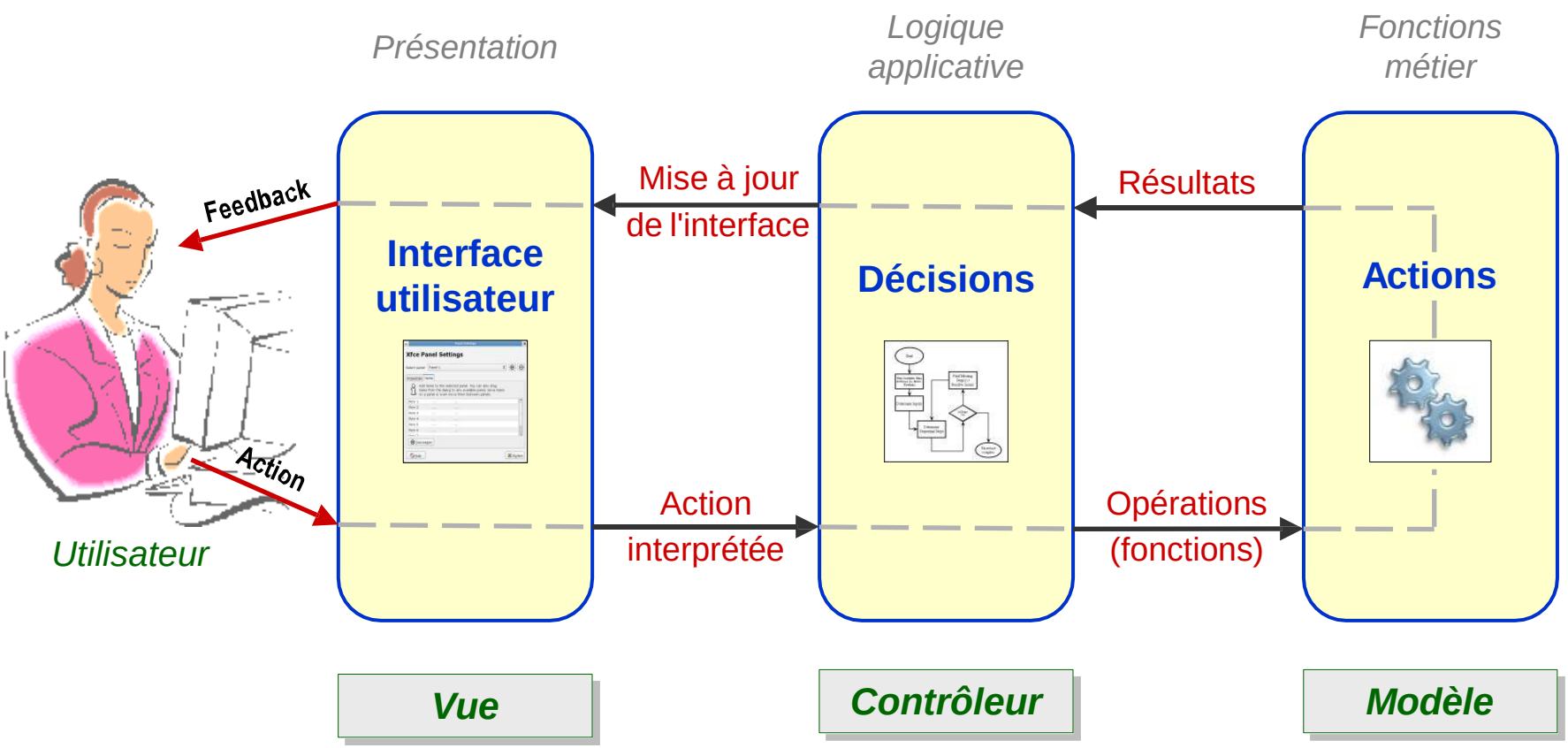
- **La vue** ***(View)*** se charge de la **représentation visuelle** des informations.
- La vue utilise les données provenant du modèle pour afficher les informations. La vue doit être informée des modifications intervenues dans certaines données du modèle (celles qui influencent l'affichage).
- Plusieurs vues différentes peuvent utiliser le même modèle (plusieurs représentations possibles d'un même jeu de données).
- La vue intercepte certaines actions de l'utilisateur et les transmet au contrôleur pour qu'il les traite (souris, événements clavier, …).
- Le contrôleur peut également modifier la vue en réaction à certaines actions de l'utilisateur (par exemple afficher une nouvelle fenêtre).
- La représentation visuelle des informations affichées peut dépendre du *Look-and-Feel* adopté (ou imposé) et peut varier d'un système d'exploitation à l'autre. L'utilisateur peut parfois modifier lui même la présentation des informations en choisissant par exemple un thème.

