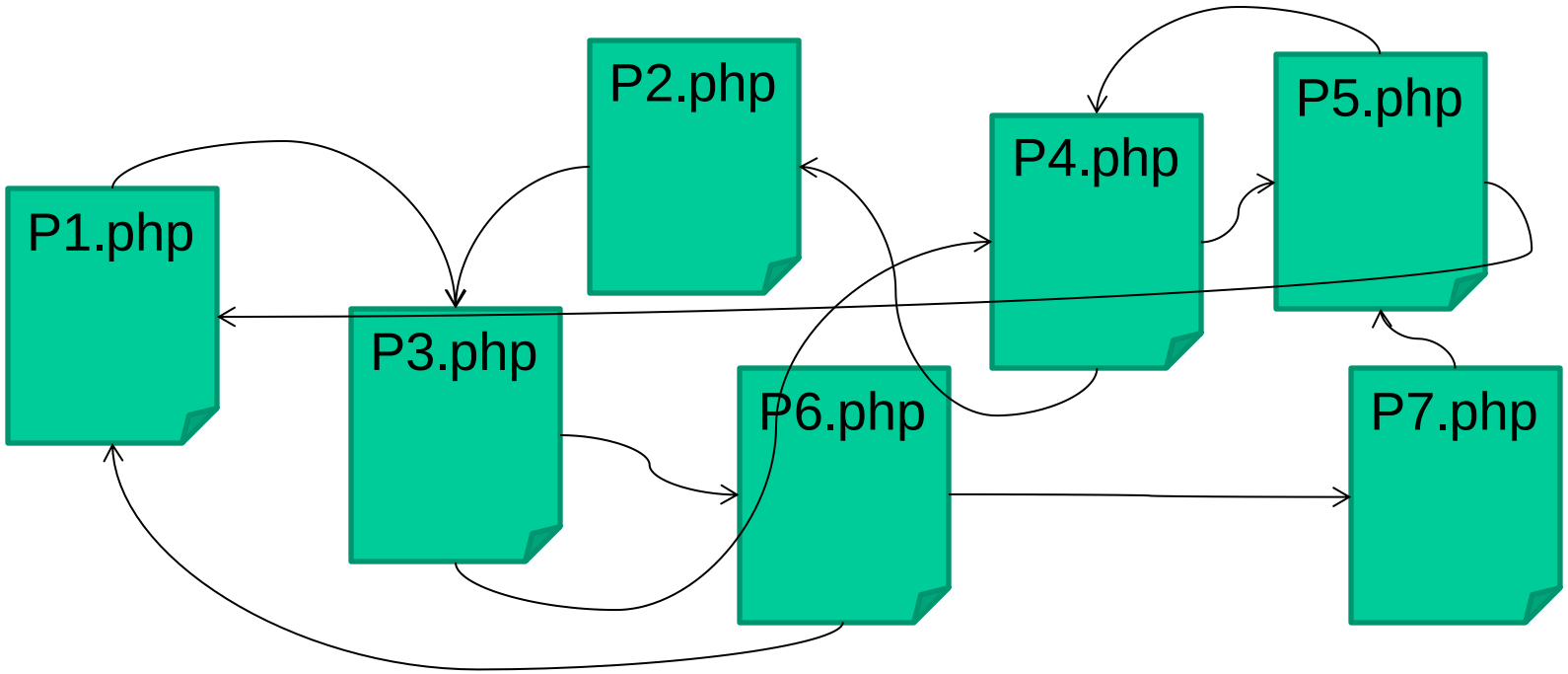
Chapitre 8
Structuration d'un projet WEB
Patron de conception MVC

Un site WEB sans structure connue

- Mélange de PHP et d'HTML
- Difficile de comprendre l'organisation du code
- Point de départ du site ?
- Fonctionnement du site ?
- Visuel mélangé avec le noyau fonctionnel



