Nancy-Université

Mise en œuvre des serveurs d'application

UE 203d

Master 1 IST-IE

Printemps 2008

Ces transparents, ainsi que les énoncés des TDs, seront disponibles à l'adresse :

`http://www.loria.fr/~burel/empty_cours.html`

Première partie I

Principes généraux

Plan

Architecture à 1 niveau

Toutes les opérations sont effectuées sur la même machine
On communique sur des terminaux
Exemple : éditeur de texte

Critique

Avantages :

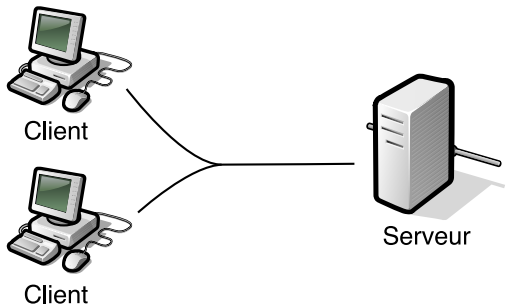
- simple
- performant
- autocontenu

Inconvénients :

- pas de communications (pas de calcul distribué)
- code logiciel souvent mal architecturé

Architecture à 2 niveaux

Aussi appelées architecture client-serveur :
Les calculs sont effectués sur le serveur
Les clients interrogent le serveur
Exemple : Messagerie instantanée



Critique

Avantages :

- séparation de la présentation et du contenu
- calcul distribué

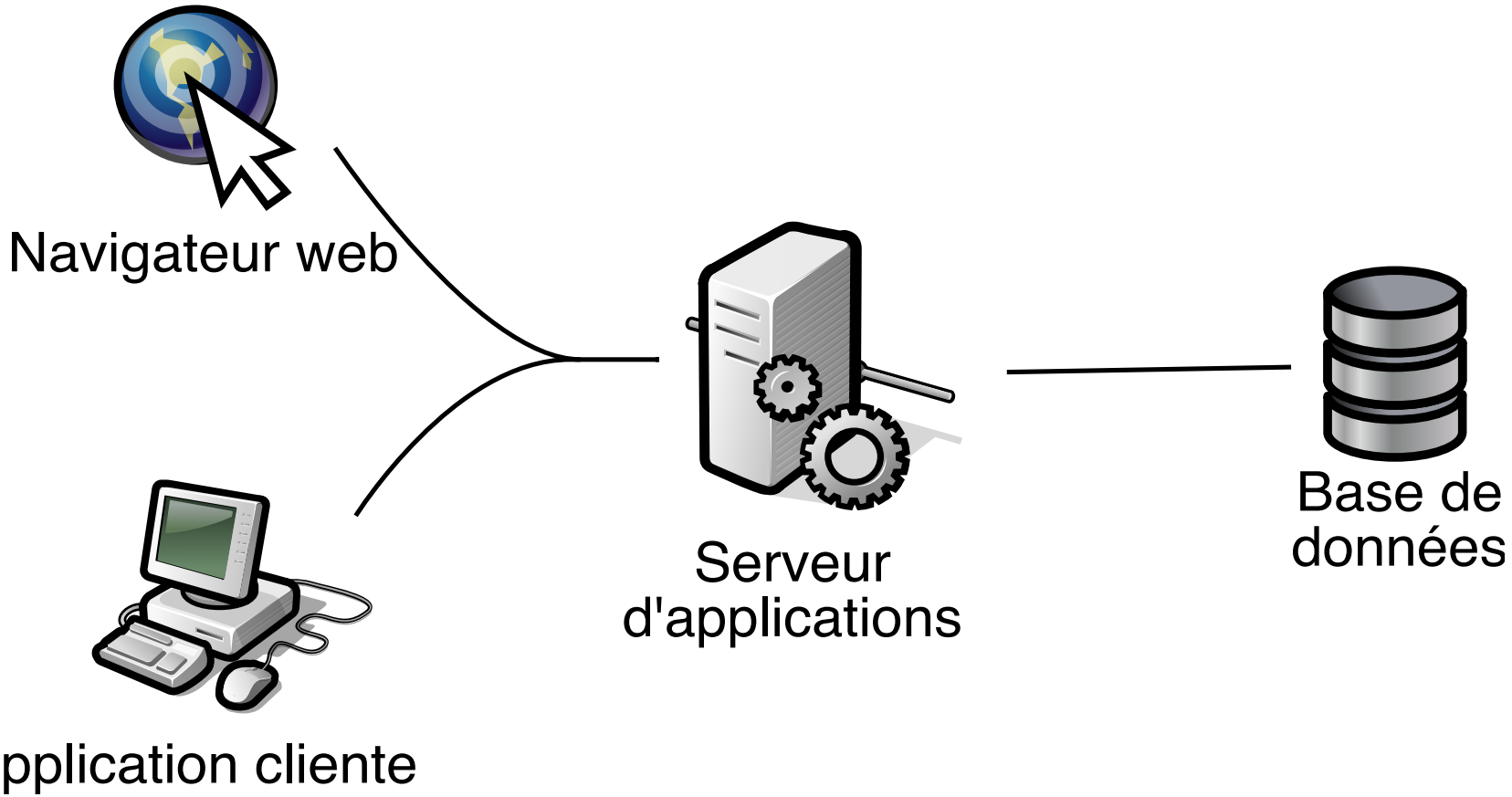
Inconvénients :