MVC / Le contrôleur (Controller)

- **Le contrôleur** ***(Controller)*** est chargé de réagir aux différents événements qui peuvent survenir.
- Les **événements** sont constitués soit par des **actions de l'utilisateur** (presser sur une touche, cliquer sur un bouton, fermer une fenêtre, ...) ou par des **directives venant du programme** lui-même (un changement intervenu dans un autre composant, l'écoulement d'un certain temps, etc...).
- Le contrôleur **définit le comportement** de l'application (comment elle réagit aux sollicitations).
- Dans les applications simples, le contrôleur gère la synchronisation entre la vue et le modèle (rôle de chef d'orchestre).
- Le contrôleur est informé des événements qui doivent être traités et sait d'où ils proviennent.
- Le contrôleur peut agir sur la vue en modifiant les éléments affichés.
- Le contrôleur peut également, si nécessaire, modifier le modèle en réaction à certains événements.

Rôles des éléments de l'architecture MVC

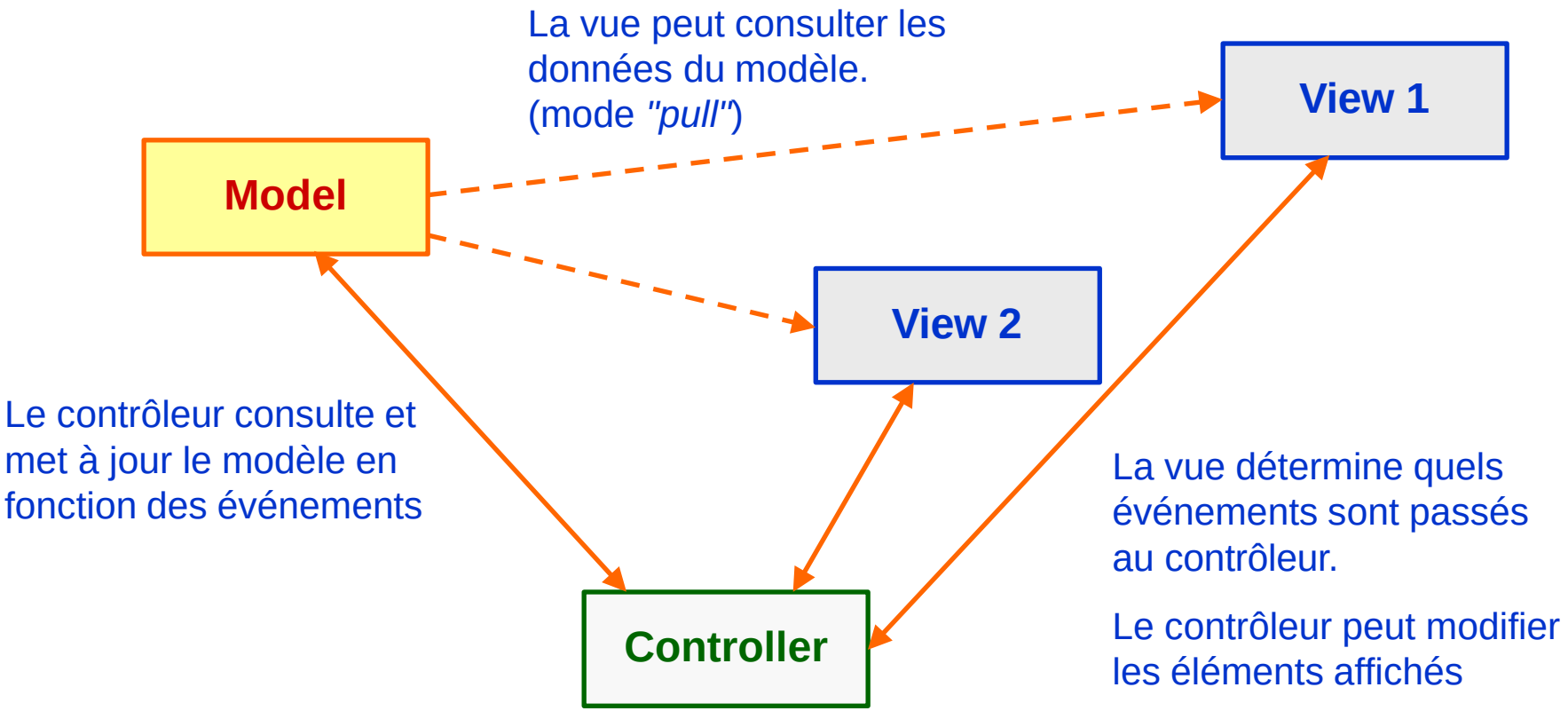
- Lorsqu'un utilisateur interagit avec une interface, les différents éléments de l'architecture MVC jouent chacun un rôle bien défini.