Organisation du code

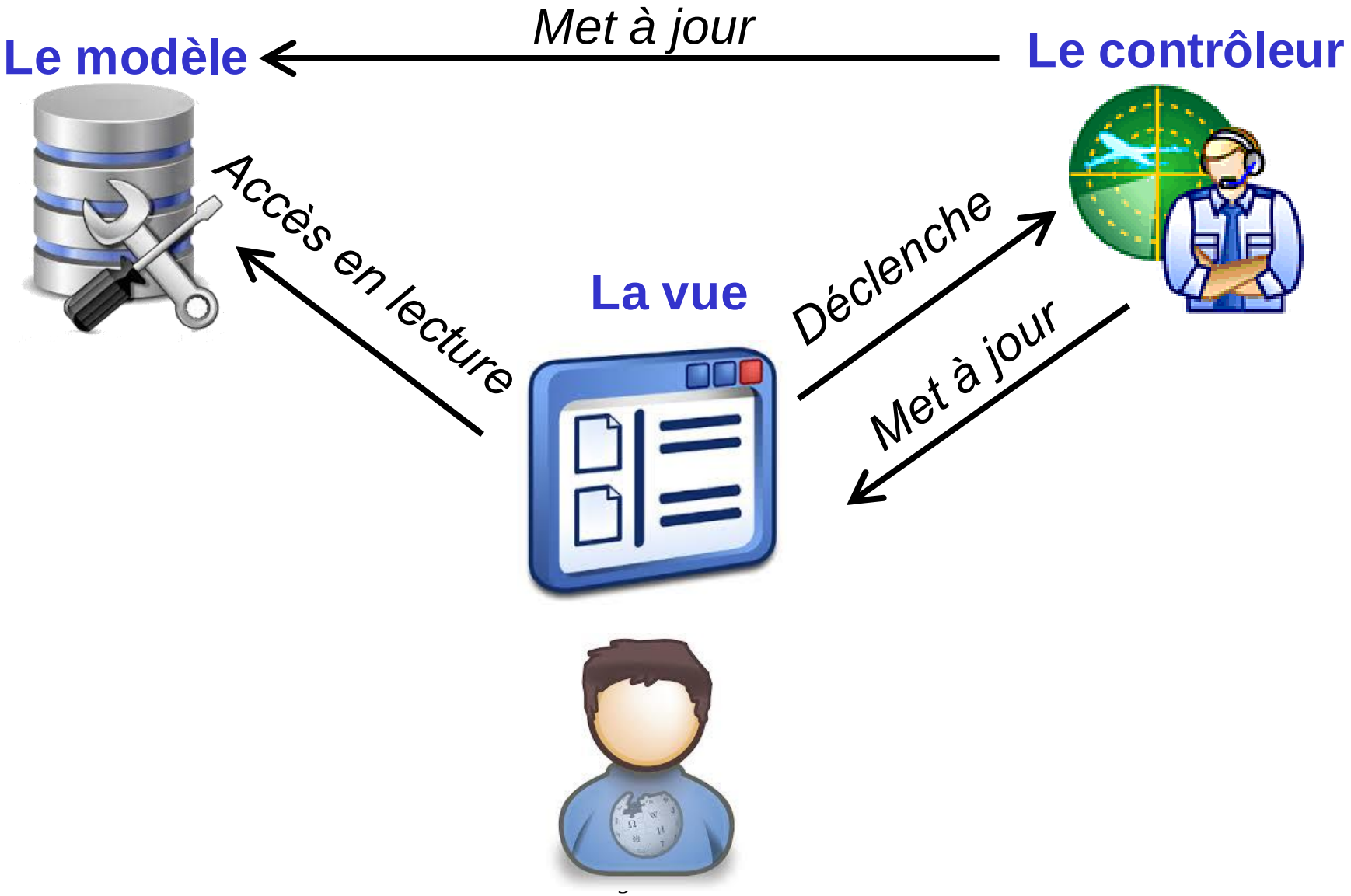
- Tout intégré (sauf CSS) :
  - Un seul fichier qui contient le programme PHP et l'affichage HTML, un fichier séparé pour le CSS
  - Pour un tout petit projet
- Bibliothèque et Modèle séparé
  - Le code général et réutilisable est dans un fichier bibliothèque
  - Le code du modèle de l'application (là ou se font les calculs)
  - Utilisation d'inclusion
- Vue séparé
  - La vue : la partie HTML produite pour l'affichage et l'interaction
  - Le HTML dans un fichier séparé, ou structuré dans plusieurs fichiers
- **Vue, Modèle et Contrôle** séparé
  - Modèle **MVC**

