# Applications orientées données (NSY135)

## 2 – Applications Web Dynamiques

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

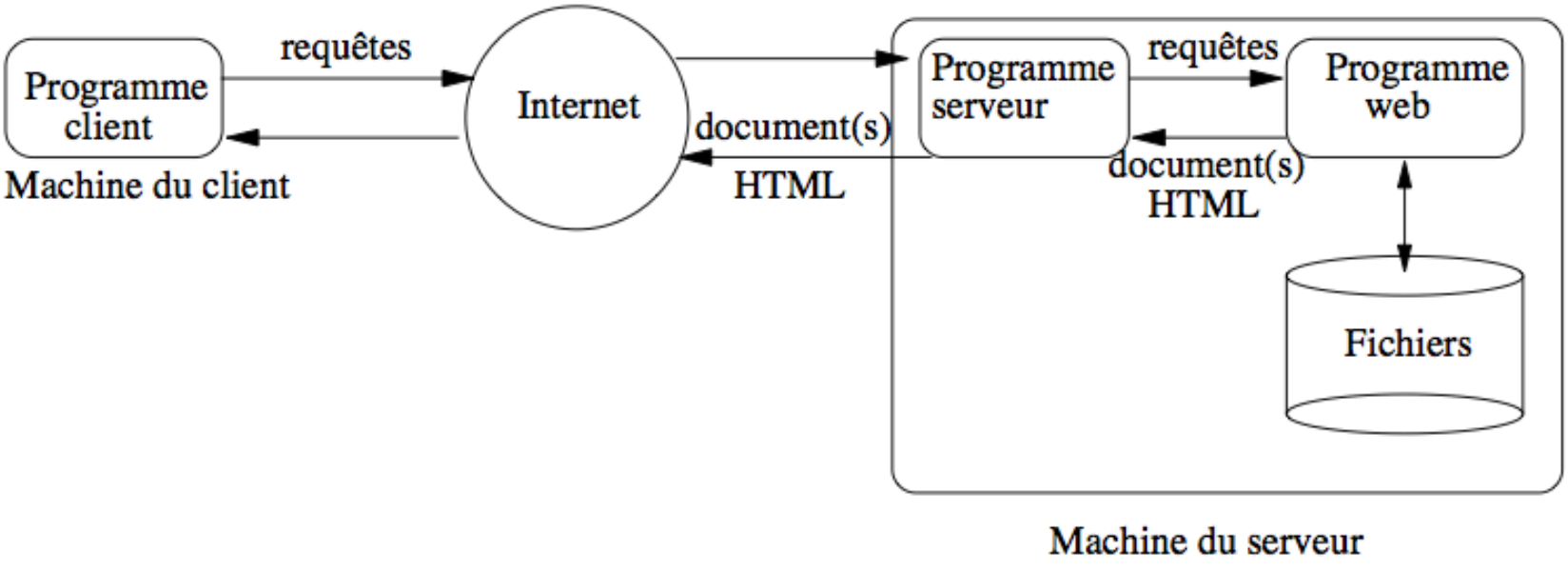
# Plan du cours

S2 Applications Web et frameworks

# Application Web

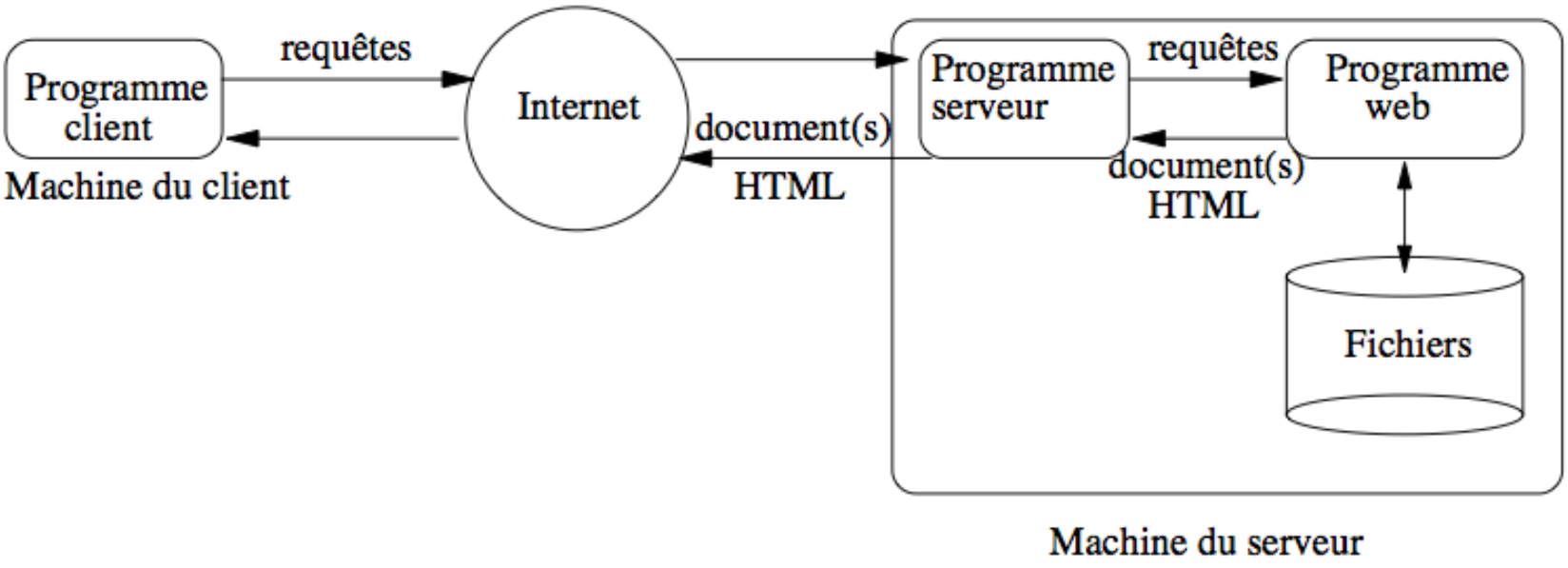
- écrire *à la main* toutes les pages Web d'un site est fastidieux, voire impossible (personnalisation en temps réel pour l'utilisateur)
- Il est donc indispensable de disposer d'application web qui peut interagir avec les besoins de l'utilisateur et lui proposer la ou les pages adaptées, dynamiquement
- Les demandes sont formulées via le navigateur, par le biais de requêtes, prenant la forme d'URL (comportant divers paramètres)

