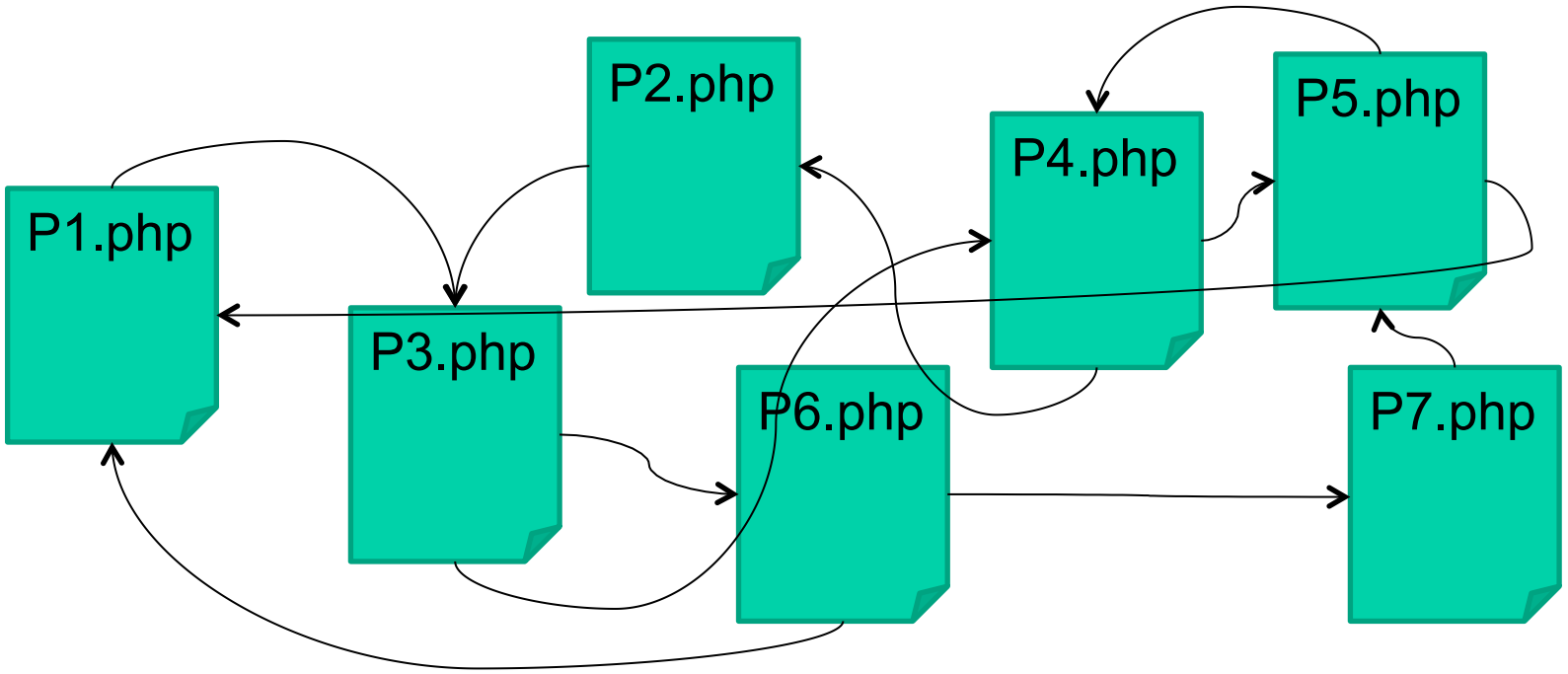
Chapitre 10
Structuration d'un projet WEB
MVC et PAC