Solution : patron MVC

- Idée : séparer les **données**, la **présentation** et les **traitements**
- Un cadre normalisé pour structurer une application
  - Patron de conception
- MVC : des concepts, des guides pour programmer

- **M pour modèle**
  - Le noyau fonctionnel de l'application
  - Dépositaire des données et calculs
- **V pour vue**
  - Ce qui est visible à l'utilisateur
  - Pour l'interaction en entrée et sortie
- **C pour contrôleur**
  - Le chef d'orchestre de l'application
  - Déclenche les calculs du modèle et choisit les vues

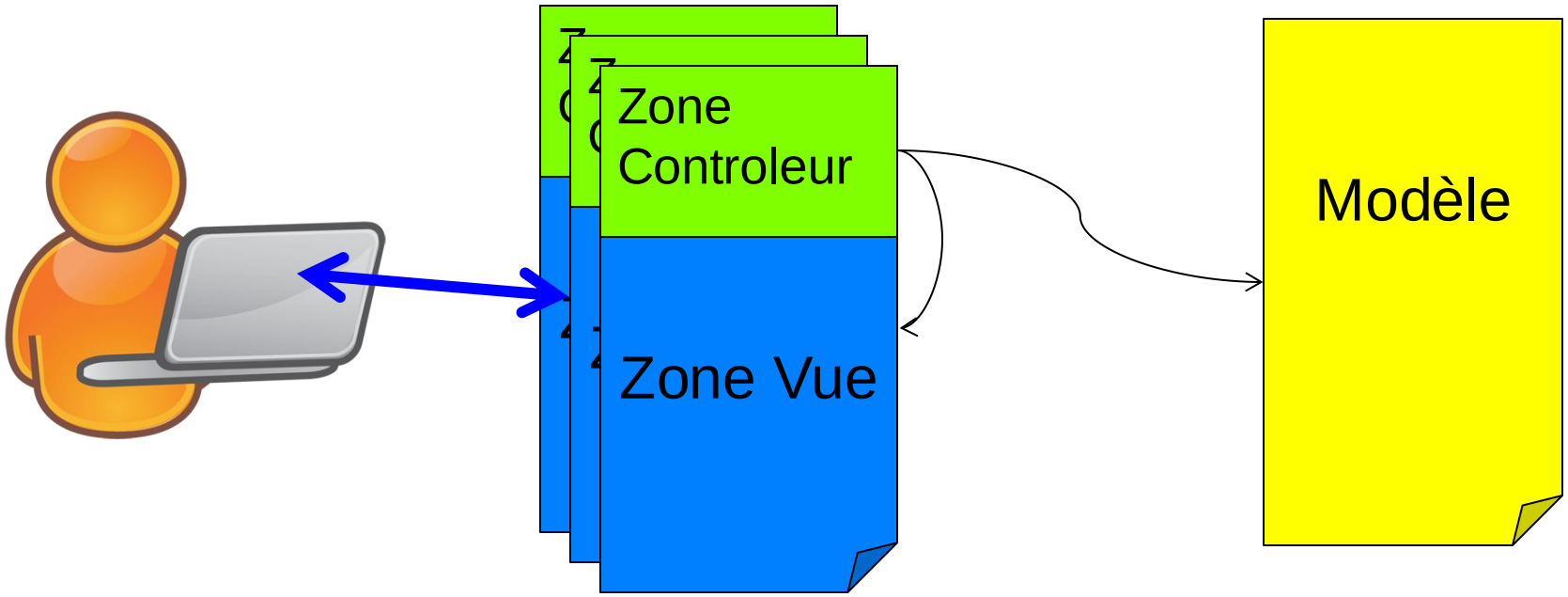
Patron : Modèle Vue Contrôleur



MVC : Structure simple

2 types de fichiers PHP

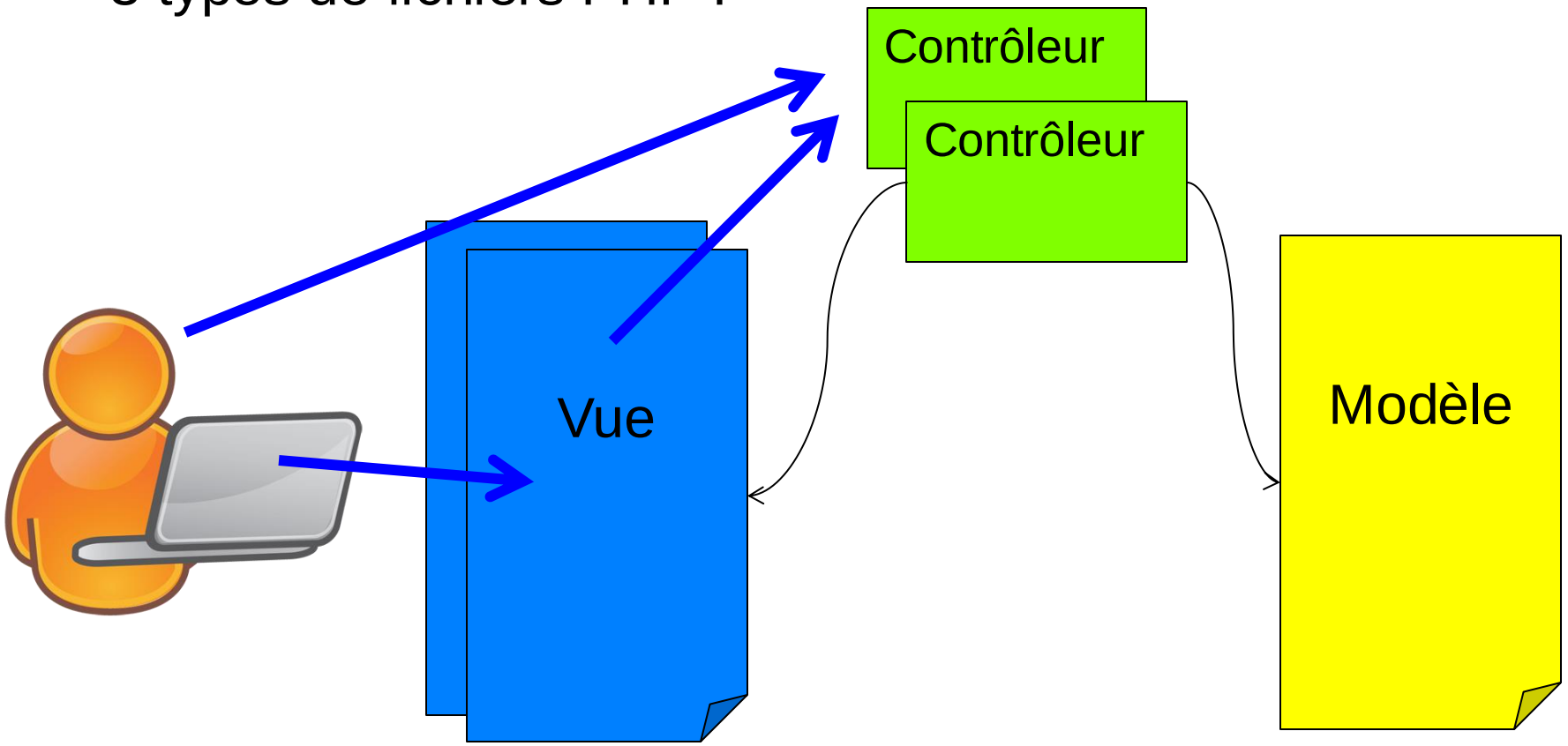
- Contrôleur + vue : liés
  - contrôle : pas le choix de la vue
- Modèle