MVC : Découplage et modèle synchrone

- L'**architecture MVC** est un modèle de conception qui présente comme intérêt principal de modulariser (découper) l'application en éléments ayant des rôles distincts et qui permet (si l'on respecte l'esprit de cette architecture) de minimiser les dépendances entre ces modules.
- Le découplage entre les modules favorise la lisibilité ainsi que la maintenance des applications (on peut modifier un élément du système sans que tous les autres en soient affectés).
- Il existe différentes manières de gérer la communication entre les trois entités : *Modèle* - *Vue(s)* - *Contrôleur(s)*.
- Avec un **modèle synchrone** - dans lequel le contrôleur joue le rôle de *chef d'orchestre* - le couplage reste assez fort entre le contrôleur et la ou les vues. L'essentiel des mises à jour des vues passe par le contrôleur.
- Ce modèle d'interaction fonctionne pour des applications de petite envergure mais n'est pas adapté si l'application est plus complexe et notamment lorsqu'il y a plusieurs vues pour un même modèle.

MVC : Interactions synchrones



MVC : Découplage et modèle asynchrone

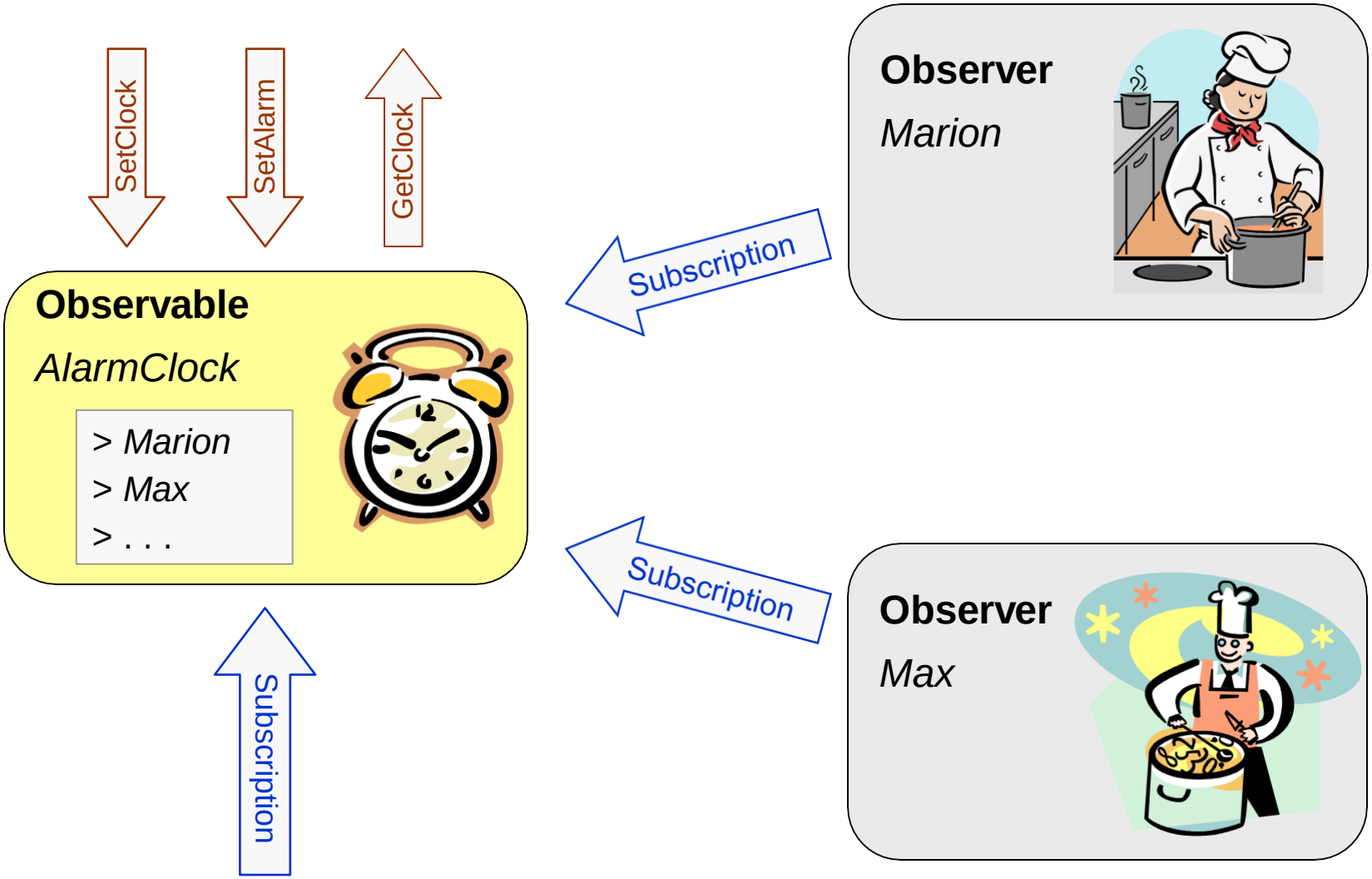
- Il est possible de réduire le couplage entre les modules et de restreindre le rôle central du contrôleur en utilisant un **modèle asynchrone** basé sur le modèle de conception *Observer pattern.*
- Ce mode de communication entre deux entités est fréquent et on le trouve sous différentes désignations :
  - Observer / Observable (Subject)
  - Subscribe / Notify
  - Subscriber-Publisher Model