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

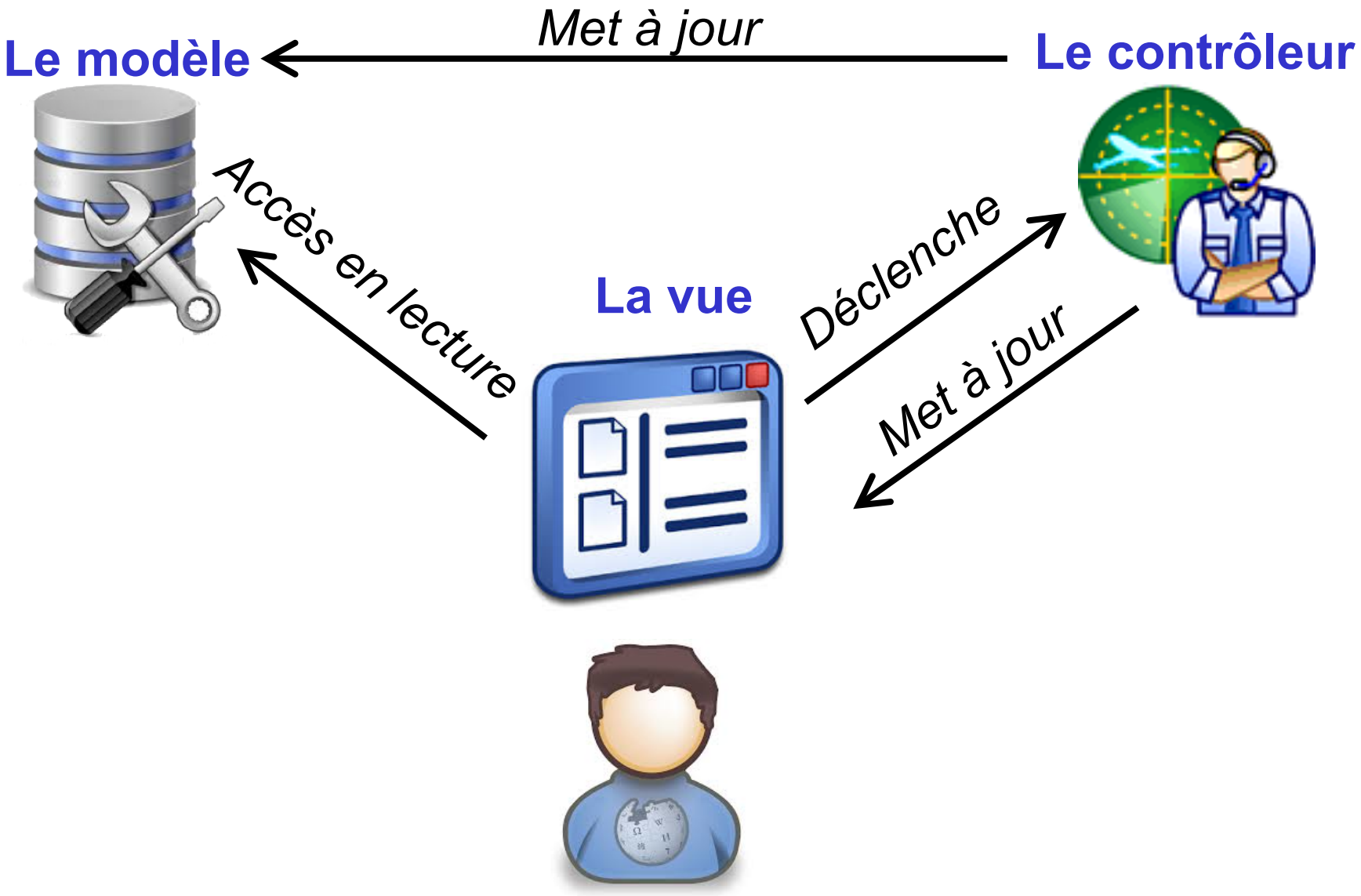
- Tout intégré (sauf CSS) :
  - Un seul fichier qui contient le programme PHP et l'affichage HTML, un fichier séparé pour le CSS
  - Pour un tout petit projet
- Bibliothèque et Modèle séparé
  - Le code général et réutilisable est dans un fichier bibliothèque
  - Le code du modèle de l'application (là 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