# Architecture



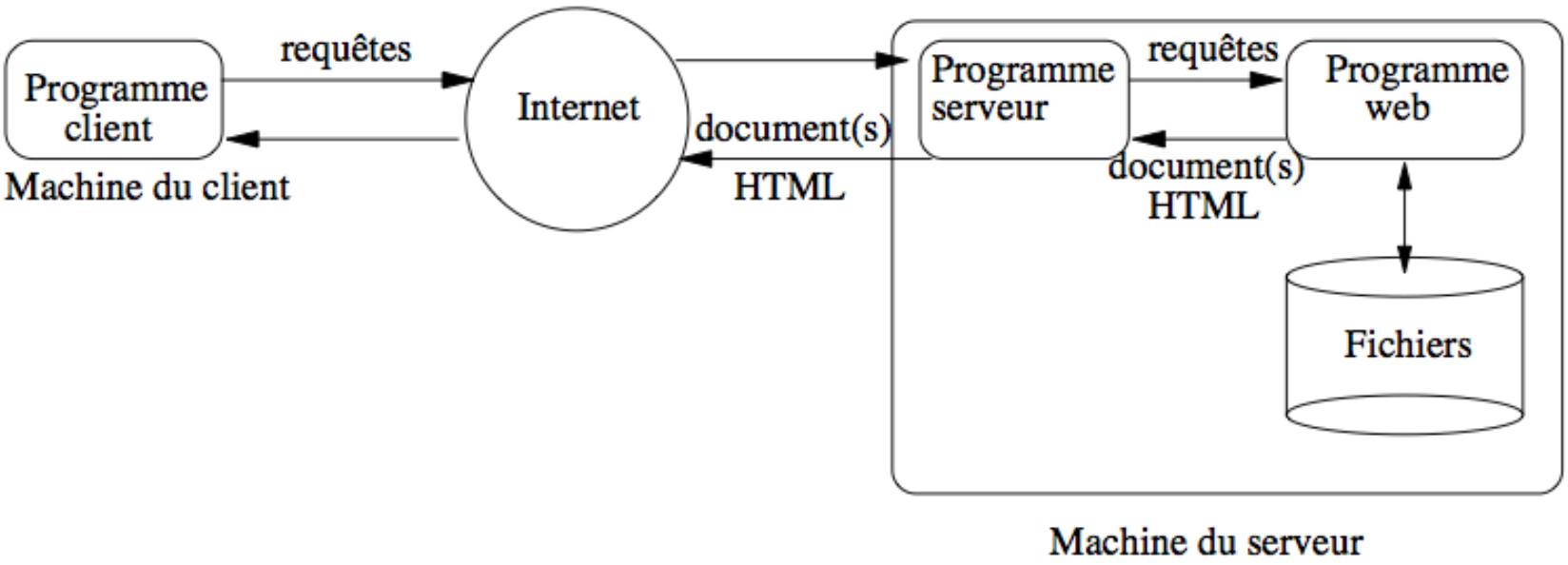

1. Constitution de la requête par le client : le navigateur construit une URL contenant le nom du programme à exécuter, accompagné, le plus souvent, de paramètres saisis par l'internaute dans un formulaire;

# Architecture




2 Réception de la requête par le serveur : le programme serveur récupère les informations transmises par le navigateur, et déclenche l'exécution du programme, en lui fournissant les paramètres reçus;

# Architecture




3. Transmission de la réponse : le programme renvoie le résultat de son exécution