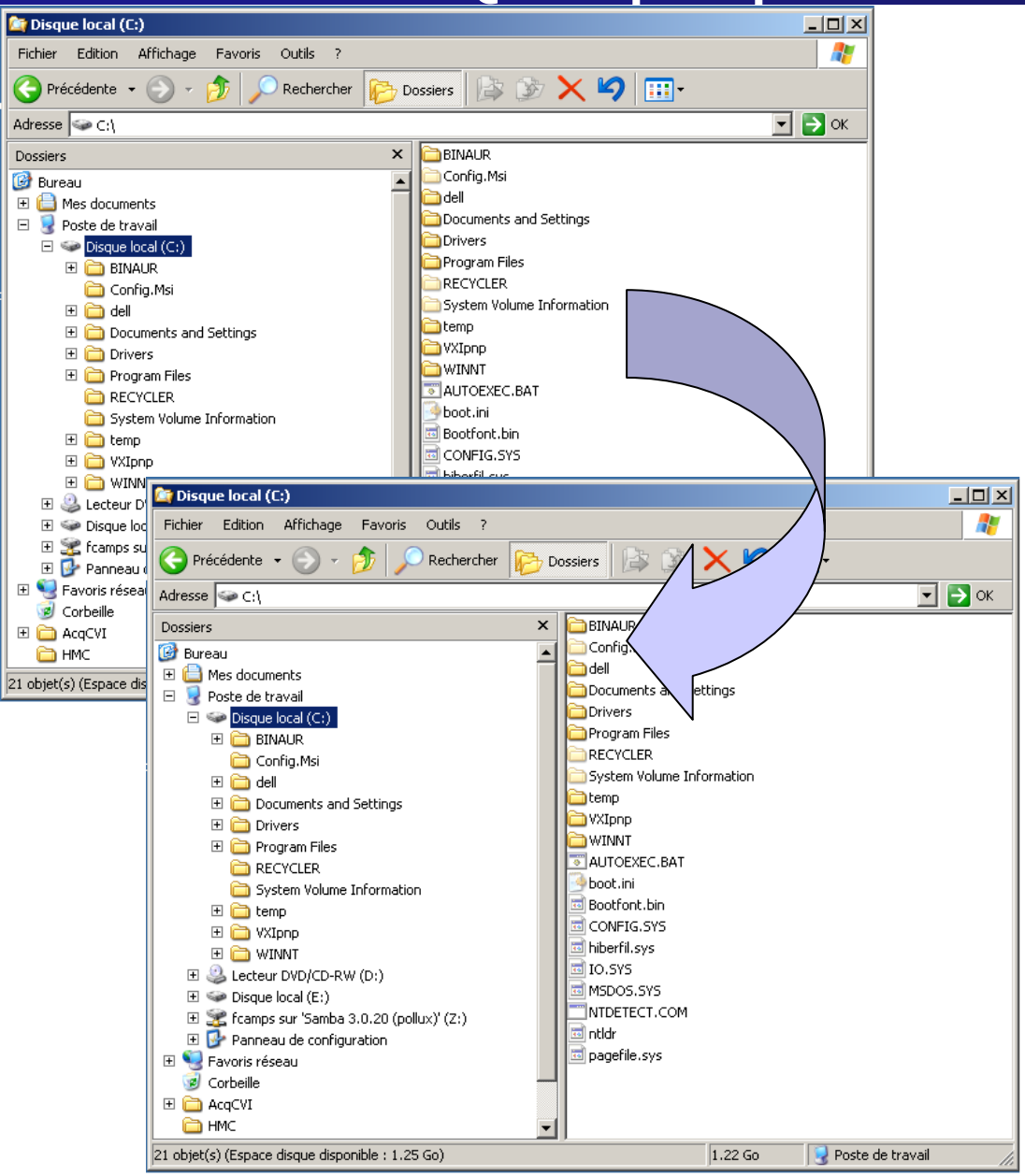
Briques de base pour la conception

Que propose une IHM ?

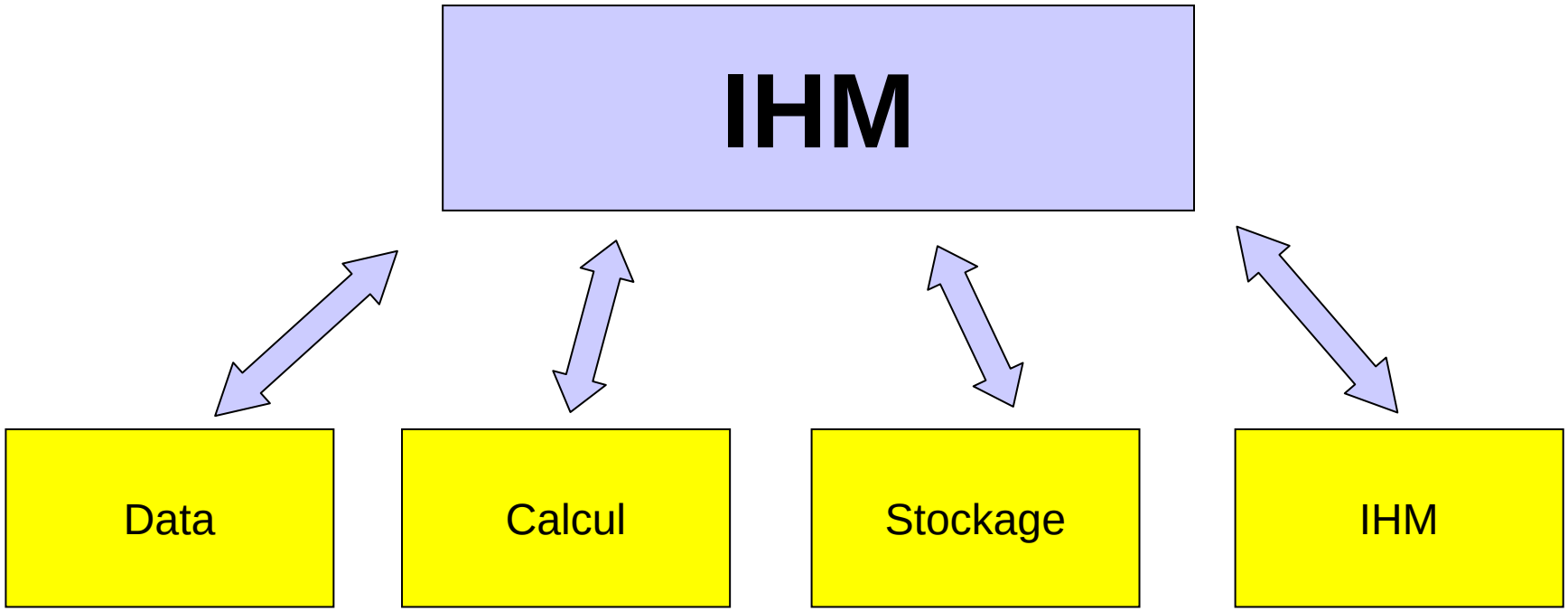


Objet graphique qui facilite la manipulation des données ou des systèmes :