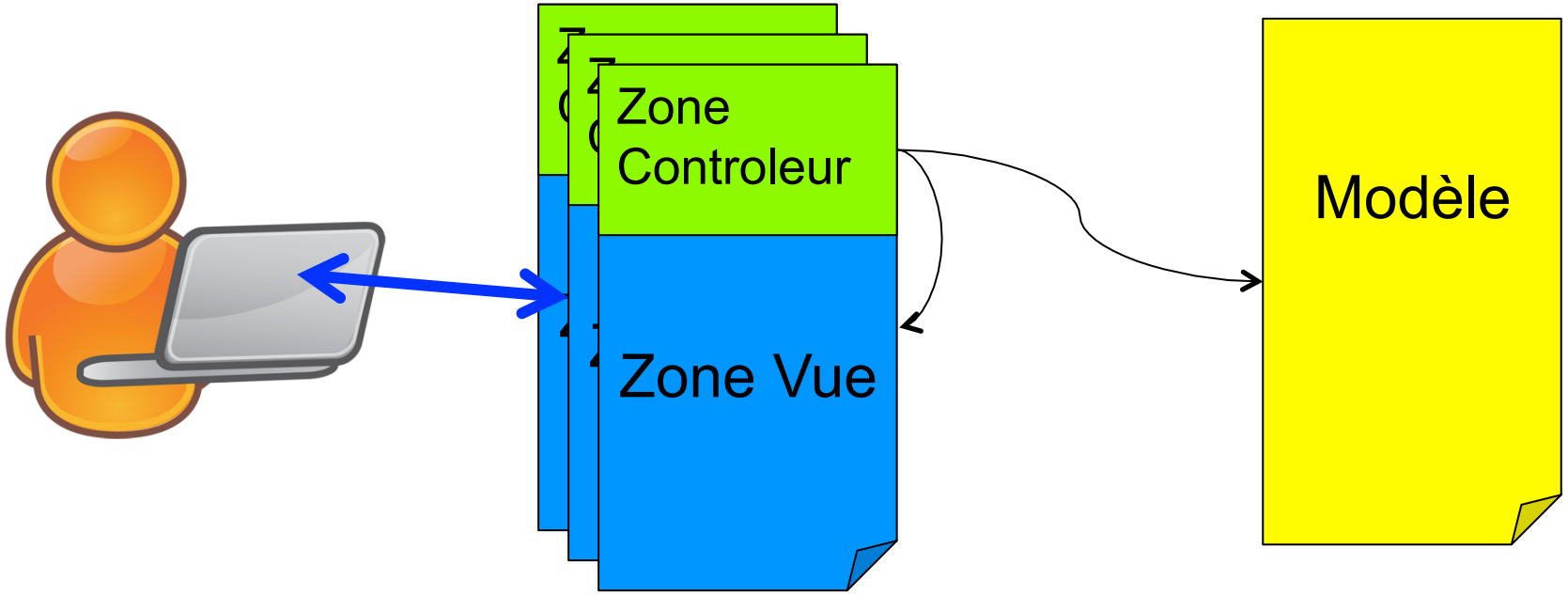
Patron : Modèle Vue Contrôleur



MVC : Structure simple (type 1)