au serveur sous la forme d'un document HTML, le serveur se contentant alors de faire suivre au client.

# Langages

Une application web, peut être écrite :

- Dans un langage spécialisé (comme PHP) qui s'intègre étroitement au programme serveur et facilite le mode de programmation particulier aux applications Web
- Dans un langage généraliste comme Java ou Python, mais accompagné dans ce cas d'un ensemble de composants facilitant considérablement la réalisation des applications Web (comme JEE, Java Enterprise Edition)

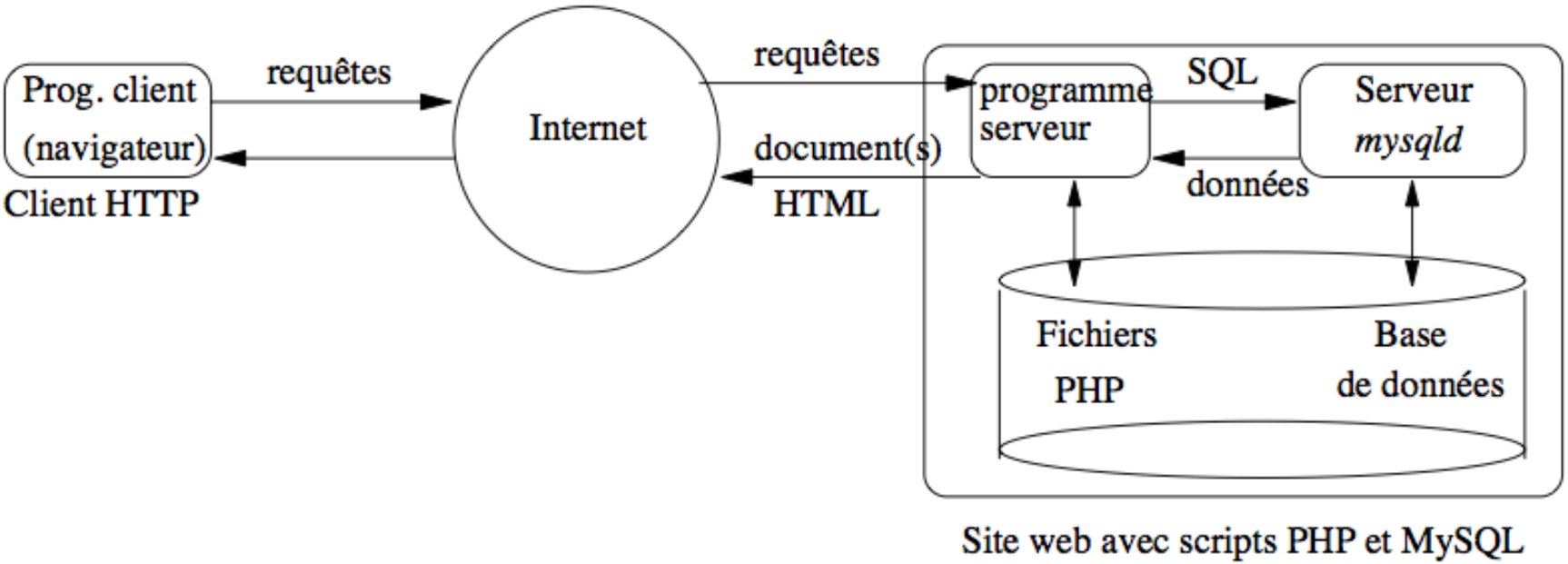
Fonctionnalités :