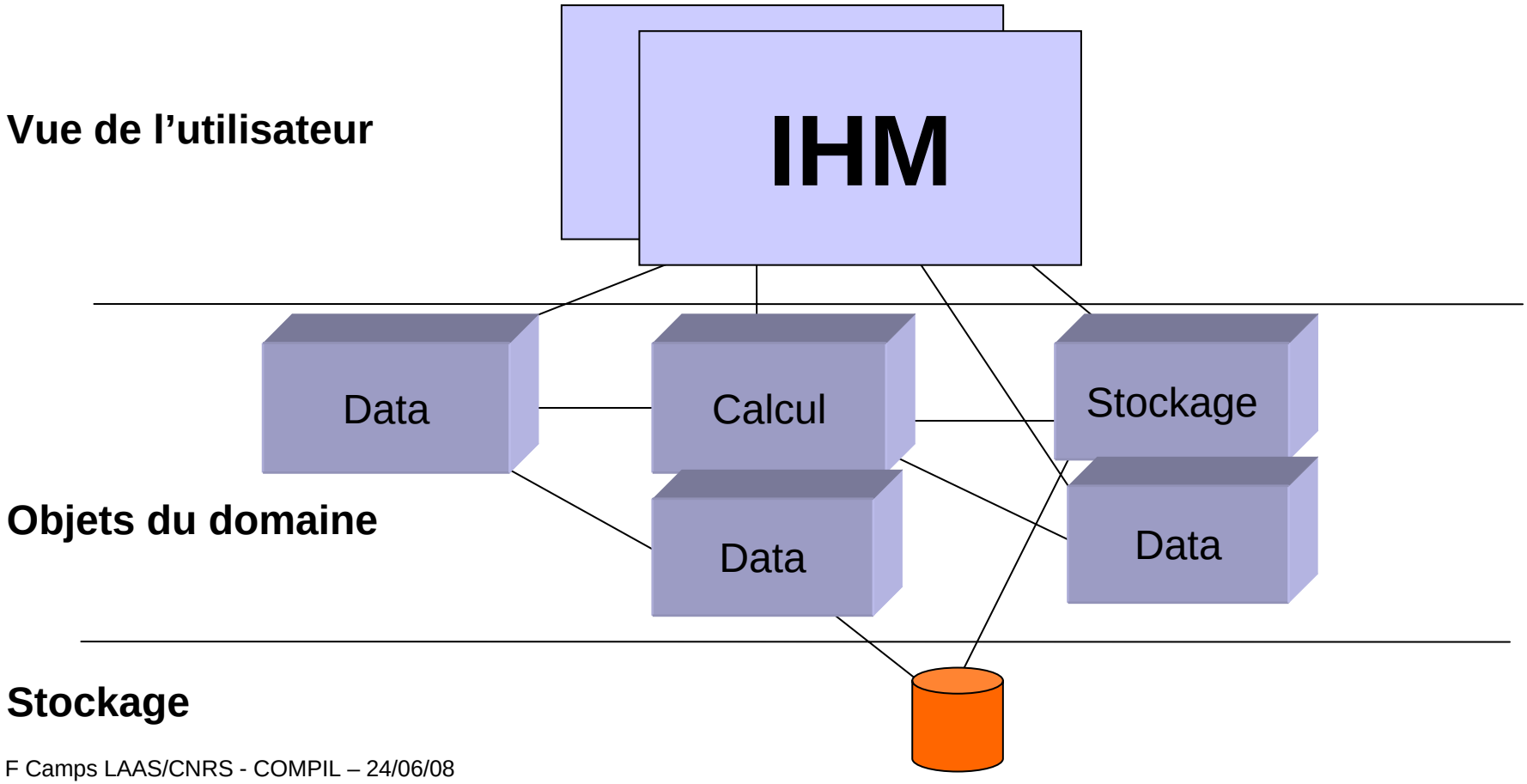
-Affichage de données
-Manipulation de données
-Mise à jour événementielle
-Lien avec d'autres logiciels
-Lancement de module de calcul
…..

Quels sont les liens fonctionnels ?