2 types de fichiers PHP

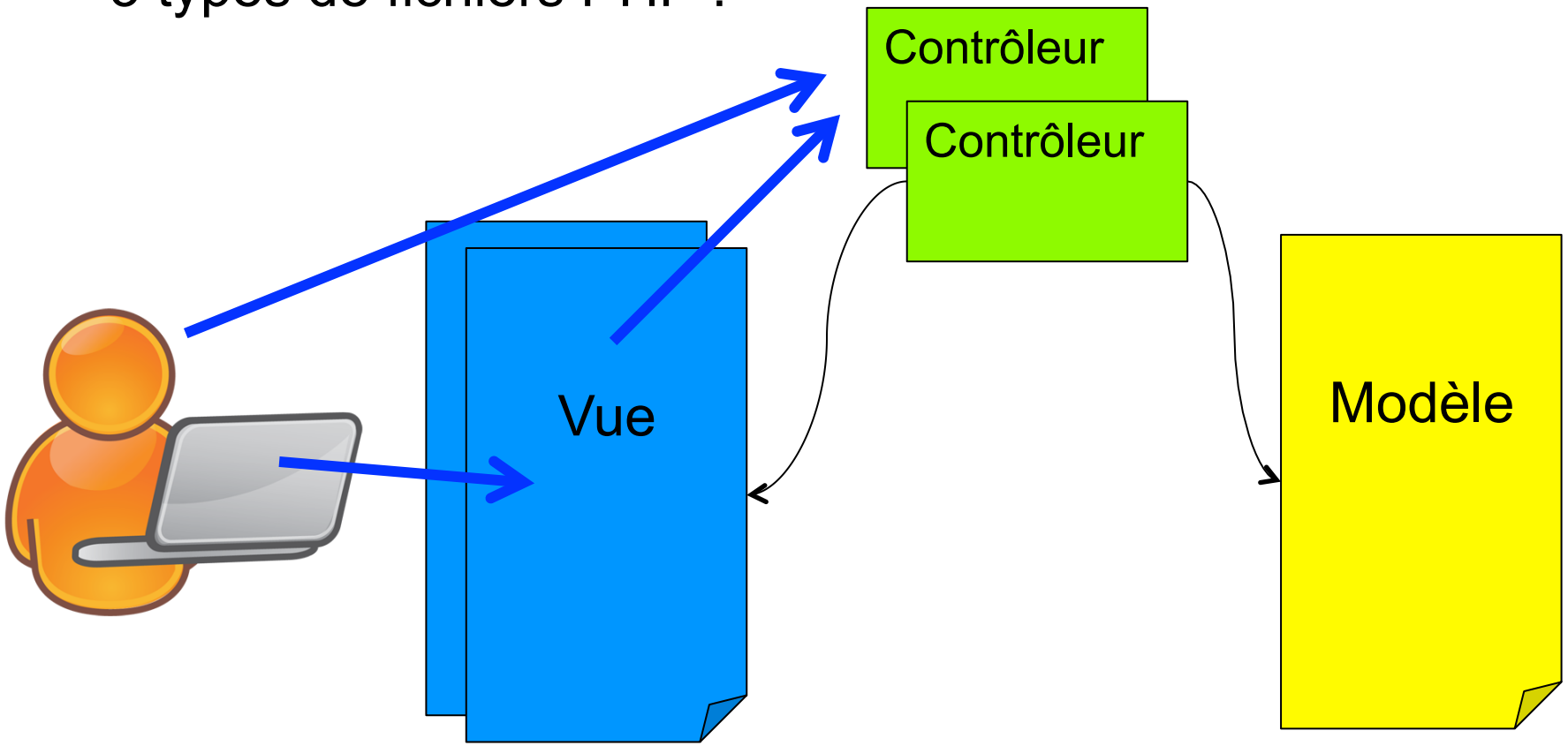
- Contrôleur + vue : liés
  - contrôle : pas le choix de la vue
- Modèle