- Méthode simple pour récupérer les paramètres de la requête, et autres informations (entêtes) transmis par HTTP
- Rechercher et transmettre des fichiers ou des images, effectuer des contrôles, des calculs, créer des rapports, etc.
- accéder à une base de données pour insérer ou rechercher des informations

# Base de données

- La plupart des applications Web ont besoin de s'appuyer sur une base de données pour stocker des données persistantes: comptes clients, catalogues, etc.
- L'application Web, qu'elle soit en Java, PHP ou n'importe quel autre langage, fait donc intervenir un autre acteur, le serveur de données (le plus souvent un système relationnel comme MySQL), et communique avec lui par des requêtes SQL.
- Dans le cas de PHP par exemple, il est possible à partir d'un script de se connecter à un serveur "mysqld" pour récupérer des données dans la base que l'on va ensuite afficher dans des documents HTML. D'une certaine manière, PHP permet de faire d'Apache un client MySQL

# Architecture MySQL PHP



# Caractéristiques des applications Web

Pour simplifier, disons que les applications Web sont

- interactives et distribuées
- écrites à l'aide de divers langages de programmation/présentation
- ouvertes et sujettes aux problèmes de sécurité

⇒ On a donc besoin de bonnes pratiques et de conventions pour leur architecture et leur développement

# Qu'est ce qu'un framework?

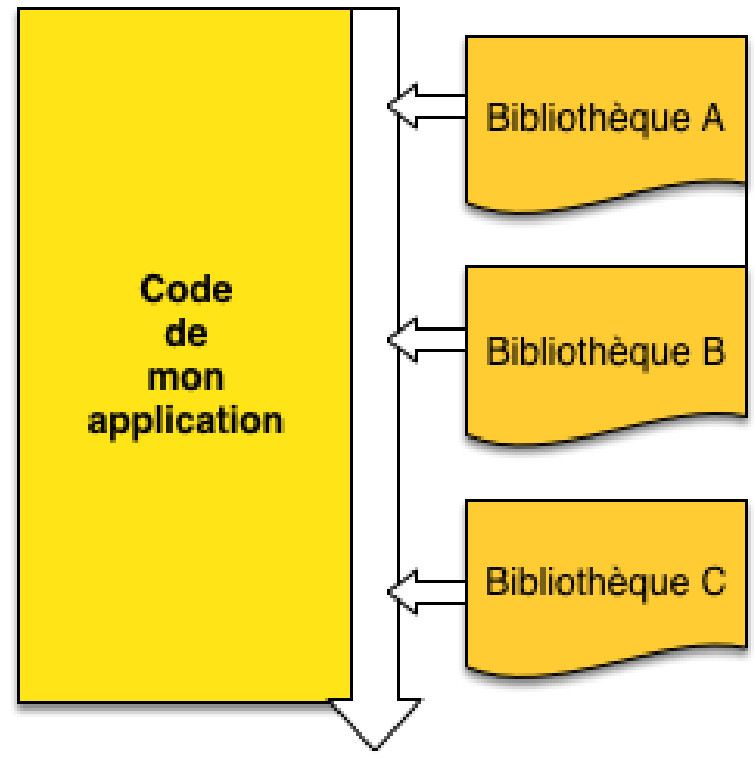
*Framework* = "cadre de travail". Concrètement, un *framework*, c'est :

- une conception: une architecture, un modèle, une approche générique pour répondre à un besoin ou problème (générique) donné
- des bonnes pratiques: un ensemble de règles, de conventions, de recommandations, de patrons (*design patterns*) pour mettre en œuvre la conception
- et enfin, un environnement logiciel qui aide le développeur – et parfois le force – à adopter l'architecture/modèle et respecter les règles.