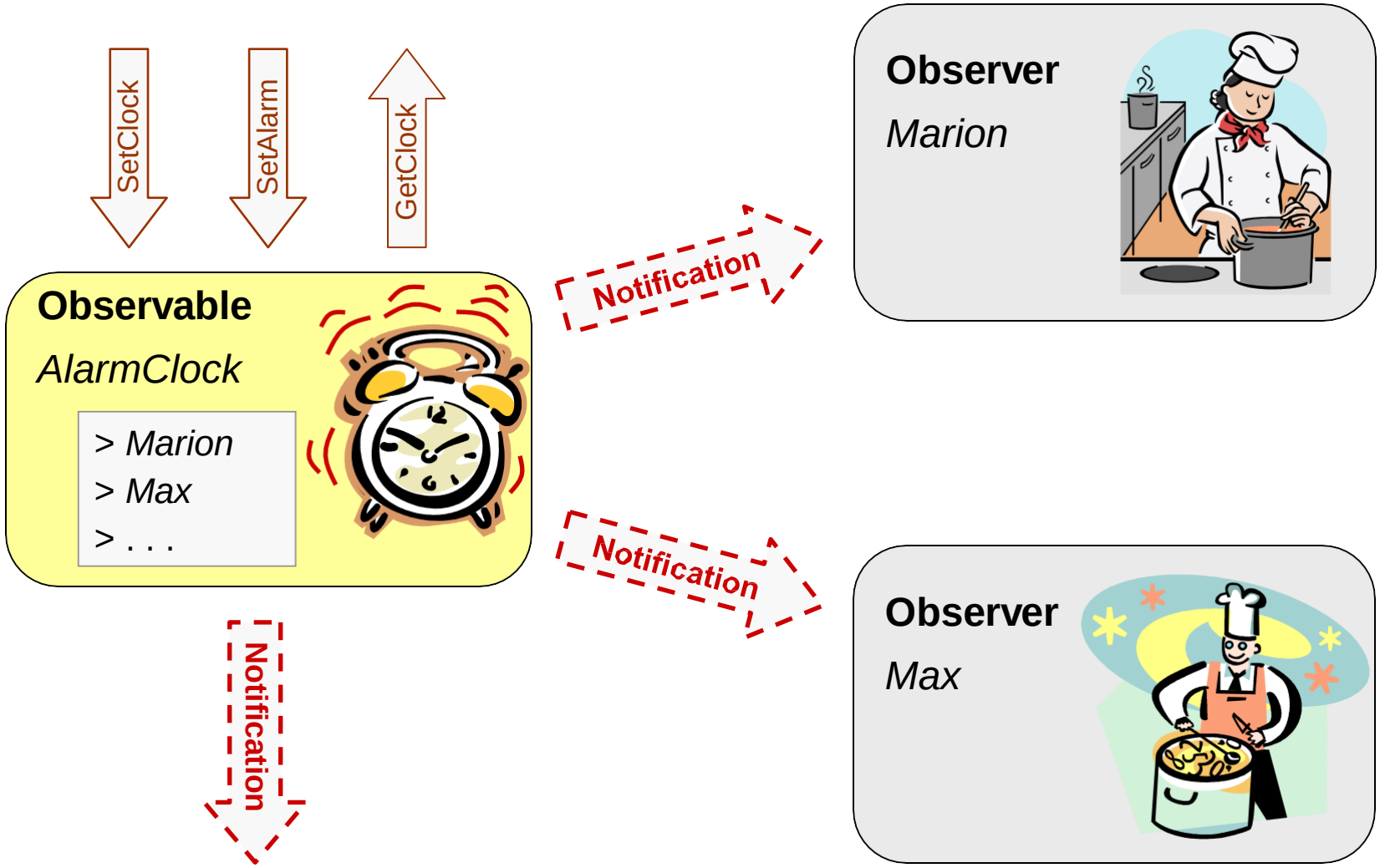
  Push Model

  Callback Pattern

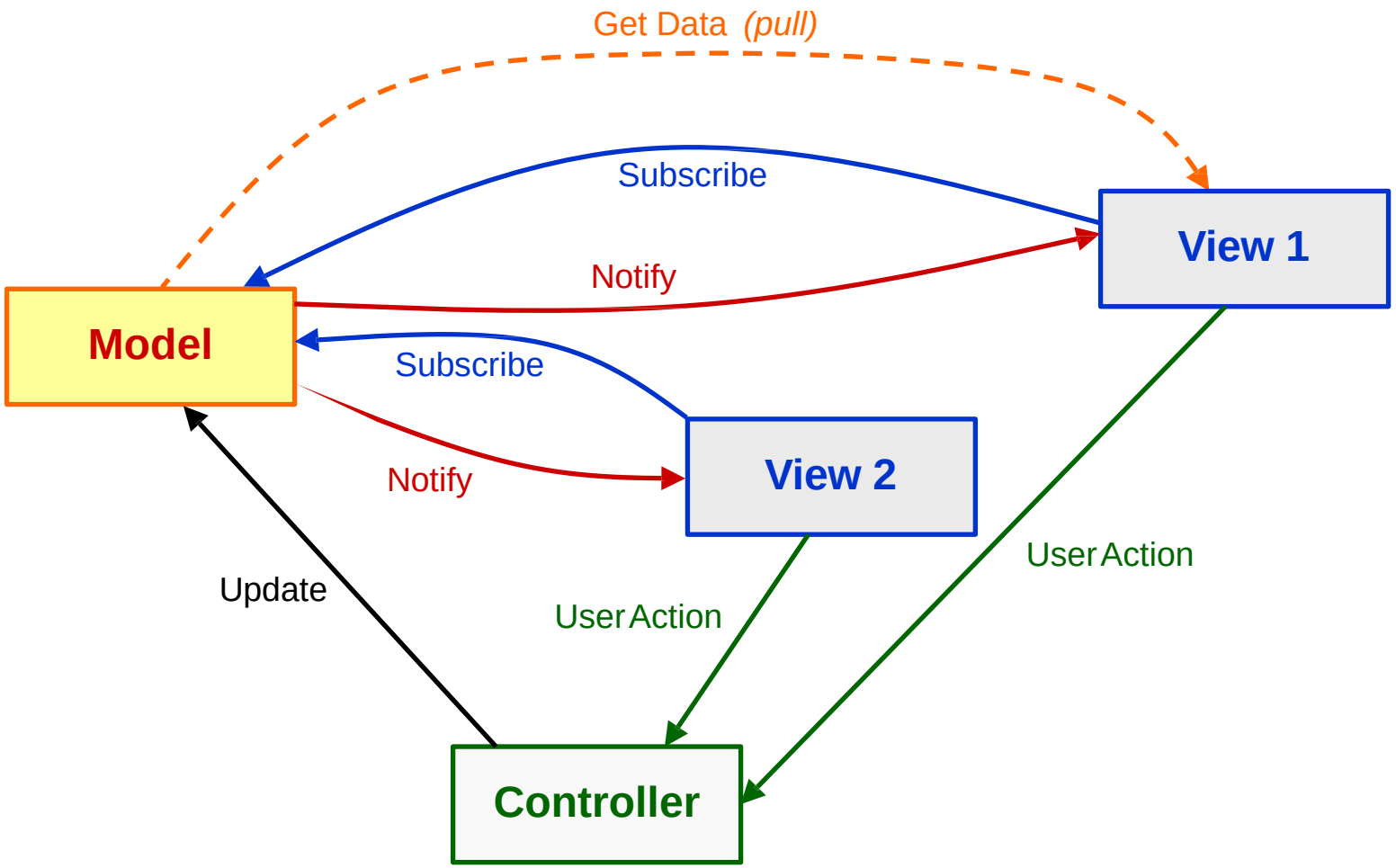
  . . .
- Le **principe général** de ce modèle asynchrone est que les vues s'inscrivent (s'enregistrent) auprès du modèle et sont informées lorsque le modèle change.
- Ce modèle asynchrone peut être implémenté de différentes manières.
- Le **niveau de granularité** des événements (changements globaux ou sélectifs) ainsi que celui des informations qui circulent (seulement les valeurs modifiées ou toutes les données du modèle) dépendent du type d'application.