Les vues + contrôleur de la vue

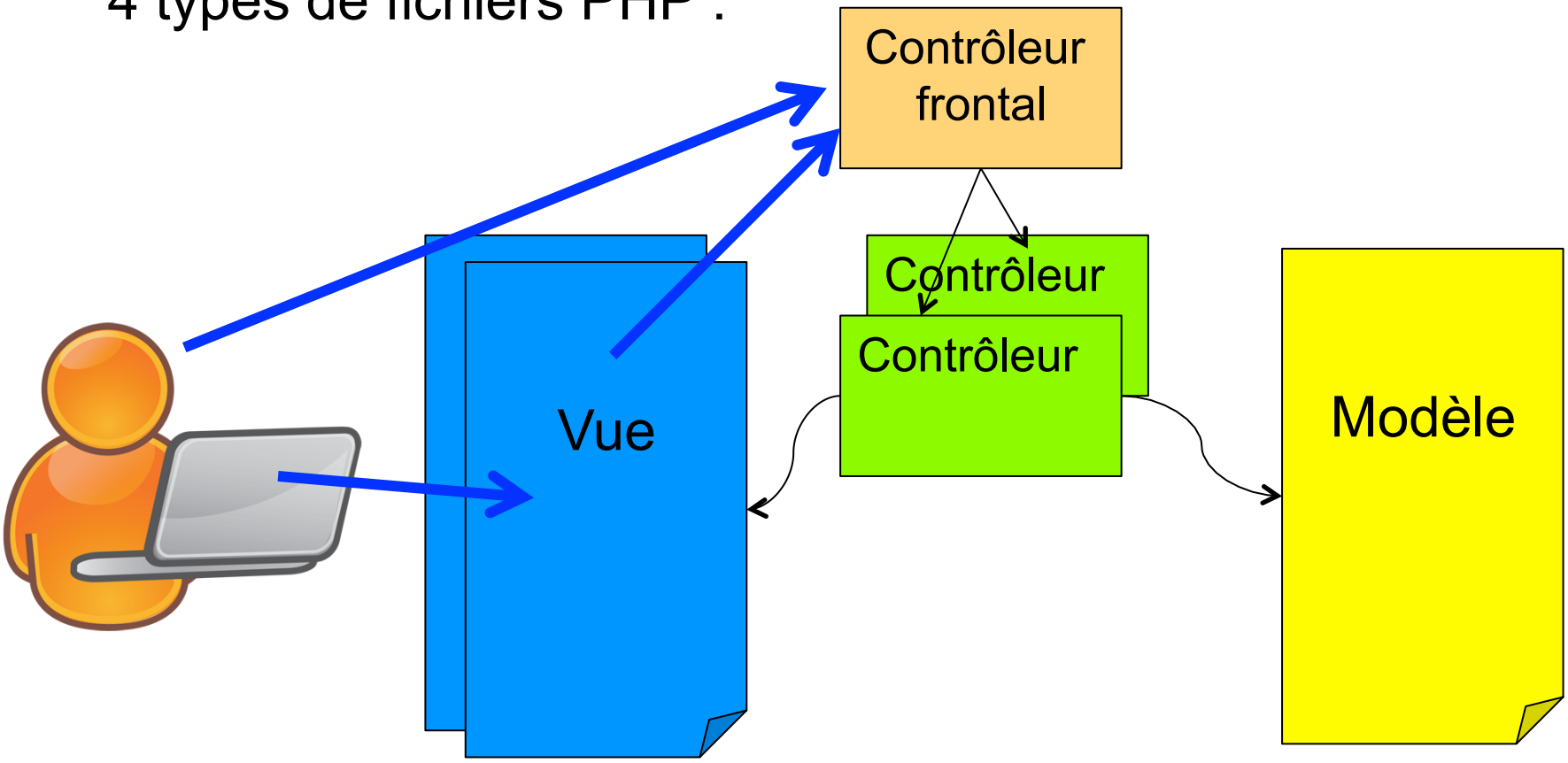
MVC : Structure complète (type 2)

3 types de fichiers PHP :



MVC : Structure avec méta-contrôleur (Type 3)

4 types de fichiers PHP :



MVC : mise en place

- **Modèle** :
  - Du PHP pur (PAS DE HTML !)
  - De préférence objet
  - Décrit tout 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 (ex: **$DATA**)
  - **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 (type 1)
- bien être stockée dans un fichier PHP à part (type 2 et type 3)