Liens fonctionnels

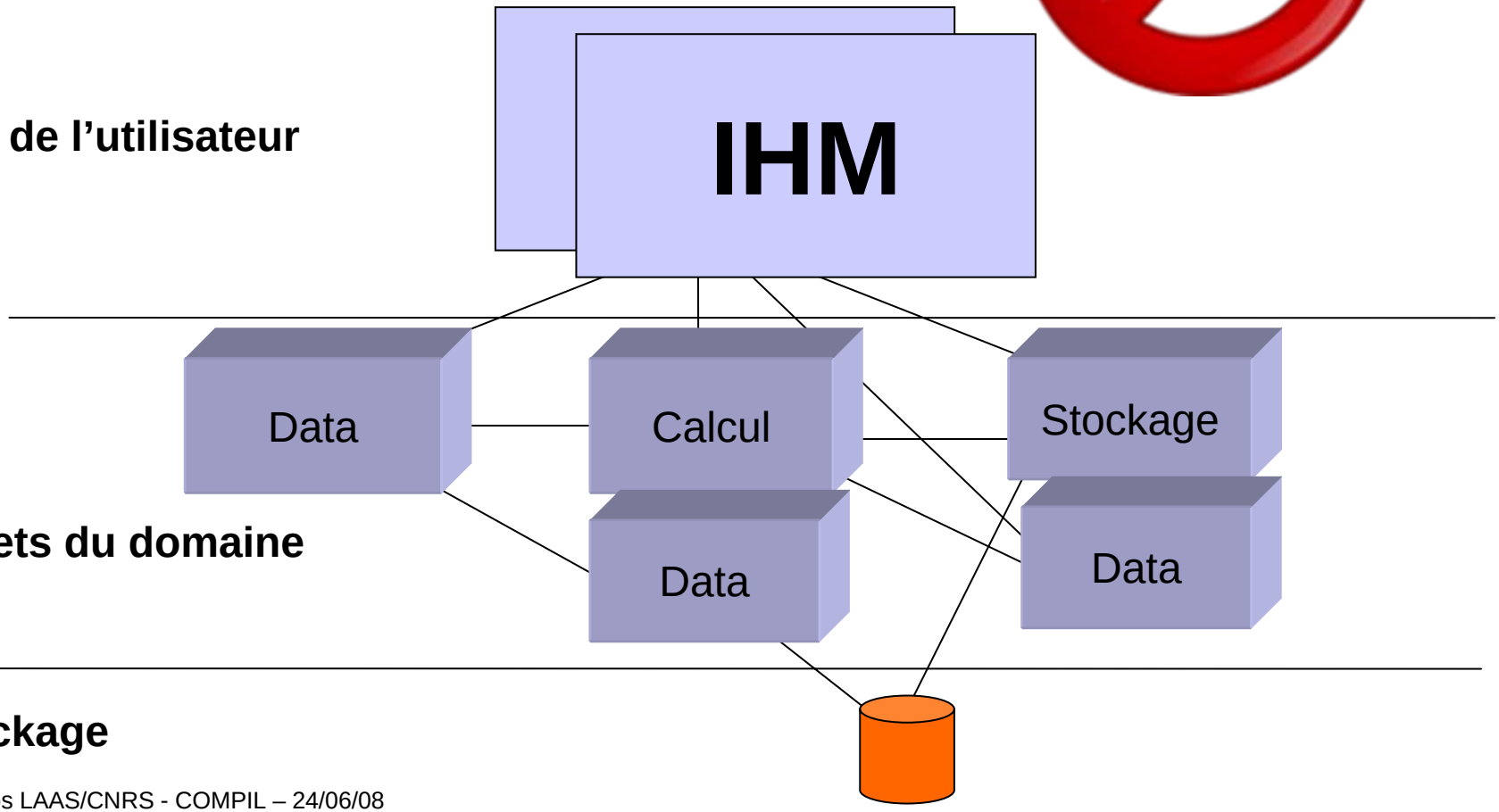


Couplage logiciel fort



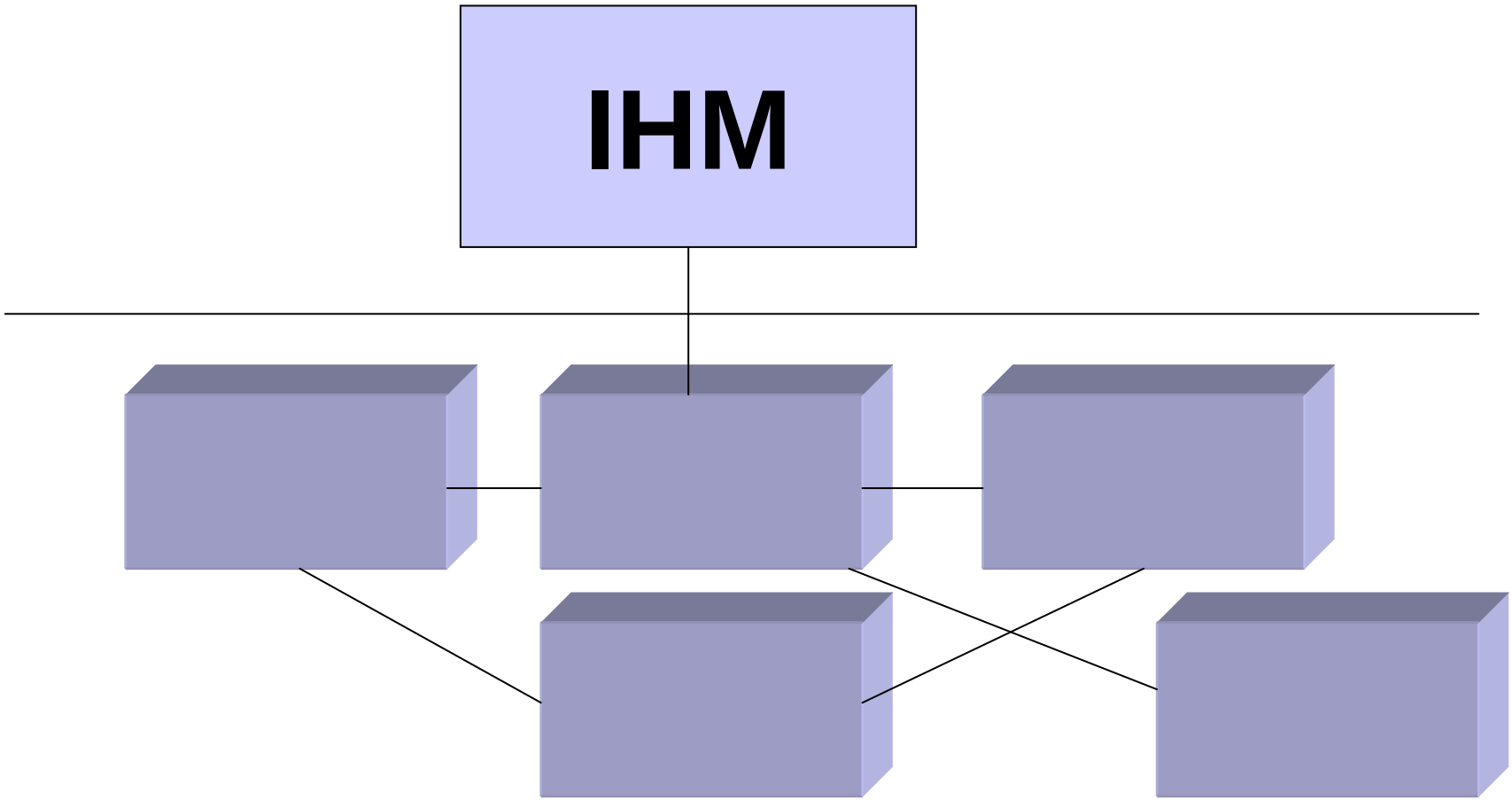
Couplage logiciel fort

Solution à éviter : un fort couplage entre couches logicielles n'est pas souhaitable



Organisation de l'IHM

Solution préférable :

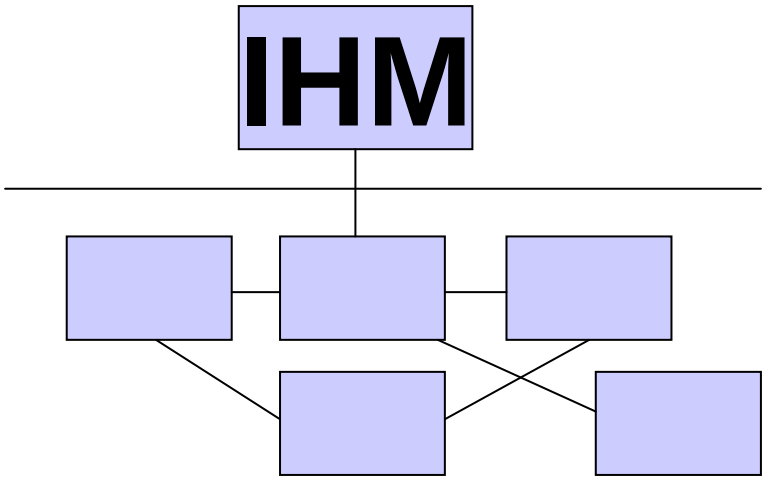


Selon quel schéma ?

- **Faible couplage**. Eviter les associations inutiles entre Classes. Et surtout les associations immutables (héritage par exemple, préférer la délégation)

- **Forte cohésion** : Ttes les responsabilités d'un objet doivent faire partie du même domaine (par ex. 1 classe métier ne doit se soucier que de la valeur de ses attributs, pas de leur persistance. ex. une classe Commande ne peut pas avoir de méthode getNomClient (). C'est de la responsablité de la Classe Client.