- difficile de modifier une partie du serveur sans tout modifier
- les calculs faits par le serveur ne sont pas distribués

Architecture à 3 niveaux

On sépare la gestion des données (stockage, recherche) de leur traitement (calcul)

Exemple : Application bancaire



Critique

Avantages :

- les clients ne dépendent pas des bases de données
- le niveau du milieu se concentre sur la logique métier
- tâches distribuées, code facilement modifiable

Inconvénients :

- demande plus de rigueur
- standards assez complexes
- code parfois répétitif

Architecture à n niveaux

La logique métier est séparées en plusieurs niveaux
En particulier la partie web, la gestion de messages asynchrones (ex. : envoi de mail) peuvent être distingués
Permet de gérer la sécurité : chaque niveau a des droits d'accès différents
Exemple : Site commercial

Critique

Avantages :

- code très distribué : une machine pour une tâche, passage à l'échelle
- réutilisabilité de chacun des niveaux
- code facilement maintenable

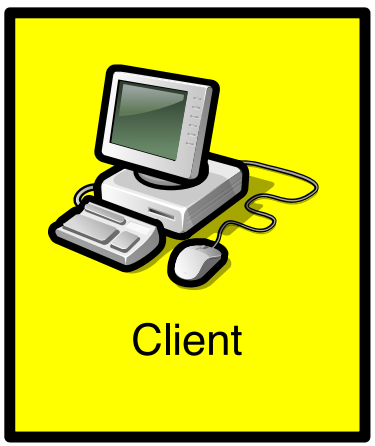
Inconvénients :

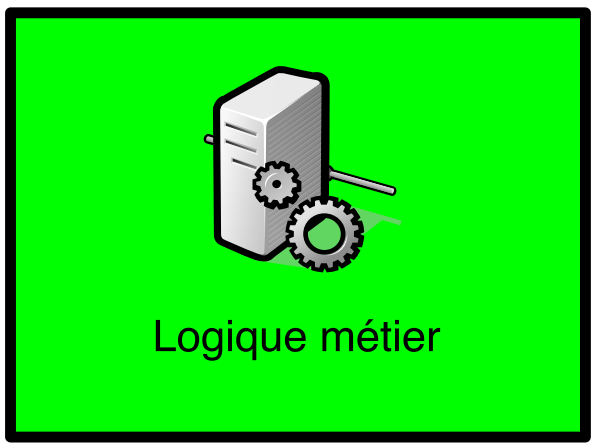
- architecture complexe à mettre au point
  $\longrightarrow$ besoin de standards
- performance (trop de communications/calculs)

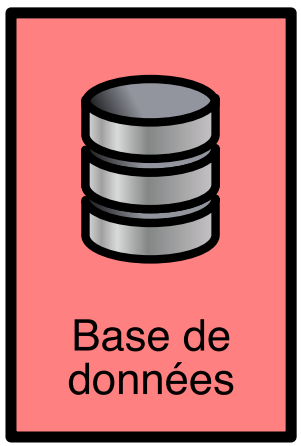
Plan

- Architecture multi-niveaux
  - Architecture à 1 niveau
  - Architecture à 2 niveaux
  - Architecture à 3 niveaux
  - Architecture à n niveaux
- Les serveurs d'application
  - Description
  - Niveau web
  - Niveau métier
- Type de serveurs d'application
  - J2EE
  - .NET
  - Autres