Les vues + contrôleur de la vue

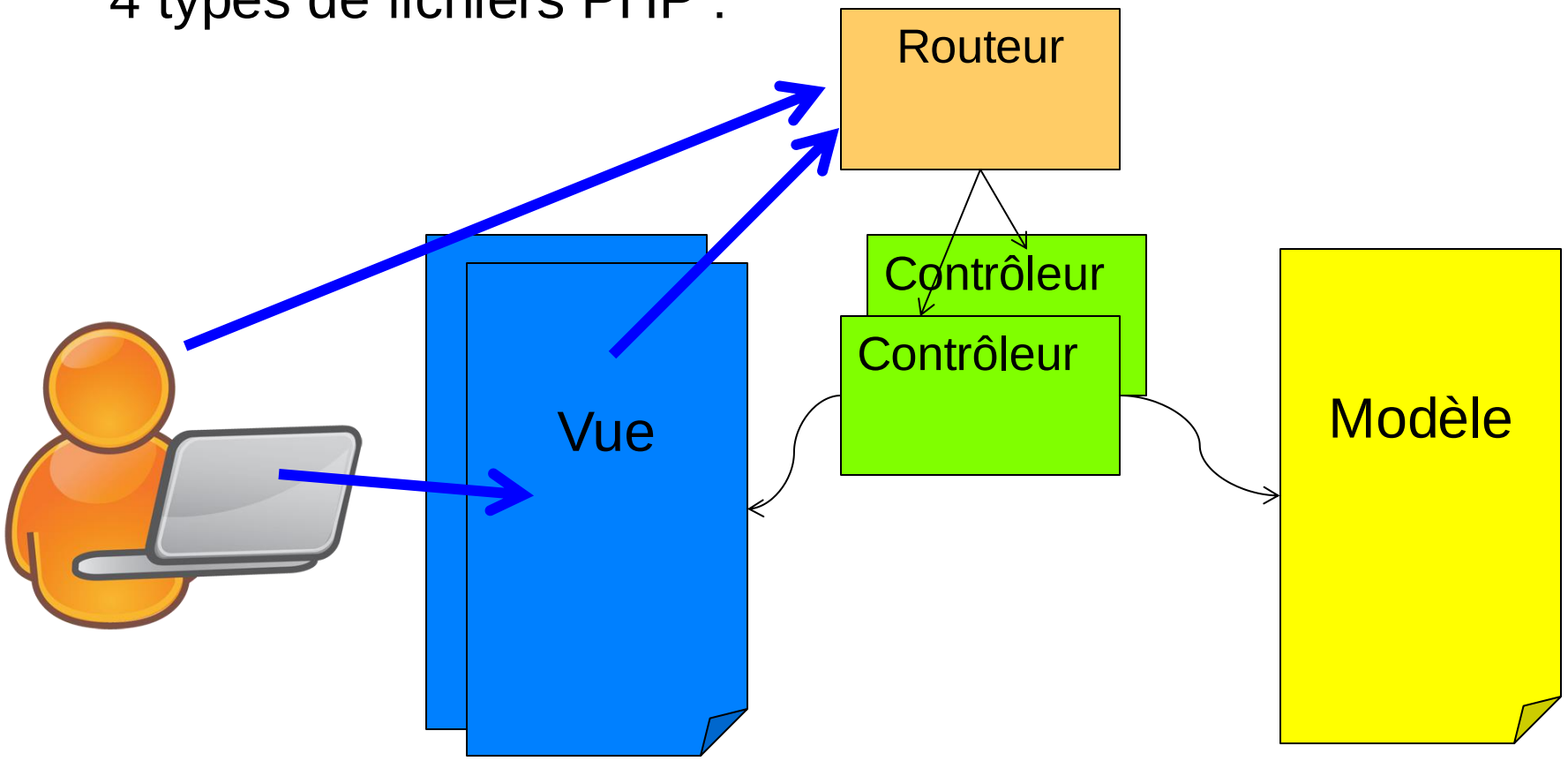
MVC : Structure complète

3 types de fichiers PHP :



MVC : Structure avec un routeur

4 types de fichiers PHP :



Choix de *tous* les frameworks

MVC : mise en place

- **Modèle** :
  - Du PHP pur (PAS DE HTML !)
  - De préférence objet (des classes)
  - Décrit tous les objets métiers de l'application
  - Besoin de Data Access Objet (DAO) pour la persistance
  - Utilise Object-Relational Mapping (ORM)
- **Vues**
  - Du HTML principalement : le PHP est dans le HTML,
  - **Sortie** : Le PHP sert à placer les données et à les collecter du modèle, des variables contiennent ce qu'il faut afficher.
  - **Entrées** : des liens (<a>) avec URL ou des formulaires
- **Contrôleur**
  - Implémente le graphe d'interaction de l'application
  - Modifie le modèle
  - Choisit l'affichage en fonction des paramètres ou de l'action

Le Contrôleur

3 parties :