Quelle solution ?



Utilisation d'un Design Pattern

Une méthode de conception qui organise l'interface Homme-machine d'une application logicielle. Il divise l'IHM en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des évènements, synchronisation), chacun ayant un rôle précis dans l'interface. Cette méthode a été mise au point en **1979** par **Trygve Reenskaug**, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC

DESIGN PATTERN ?

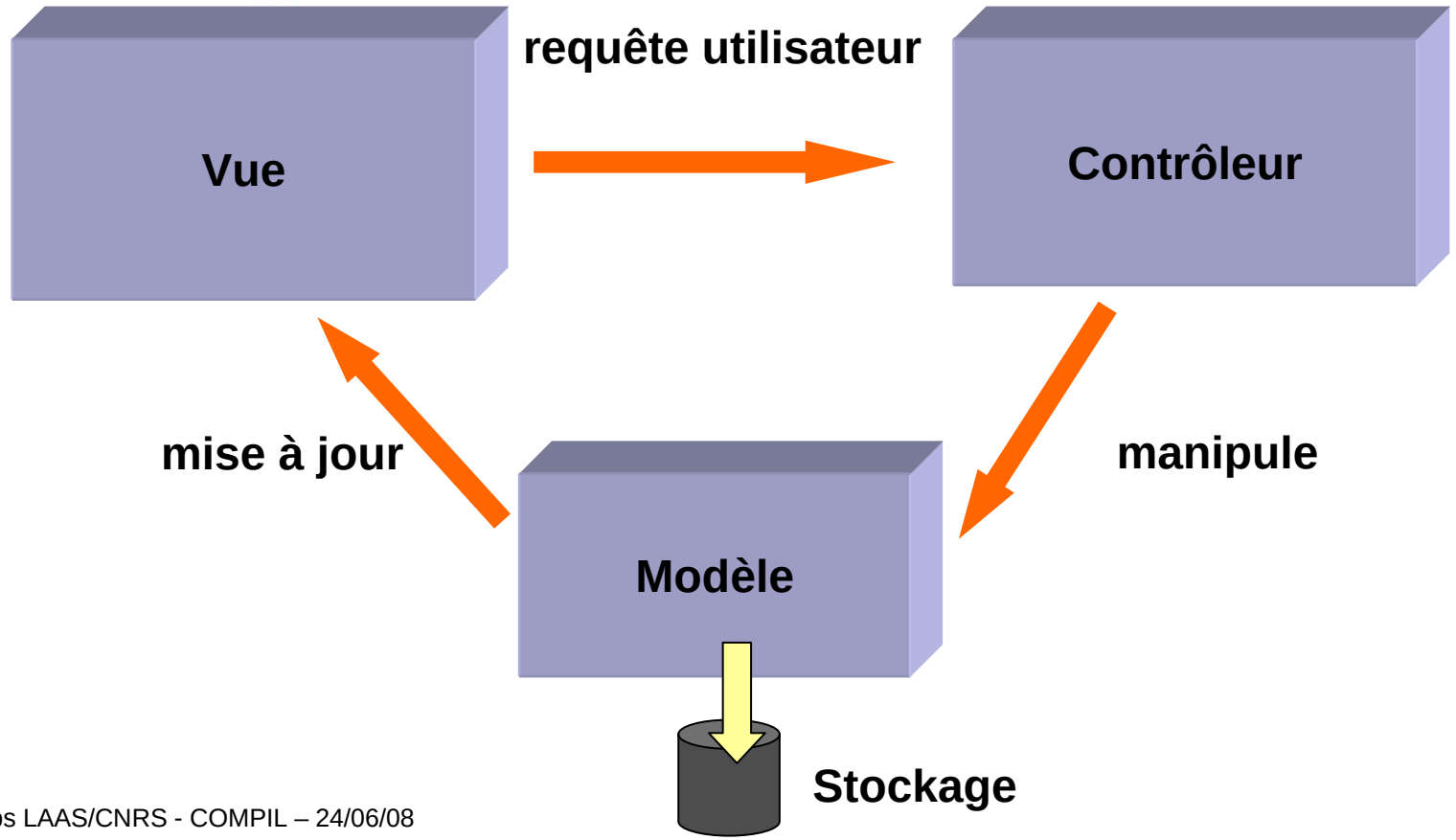
- **Les Design Patterns** (en français Patrons de conception, Modèles de conception ou encore Motifs de conception) sont un recueil de bonnes pratiques de conception pour un certain nombre de problèmes récurrents en programmation orientée objet.

- Le concept de Design Pattern est issu des travaux de 4 personnes (Erich Gamma, Richard Helm, Ralph Johnson, et John Vlissides connus sous le patronyme de « Gang of Four ») dans leur ouvrage « Design Patterns: Elements of Reusable Object-Oriented Software » édité en 1995 et proposant 23 motifs de conception.

DESIGN PATTERN ?

- GRASP : **Patterns généraux d'affectation des responsabilités**
  - **l'expert** (en Information) qui permet d'affecter au mieux une responsabilité à une classe logicielle.
  - **le créateur** qui consiste en la détermination de la responsabilité de la création d'une instance d'une classe par une autre
  - **le faible couplage**, qui a pour objectif de faciliter la maintenance en minimisant les dépendances entre éléments
  - **la forte cohésion** qui mesure le degré de spécialisation des responsabilités d'un composant / classe
  - **le contrôleur** qui consiste en l'affectation de la responsabilité de la réception et/ou du traitement d'un message système a une classe