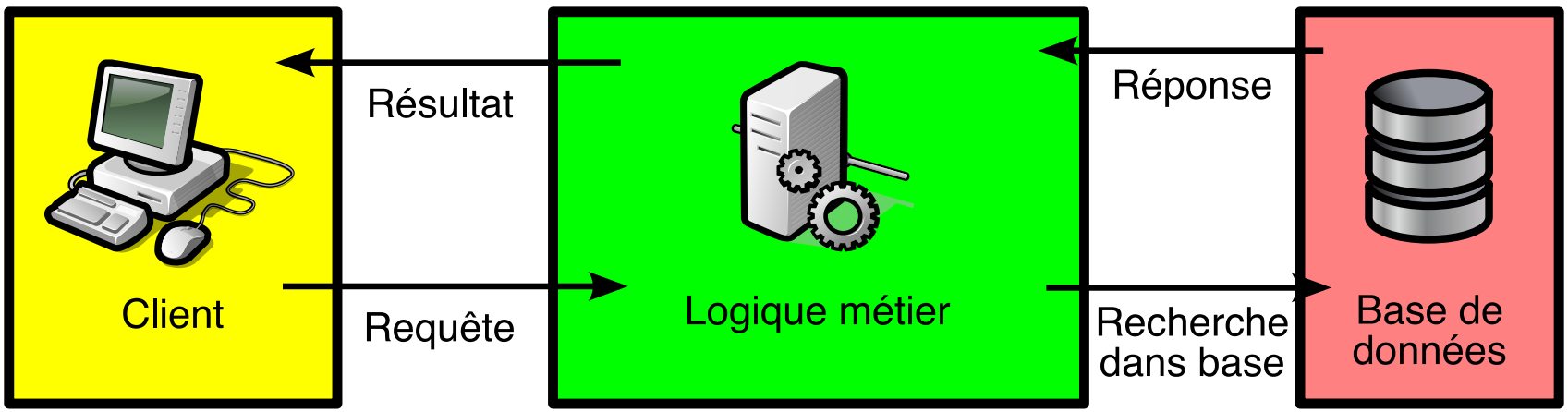
Architecture à 3 niveaux



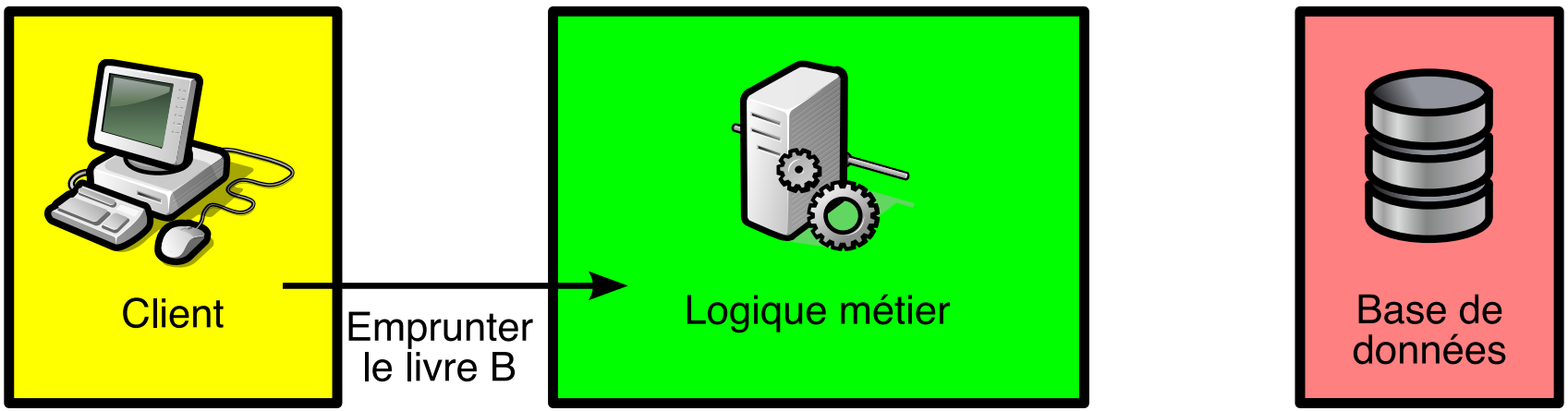




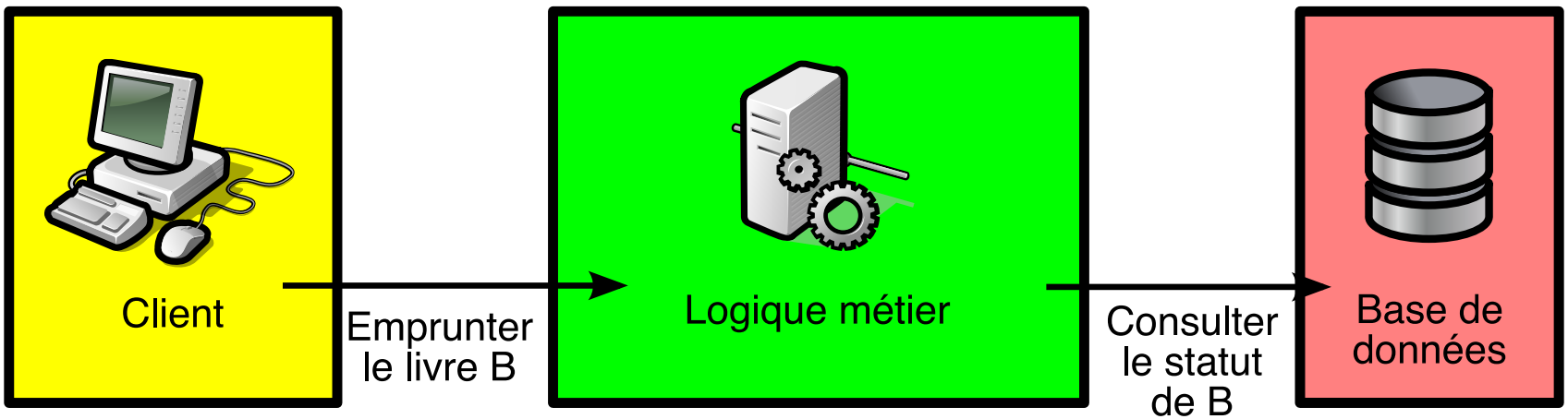