Observer Pattern

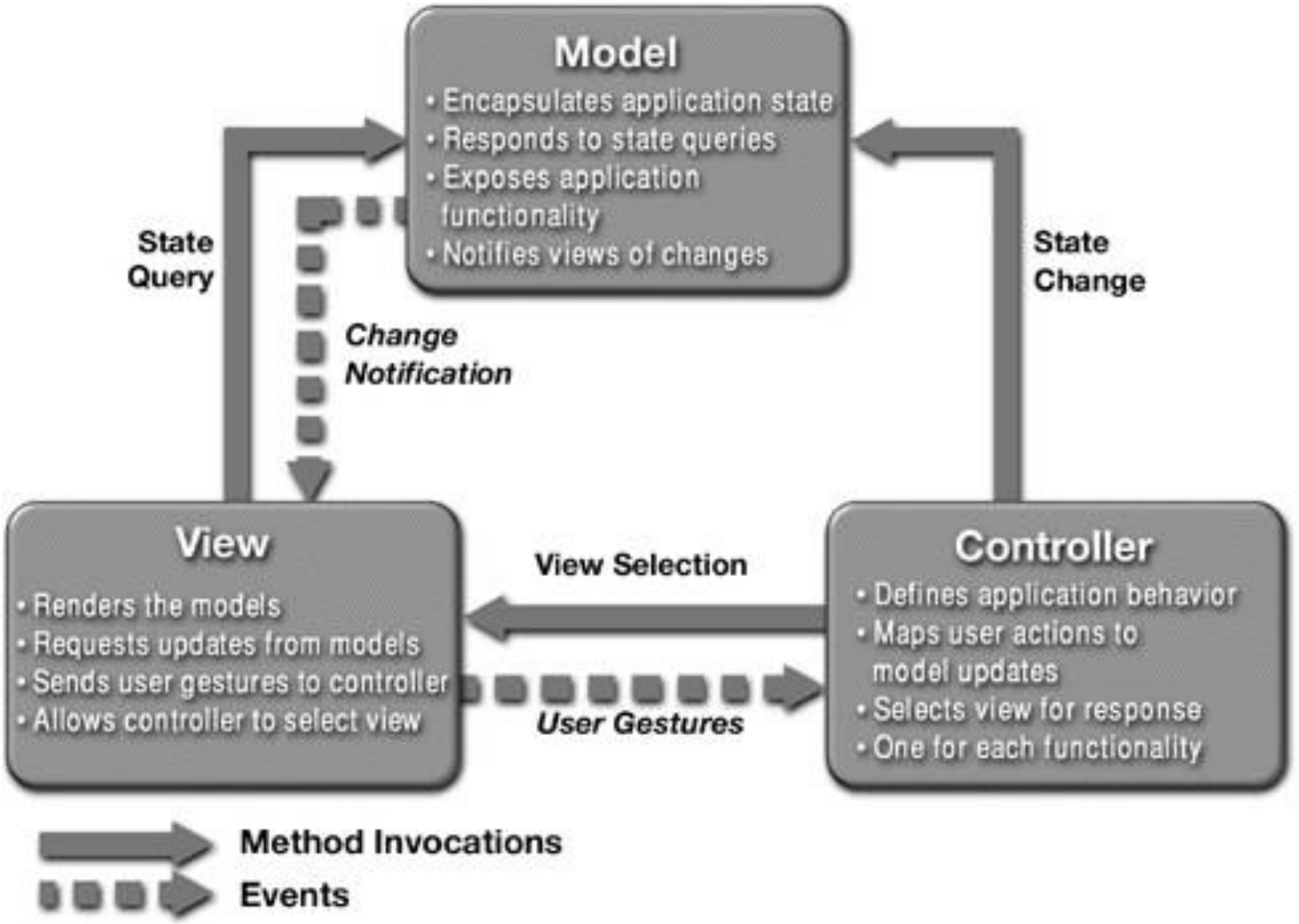



SetClock
SetAlarm
GetClock
Observable
AlarmClock
> Marion
> Max
> . . .
Notification
Notification
Notification
Observer
Marion
Observer
Max

MVC : Interactions asynchrones [1]



MVC : Interactions asynchrones [2]



Implémentation du modèle asynchrone

- En Java, il est possible d'implémenter ce modèle d'interaction asynchrone de différentes manières, notamment :
  - En utilisant la classe Observable et l'interface Observer (java.util)
  - En utilisant la technique de notification des *JavaBeans* qui est utilisée dans la librairie *Swing (EventListeners)* avec la notion de propriété liée *(Bound Property)*.

    On fera appel à la classe PropertyChangeEvent ainsi qu'à l'interface PropertyChangeListener (dans java.beans). La classe utilitaire PropertyChangeSupport peut être utile pour faciliter la mise en place de ce mécanisme (gestion de la liste des récepteurs et déclenchement des événements).
  - En utilisant la classe ChangeEvent et l'interface ChangeListener (dans javax.swing.event). La classe EventListenerList (du même package) permet de gérer la liste des récepteurs abonnés aux événements.
  - Création d'un événement spécifique en créant une sous-classe de EventObject et une sous-interface de EventListener (dans java.util) avec gestion de la liste des *abonnés*

Exemples

- Les exemples qui suivent illustrent différentes implémentations du modèle d'interaction asynchrone.
- Ils représentent une mini-application comprenant :
  - Un modèle simple qui est un compteur que l'on peut incrémenter et consulter
    - ✓ **Counter** { `incrCounter(int i); getCounter(); setCounter(int i);` }
  - Des vues triviales (simple affichage de la valeur du compteur sur la console)
    - ✓ **Viewer**
  - Des classes utilitaires pour mettre en place et gérer le mécanisme
    - ✓ Par exemple enregistrement des abonnés, récepteur d'événements (*EventListeners*)
  - Un programme principal qui modifie périodiquement l'état du modèle et qui permet de tester le fonctionnement de l'ensemble
    - ✓ **Test**...