Modèle MVC : Model – View - Controller



Modèle MVC : Model – View - Controller

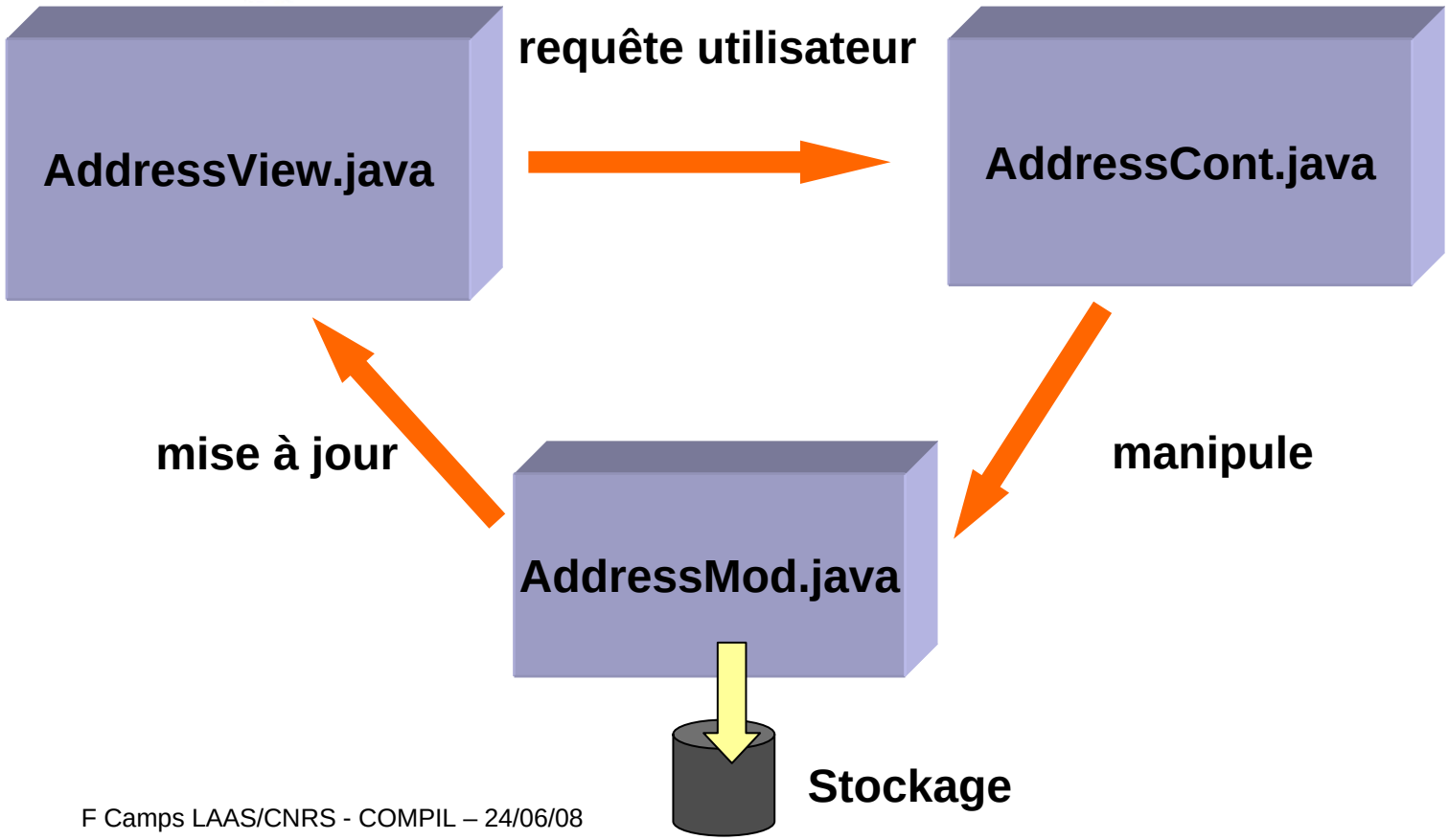
- **La vue** : c'est la partie visuelle, basée sur les données du modèle et qui va agir différemment en fonction des évènements envoyés par celui-ci. Il peut y avoir plusieurs vues ayant pour un seul modèle (c'est bien le but !) et chacune d'entre elles adopteront des comportements spécifiques lorsqu'elles recevront les évènements générés par les modèles. La vue ne prends pas de décision.

- **Le contrôleur** : les vues interagissant avec le modèle, elles ont parfois (souvent) besoin de modifier les données. Le contrôleur sert à faire la liaison entre le modèle et la vue. C'est notamment lui qui, dans l'exemple ci-dessous, s'occupe du traitement des données avant de les transmettre aux modèle. Le traitement des données étant souvent spécifique à chaque vue, il y en a généralement un par vue.

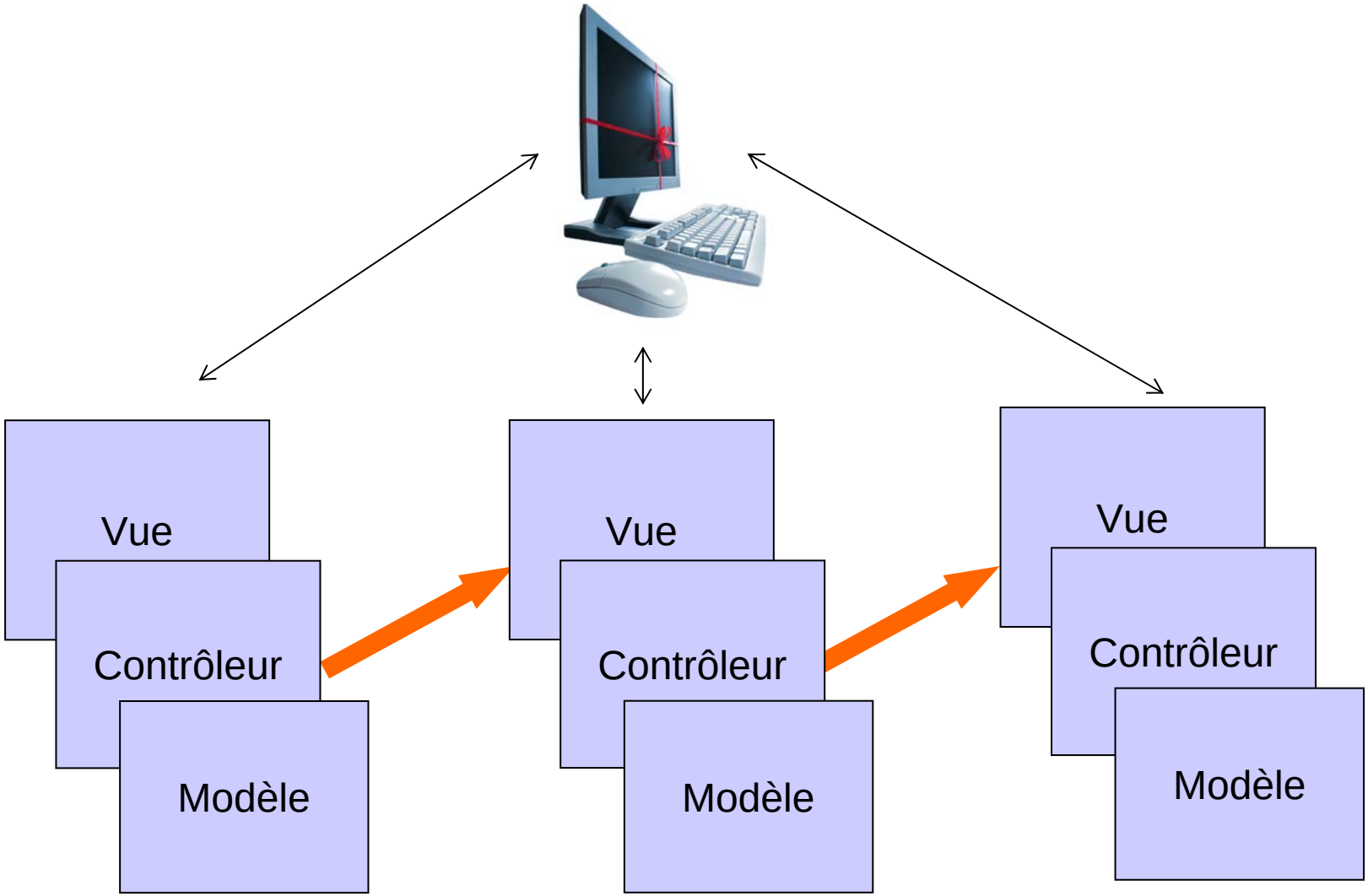
- **Le modèle** : Le modèle représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'une base de données, c'est le modèle qui la contient. .