Le modèle

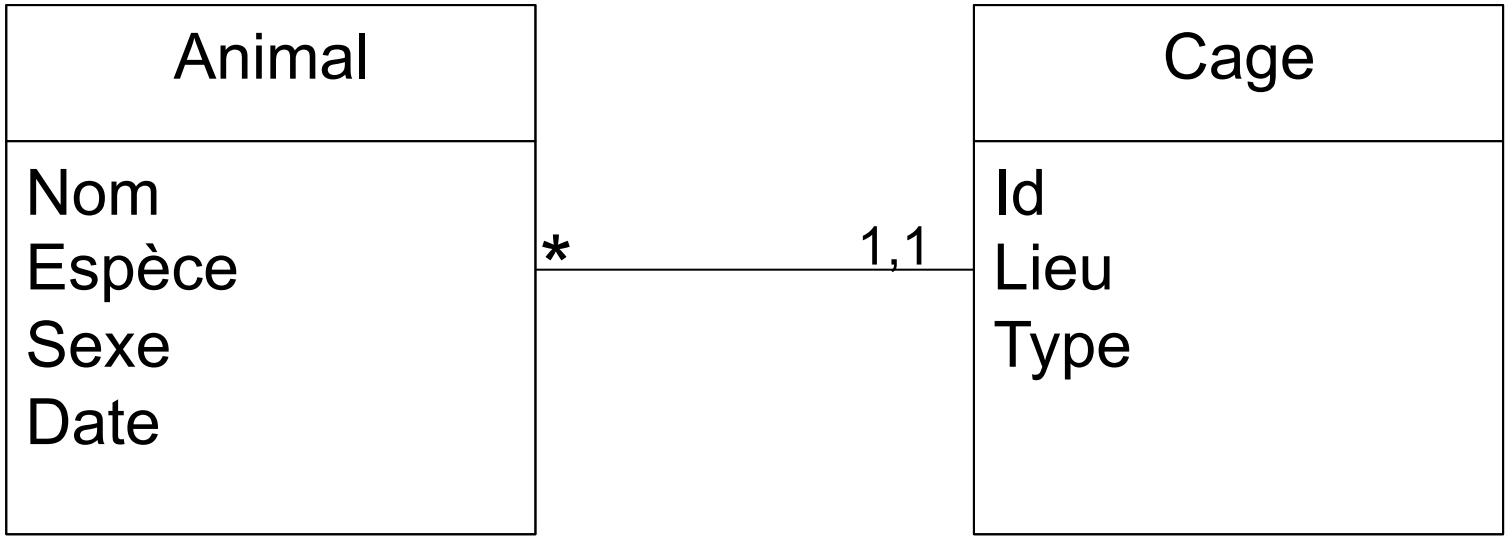
- Décrit la réalité qu'il faut coder dans le système
- Résultat d'une analyse (ex: UML)

- Exemple :
  - On veux 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

Modèle conceptuel des données



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 donnée par un objet
- Organisation
  - Dans un ou plusieurs fichiers
  - Une table correspond à une classe
  - 1 classe par entité + 1 classe DAO pour la persistance
  - Créer si besoin les méthodes **CRUD** :
    - *Create, Read, Update, Delete*

Classe DAO : constructeur

```
function __construct() {
 try {
       $this->db = new PDO($this->database);
       if (!$this->db) { die ("Database error"); }
     } catch (PDOException $e) {
      die("PDO Error :".$e->getMessage()."$this->database\n");
     }
}
```

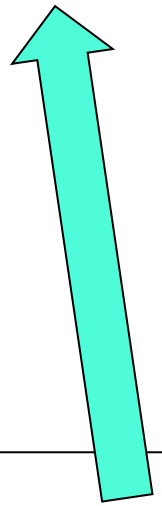
- Attribut privés pour la base de donnée
- Constructeur : ouvre la BD

Récupérer un objet

```
$req="SELECT * FROM animal WHERE nom='$nom'";

$sth=$dbh->query($req);
$result=$sth->fetchAll(PDO::FETCH_CLASS,'Animal');

//affichage nom de la 2ème ligne
$obj=$result[1];
echo $obj->nom;
```



- PDO : directement **un tableau d'objets** de la bonne classe !