1. **Récupération des données :**
   - Reconstruire à l'état de certaines variables
   - Transférer les paramètres de l'URL ou de POST dans des variables en vérifiant leur existence et en plaçant des valeurs par défaut si nécessaire.
2. **Calculs : utilisation du modèle**
   - Examiner les variables et calculer le nouvel état de l'interface
   - Lancer les calculs, utiliser le modèle (ex: la base de données)
3. **Choix de la vue**
   - En fonction de l'état de l'interaction et du résultat du calcul avec le modèle

La partie contrôle et vue

- peut être entièrement contenu dans la page
- ou bien être stockée dans un fichier PHP à part

Le modèle

- Décrit la réalité qu'il faut coder dans le système
- Résultat d'une analyse (ex: UML)

- Exemple :
  - On veut représenter dans un site WEB la liste des animaux d'un Zoo.
  - Chaque animal a un nom, un sexe, une date de naissance et appartient à une espèce.

Une analyse simple identifie deux entités :

- Un Animal
- Une Cage
- Tout animal est dans une cage
- Une cage peut contenir plusieurs animaux

La persistance des données en PHP : le **DAO**

- Notion de modèle
  - Description des données et de opérations sur ces données
  - Pas de présentation
  - Encapsulation par des méthodes
- Persistance des données
  - Utilisation d'une classe **Data Acces Object (DAO)**
  - Permet de représenter l'accès à la base de données par un objet
- BD relationnel ⇔ PHP classe et objet
  - Une table correspond à une classe
  - Coder le passage de l'un à l'autre : Object-Relational Mapping (ORM)
  - Créer les méthodes **CRUD** :
    - *Create, Read, Update, Delete*
  - Dans les framework : code produit automatiquement

Object-Relational Mapping (ORM)

Une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle (Wikipedia)

- Une classe par table (en général …)
- Une classe DAO
- Un objet DAO (et 1 seul !) qui représente la BD
- Des méthodes CRUD dans chaque classe

- => Le rôle des frameworks
  - ex: Doctrine de Symfony

MVC: codage d'une vue

- Principalement du HTML
  - Le PHP pour placer les valeurs dans le HTML
- Charger une vue
  - Un simple "**include**" !!
- Utilisation possible de templates
- Décomposer la vue en sous vues
  - Exemple : un menu
  - Identifier les moyens de passer des informations
- Codage du paramétrage d'une vue avec MVC
  - Utiliser des variables pour placer les valeurs dans la vue
  - Accès en lecture au modèle

Coder une Vue

*Patron MVC : la vue est le moyen d'interaction avec l'utilisateur, c'est aussi **l'objet logiciel** qui permet une liaison avec le contrôleur et le modèle.*

- Exemple de codage minimum d'un vue :
  - En HTML si la vue n'est pas dynamique.
  - Dur HTML avec du PHP pour placer les éléments de la vue.
- "montrer" la vue consiste à faire un *include*. Le code HTML et PHP inclus dans la portée de l'endroit ou se fait l'inclusion.

- Attention : si l'inclusion est au niveau global
  - Toutes les variables globales sont accessibles à la vue
    - => sécurité du code ??

Coder une vue : mise en place

```
<HTML>
<!-- Fichier de la vue : unePersonne.view.php -->
<h1> Informations sur la personne </h1>
<ul>
  <li>Nom : <?= $nom ?> </li>
  <li>Prénom : <?= $prenom ?> </li>
  <li>Véhicule : <?= $vehicule->marqueEtModele() ?> </li>
</ul>
```

- Accès à des valeurs ou à des objets du modèle (ex: Vehicule)

MVC en pratique

- **Répertoire model**
  - Les classes du modèle
  - La base de donnée
- **Répertoire view**
  - Les fichier HTML/PHP des vues
  - Les sous vues : exemple le menu
- **Répertoire controller**
  - Fichiers PHP : nom de fichier comporte un verbe
- **Répertoire data**
  - Les données, la BD
- **Répertoire public**
  - tout ce qui doit être **accessible directement sans passer par le routeur**
  - Ex: le style CSS, les images incluses, le Javascript.
- **index.php** : le routeur qui charge le bon contrôleur, le seul accessible