Modèle MVC : Model – View - Controller

Dans les framework la séparation se fait au niveau du code. Considérez une application de gestion d'adresse :

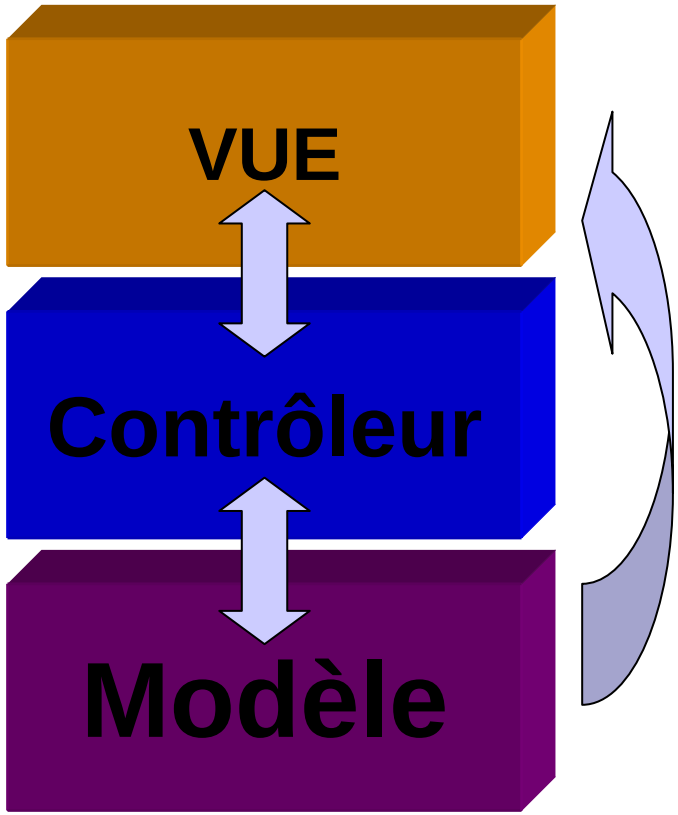


Modèle MVC : Model – View - Controller



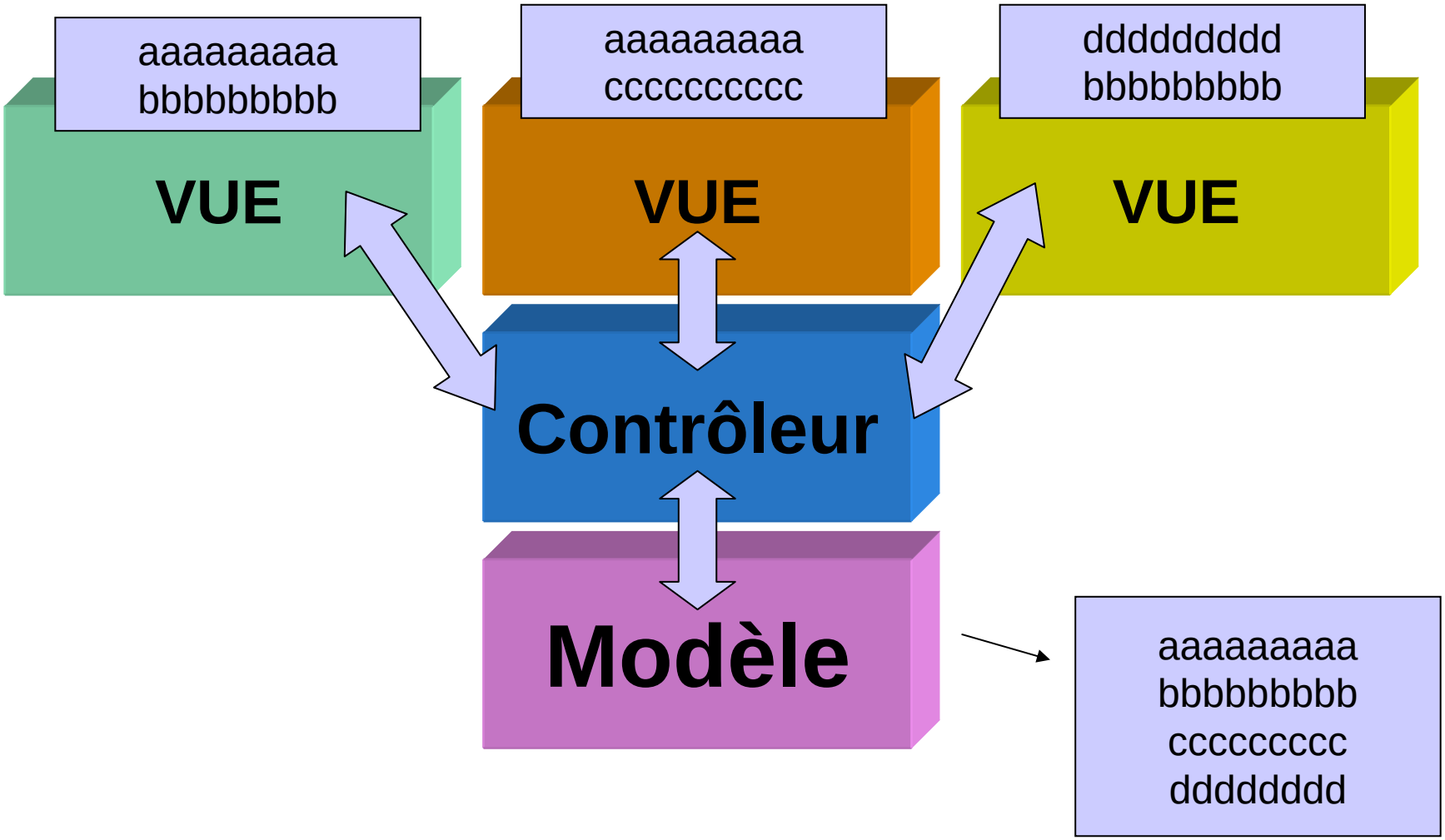
Modèle MVC : Model – View - Controller

- Avantages : briques de base modulables



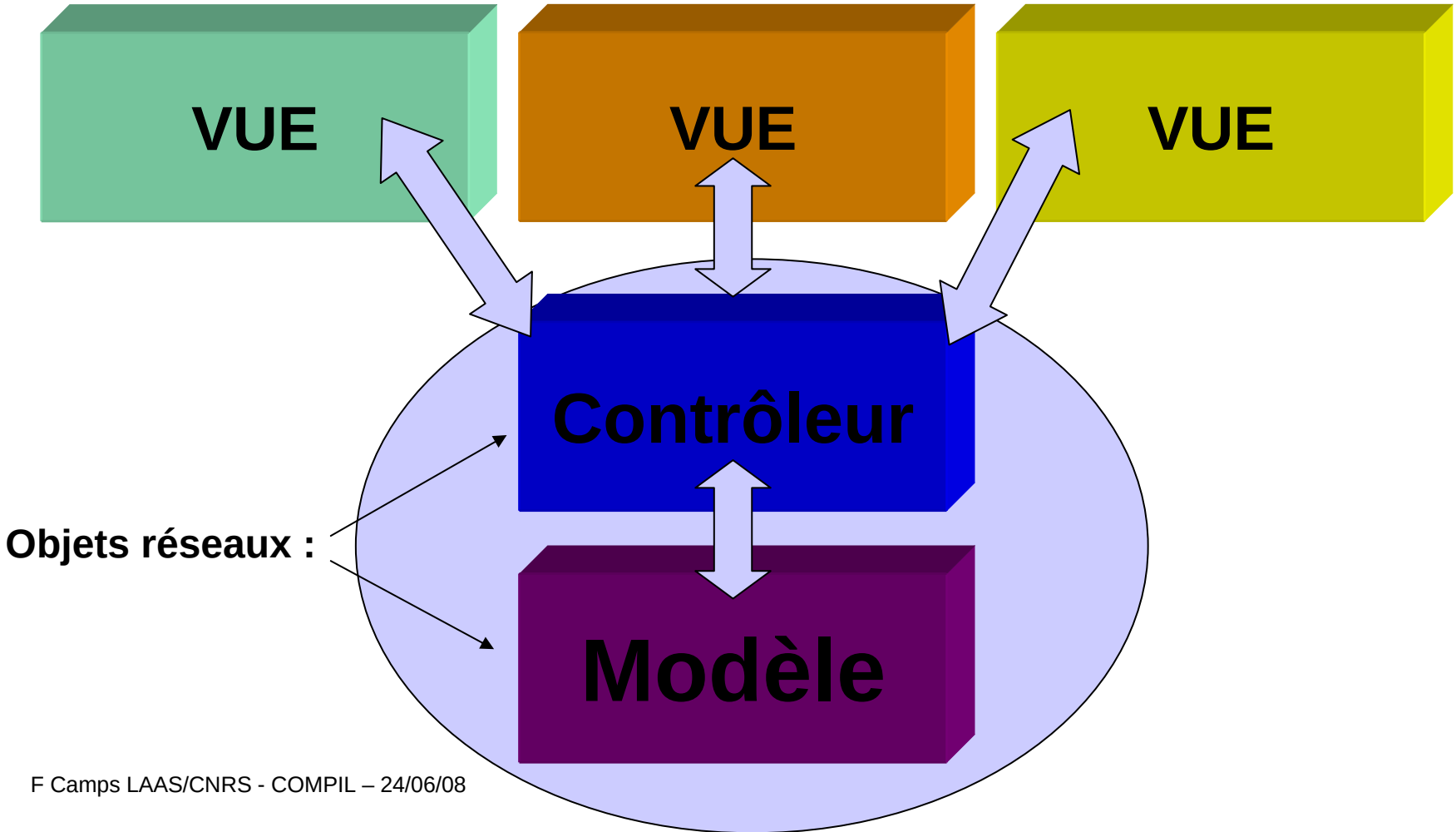
Modèle MVC : Model – View - Controller

- Avantages : plusieurs vues même data



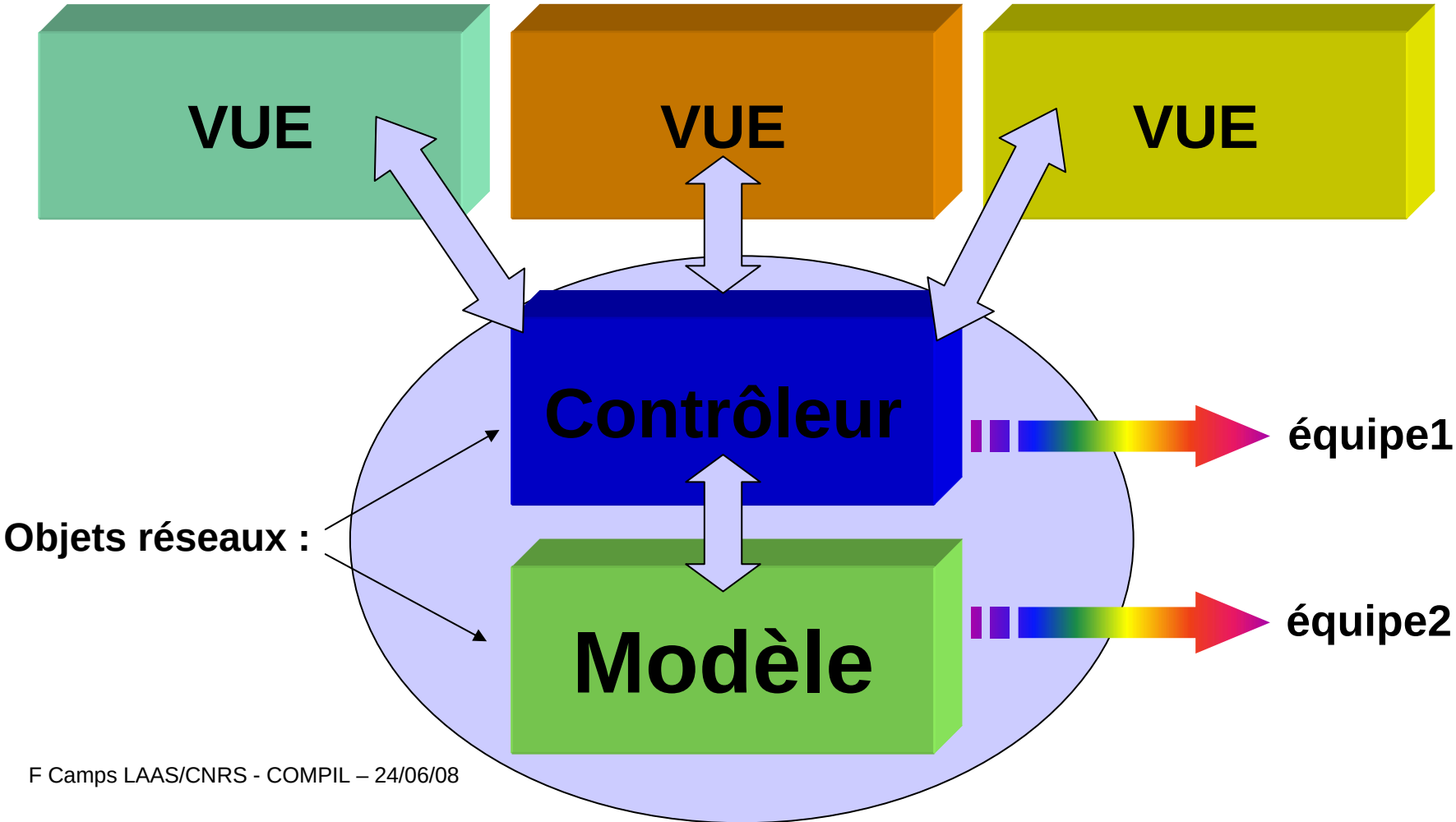
Modèle MVC : Model – View - Controller

- Avantages : répartition réseau



Modèle MVC : Model – View - Controller

- Avantages : dév. en équipe



Modèle MVC : Model – View - Controller

- SUIVRE LES FRAMEWORK DES LANGAGES (IE Jtable Java)

```
class productTableModel extends AbstractTableModel
  {
        private static final long serialVersionUID = 1L;

        private String[] columnNames = {"Parameter", "Include"};

        Class types[] = new Class[]  {  String.class, Boolean.class };


        public int getColumnCount()
        {
                return columnNames.length;
        }

        public String getColumnName(int column)