Constructeur du modèle

Attention :

- un seul constructeur en PHP
  - Pas de surcharge du constructeur
- ORM avec **`PDO::FETCH_CLASS`**
  - Le constructeur est appelé **après** la création de l'objet et la mise à jour de ses attributs venant de la base de données
- Solutions à ce problème
  1. Pas de constructeur
  2. Utiliser **`PDO::FETCH_INTO`** avec un objet créé par **`new`**
  3. Créer un constructeur avec un nombre variable de paramètres et ne **PAS MODIFIER LES ATTRIBUTS** si le constructeur est appelé sans paramètre

Classes DAO : méthodes CRUD 'Read'

```
function getAnimal($nom) {
  try {
    $r = $this->db->query("SELECT * FROM animal WHERE nom='$nom'");
     $res = $r->fetchAll(PDO::FETCH_CLASS,'Animal');
       } catch (PDOException $e) {
                die("PDO Error :".$e->getMessage());
       }
       if (sizeof($res)==0) {
                die("No value found for '$nom'\n");
       }
       return $res[0];}
```

- Une méthode par cas d'utilisation.
- Tester tous les cas d'erreurs : facilite la mise au point

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 la classe DAO
- Des méthodes pour faciliter l'accès objet dans les classes par tables

- => 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 une variable globale toujours la même : **$data[]**
  - Accès en lecture au modèle
- Plus strict (Patron **PAC**)
  - Utiliser un objet : toujours le même : $data, les attributs correspondent aux éléments à, placer dans la vue

MVC en pratique (type 2)

- **Répertoire model**
  - Les classes du modèle
  - La base de donnée
- **Répertoire view**
  - Les fichier HTML/PHP des vues
  - La mise en forme par CSS
  - Les images pour le design
  - Les sous vues : exemple le menu
- **Répertoire controler**
  - Fichiers PHP : nom de fichier comporte un verbe
  - Choisir un contrôleur par défaut et le placer dans index.php à la raciné du projet :

```
<?php  header('Location: '.'Controleur/afficher_animaux.php'); ?>
```

- Répertoire **data**
  - Les données autres, comme les images, etc

- **Démo du jeux**

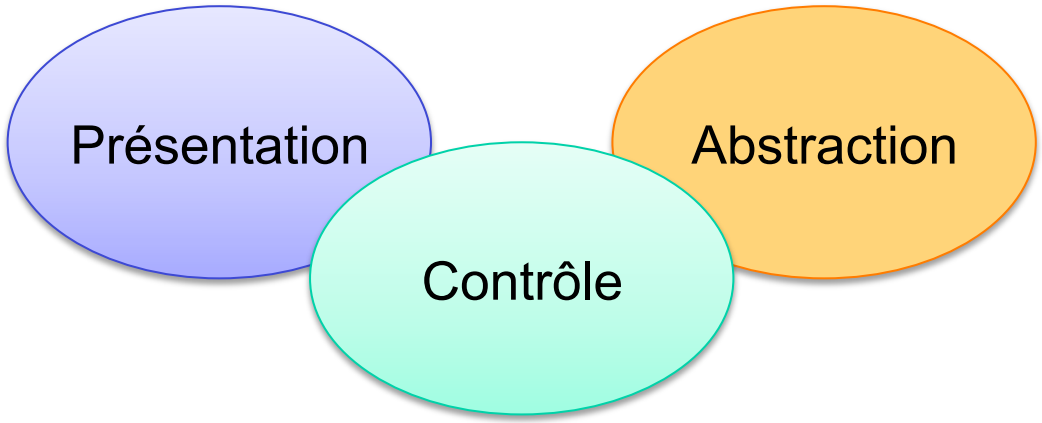
MVC : Avantage / Inconvénients

- 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



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"