Le routeur (index.php)

```php
// Calcul du routage : récupération de la route, le controleur à activer
$ctrl = $_GET['ctrl'] ?? 'main'; // Par défaut lance contrôleur 'main'

// Sécurité : liste des contrôleurs possibles
// Cette liste permet d'être sûr de ne pas charger un fichier inconnu
const CTRLS = array('main','getName','getLanguage');

// Sécurité : vérification que la route est correcte
if (! in_array($ctrl,CTRLS)) {
        // Ouvre une page d'erreur
        $error = 'Mauvais controleur : "'.$ctrl;
        require_once('view/error.view.php');
        exit(0);
}
// Calcule le chemin vers le fichier du contrôleur
$path = 'controler/'.$ctrl.'.ctrl.php';
// Charge le contrôleur
require_once($path);
```

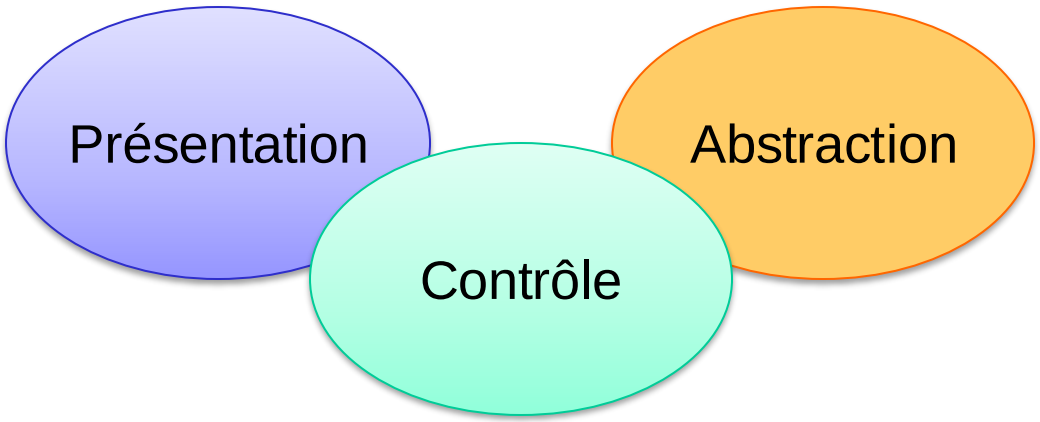
MVC : Avantage / Inconvénients

- Propose une architecture connue, simple et claire.
- La modification des traitements (Modèle) ne change pas la vue.
  - Ex: passer d'un stockage fichier à une base de donnée
- Permet le développement indépendant : vue et modèle
  - Vue : un designer
  - Modèle : un informaticien
  - Contrôleur : de graphe d'interaction
- Facilite la maintenance et l'évolution du logiciel

- Mais : un effort initial pour comprendre le modèle
- Trop complexe pour de très petits projets
- Risque d'émiettement du code, perte de visibilité globale
- Utiliser un template, inclusions => ralentissement de l'exécution

Patron PAC : Présentation, Abstraction, Contrôle

- Introduit par J Coutaz (Grenoble 1987)
- Modèle abstrait d'architecture logicielle pour les interfaces homme-machines
- Relatif à la notion d'objet : 3 objets qui communiquent
- Hiérarchisé et liaisons entre les abstractions
- Lien avec MVC :
  - Présentation : une Vue mais "passive", pas d'accès au modèle



Syntaxe alternative pour les vues

```
<?php foreach ($this->questions as $q) : ?>
  <tr>
    <td>
     <input type="text" name="titre" value="<?= $q->titre?> ">
    </td>
   </tr>
<?php endforeach; ?>
```

- Pour faciliter la syntaxe des vues
  - le mélange HTML et PHP pour les boucles
  - limiter le PHP au minimum dans les boucles
- Pour le : if, while, for, foreach, switch
  1. ajouter le caractère ':' à la place de '{' avant la balise de fermeture '?>
  2. ajouter la balise <?php endif; ?> (ou endwhile, endforeach, endswitch)

Framework (micro framework)

- Exemple de framework PHP
  - CakePHP
  - CodeIgniter : libre
  - Symfony
  - Zend Framework
  - Drupal : modèle PAC
  - …..
- Principe d'inversion de contrôle
  - C'est le framework qui a toujours "la main »
- En TP : propose un "micro" framework
  - un routeur simple dans index.php
  - Une seule URL : index.php
  - Indiquer dans la query string le controleur : ?ctrl=XXXX
  - une classe view pour ... faire jolie