        {
                return columnNames[column];
        }


        public int getSelectedRow()
        {
                return jTableProduct.getSelectedRow();
        }
```

Modèle MVC : Model – View - Controller

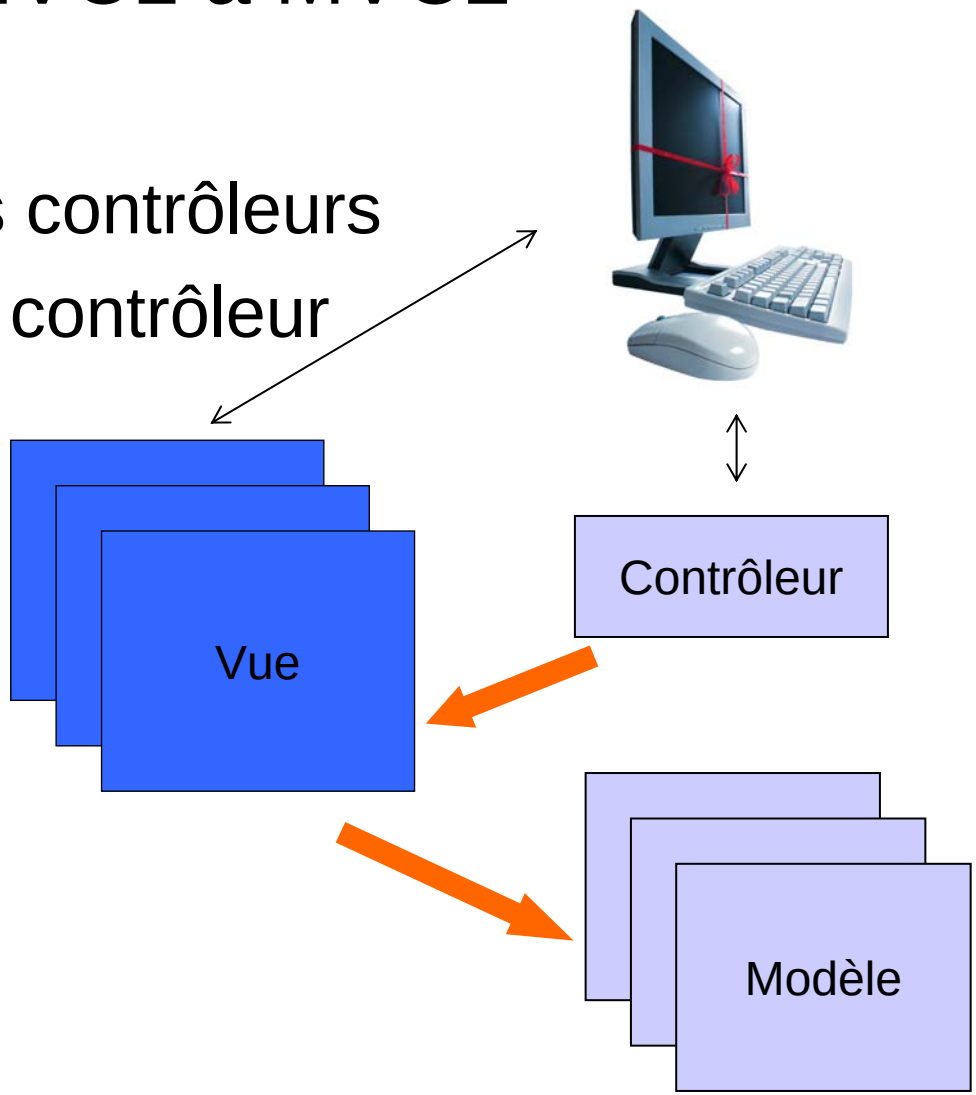
- **Avantages** :
  - Briques de bases interchangeables
  - Technologies hétérogènes / masque les détails d'implémentation -> réduit les dépendances de codage
  - Traitement distribué
  - Réutilisation du code (Vue, Contrôleur, Modèle)
  - Suivre les framework des langages
  - Framework pour appli clients lourds et légers
  - Equipes de développement spécialisées
- **Inconvénients** :
  - Parfois lourds si un programme prévois des vues qui nécessitent des contrôleurs et modèles différents
  - Un minimum de réflexion avant de coder (avantage …)
  - Les Framework nécessitent un temps d'apprentissage

Modèle MVC : Model – View - Controller

- Les erreurs lors du codage
  - Placer des fonctions de contrôle dans la vue au lieu du contrôleur (la vue est alors trop spécialisée)
  - Appeler directement des fonctions du modèle dans la vue (le modèle joue le contrôleur)
  - On a commencé une IHM sans structure MVC !