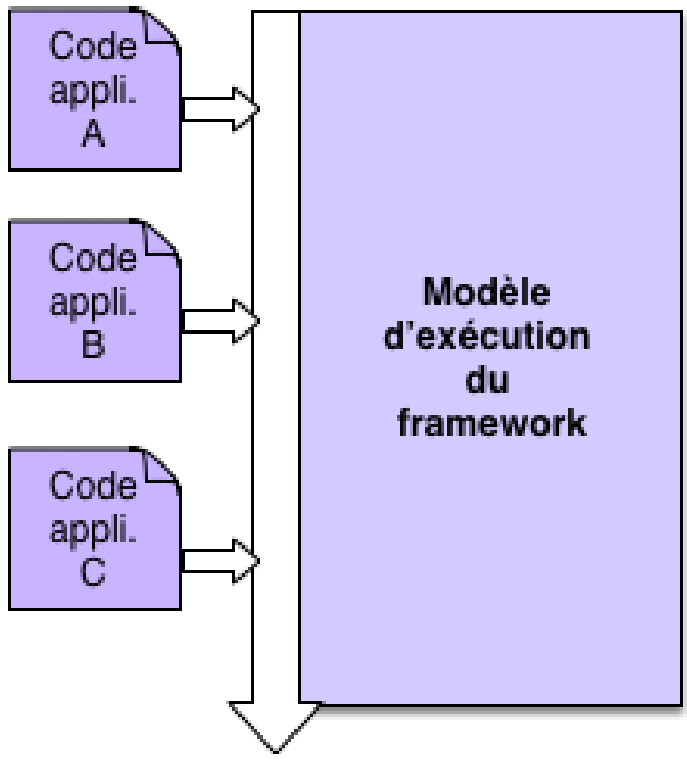
On parle à tort et à travers de *frameworks*, notamment pour désigner des choses qui n'en sont pas (librairies, composants logiciels). Nous allons essayer d'être précis !

# Un *pattern* fondamental: l'inversion de contrôle

Un *framework* est un composant actif, une bibliothèque un composant passif.




*Appel à des bibliothèques*




*Contrôle par un framework*

Le *framework* contrôle l'exécution de l'application, qui injecte des composants.

# Exemple basique: un formulaire de saisie

Solution A: je produis toutes les balises et le contenu:

```
print "<form action='...'>"
print "<input type='text' name='email'>Votre email</input>"
print "<input type='text' name='nom'>Votre nom</input>"

print "</form>"
```

Solution B: j'utilise un générateur de formulaire, et j'injecte mes questions.

```
formEngine = new FormEngine (/* paramètres */)
formEngine->add (new InputText ("email", /* paramètres */))
formEngine->add (new InputText ("nom", /* paramètres */))

formEngine->render()
```

Le moteur `formEngine` doit être *générique* et prendre en charge toutes les options possibles.

# Avantages d'un framework

Un framework fait bénéficier d'une conception rigoureuse et de bonnes pratiques éprouvées.

- Conception validée, vérifiée, amendée et perfectionnée pendant des années: garantie de qualité.
- Des pratiques mises au point par des milliers de développeurs dans le monde: beaucoup d'erreurs et de pièges déjà résolus/évités.

Un framework normalise de fait le travail de développement.

- Essentiel pour le travail en équipe
- Facilite la compréhension / validation du travail d'un développeur.
- Facilite la reprise du code si besoin est.

# Inconvénients d'un framework

Ne pas les sous-estimer!

- Une certaine lourdeur de mise en route (installation, configuration, etc.)
- Des concepts abstraits, une courbe d'apprentissage assez sévère.
- Éventuellement des problèmes de performance ou de souplesse.
- Rend votre application (en partie) dépendante du framework.

À considérer : au-delà d'une application triviale (afficher des pages web statiques) et surtout pour du travail en équipe, un *framework* représente un gain considérable.

# Exemple : créer une application Web simple

Le besoin: je veux créer une application Web; j'affiche des pages HTML, que je crée à la volée en fonction de requêtes HTTP et que j'alimente avec une base de données.

La conception (générique): mes pages HTML sont des vues, les interactions HTTP sont gérées par des contrôleurs, et la logique de mon application (persistante ou pas) est mon modèle: architecture MVC.

Bonnes pratiques et patrons de conception (*design patterns*) formalisant les bonnes pratiques: le nommage des fichiers, des variables, organisation des répertoires, jusqu'à l'indentation, les commentaires, etc.

Frameworks MVC : Symphony/PHP, Zend/PHP, Spring/Java, Grails/Java, Django/Python, Ruby on rails, etc.

# Exemple : application objet et base SQL

Le besoin: je veux rendre persistantes mes données objets / je veux "objectiser" mes données relationnelles.

La conception (générique): mes objets forment un graphe, mes données sont des tables, une fonction de *mapping* transforme l'un vers l'autre.

Bonnes pratiques: représentation des associations, de l'héritage, traduction des navigations dans le graphe et requêtes sur les données, etc.

Frameworks ORM: JPA/Hibernate, JPA/EclipseLink (Java), Doctrine, CakePHP (PHP), ActiveRecords (Ruby)