Coder une classe Vue

- Patron MVC : la vue est le moyen d'interaction avec l'utilisateur, c'est aussi **l'objet logiciel** qui permet une liaison avec le contrôleur et le modèle.
- Exemple de codage minimum d'un objet vue :
  - une classe (**View**)
  - une méthode pour "montrer" la vue (**display($path)**)
  - une représentation des valeurs de la vue (dans le tableau **param** de l'objet vue)
- "montrer" la vue consiste à faire un *require*. Le code HTML et PHP inclus dans la méthode *display* est dans la portée de cette méthode.
  - **aucune variable globale accessible** => sécurité du code
  - les **valeurs de la vue** sont des **variables locales** de la méthode **display** de la vue
- Utilisation de l'indirection **$$key** pour créer de nouvelles variables

Coder une classe vue : exemple

```
class View {
  // Paramètres de la vue, dans un tableau associatif
  private array $param;

  // Constructeur d'une vue
  function __construct() {
    // Initialise un tableau vide de paramètres
    $this->param = array();
  }
```

Coder une classe vue : exemple

```
class View {

...

// Ajoute un paramètre à la vue
  // Il n'y a aucune contrainte de type pour la valeur
  // Cela peut être par exemple un objet du modèle.
  function assign(string $varName,$value) {
    $this->param[$varName] = $value;
  }
```

Coder une classe vue : exemple

```
function display(string $filename) {
    // Ajoute le chemin relatif vers le fichier de la vue
    $filePath = "../view/".$filename;

    // Parcourt toutes les paramètres de la vue
    // Crée des variables locales
    foreach ($this->param as $key => $value) {
          $$key = $value;
    }

    // Inclusion de la vue
    require($filePath);
    // Apres cela le PHP est terminé, plus rien ne s'exécute
    exit(0);
  }
```

Coder une classe vue : utilisation dans un contrôleur

```
// Création de l'objet vue
$view = new View();

// Place les valeurs à afficher dans la vue
$view->assign("nom",$p->getNom());
$view->assign("prenom",$p->getPrenom());
// place un objet du modèle accessible à la vue
$view->assign("vehicule",$p->getVehicule());

// Affiche la vue
$view->display("unePersonne");
```

- Placer les valeurs ou objets à afficher dans la vue
- Valeur : la vue est indépendante du modèle
- référence sur un objet du modèle : crée une dépendance

Coder une classe vue : mise en place

```
<HTML>
<!-- Fichier de la vue : unePersonne.view.php -->
<h1> Informations sur la personne </h1>
<ul>
  <li>Nom : <?= $nom ?> </li>
  <li>Prénom : <?= $prenom ?> </li>
  <li>Véhicule : <?= $vehicule->marqueEtModele() ?> </li>
</ul>
```

- Accès obligatoire par des variables locales
  - Aucune autre variable n'est visible
- Accès à des valeurs ou à des objets du modèle (ex: Vehicule)

MVC avec router en pratique

- Plus besoin de désigner le fichier PHP à exécuter :
  - toujours le même : index.php
- Besoin d'indiquer la route : GET et le contrôleur
  - indiqué dans la query string (ex: ?ctrl=monControle)
  - pas pratique pour le formulaire
    - mode GET : pas d'accès direct à la query string
      - besoin de passer par un input de type "hidden"
    - mode POST : ok query string possible
      - paramètre non visibles dans l'URL
- Solution des frameworks
  - normaliser les URL : APP/ACTION/DATA1/DATA2/ETC...
  - Paramétrer le serveur pour faire de lé réécriture d'URL

  APP/ACTION/DATA1/DATA2 => index.php?ctrl=ACTION&val1=DATA1& ...

MVC à retenir

- Patron de conception très anciens mais encore efficace
- programmation WEB coté serveur :
  - Contraintes due à la "rotation" des pages
  - Plusieurs solutions possibles : simple (1 page), ou complexe (site)
- Solution à base d'un routeur : augmente la sécurité
  - un seul point de passage : index.php
  - tout le reste des fichiers non accessibles (sauf public)
  - mettre des contrôles et limitations dans le router
  - Gérer les sessions de préférence dans le router
- Utiliser la configuration du serveur (.htaccess)
  - pour interdire tout accès direct (sauf public)
  - pour faire de la ré-ecriture et uniformiser les URL