Modèle MVC : Model – View - Controller

- Evolution de MVC1 à MVC2
  - MVC plusieurs contrôleurs
  - MVC2 un seul contrôleur



Modèle MVC : Model – View - Controller

- **Dans les langages** :
  - Java : swing (Design pattern facade), struts,
  - PHP : cakePHP, Copix, PostNuke
  - Python, Perl, Ruby
  - Xul : utilisé pour construire les logiciels de la fondation Mozilla
  - .Net
  - Visual C++
- Dans les framework
  - Strust, Ruby on rails

Questions

- Est-ce qu'il faut toujours suivre un MVC ?
- Est-ce qu'il faut suivre le MVC d'une lib ?
- Langage sans framework ?
- Autres questions ?

Modèle MVC : Model – View - Controller

- Bibliographie
  - http://fr.wikipedia.org
  - http://baptiste-wicht.developpez.com/tutoriel/conception/mvc/ (exemple complet en Java)
  - http://wiki.mediabox.fr/tutoriaux/flash/designs_patterns/mvc
  - http://www.tonymarston.co.uk/php-mysql/model-view-controller.html (PHP)
  - http://ootips.org/mvc-pattern.html
  - http://dico.developpez.com/html/2118-Conception-GRASP-General-Responsability-Assignement-Software-Patterns.php
  - http://www.commentcamarche.net/genie-logiciel/design-patterns.php3
  - Struts – Campus Press