Processus


Client
Résultat
Requête
Logique métier
Réponse
Recherche dans base
Base de données

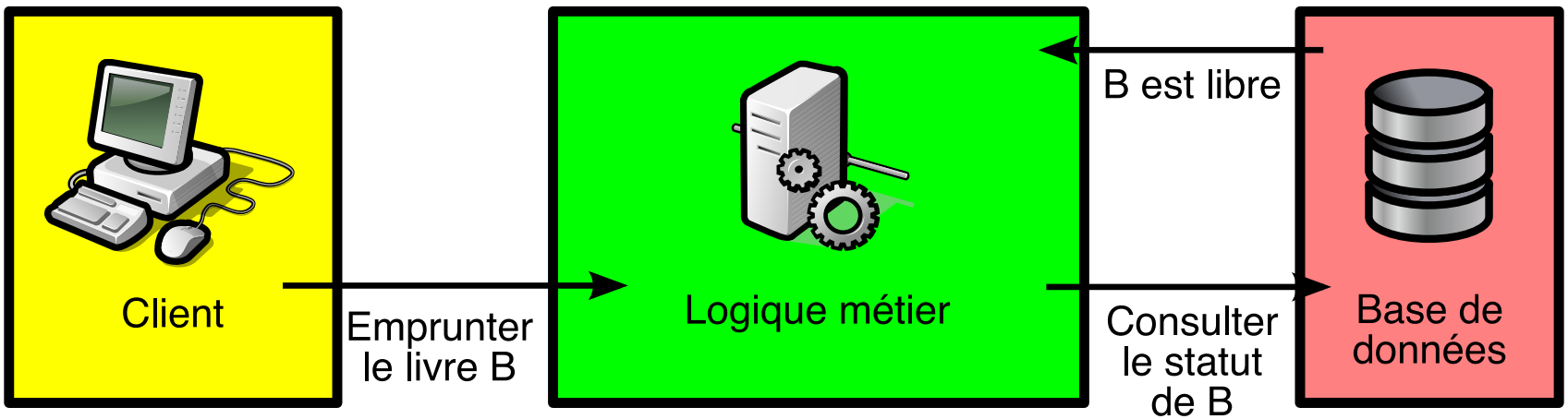
Exemple



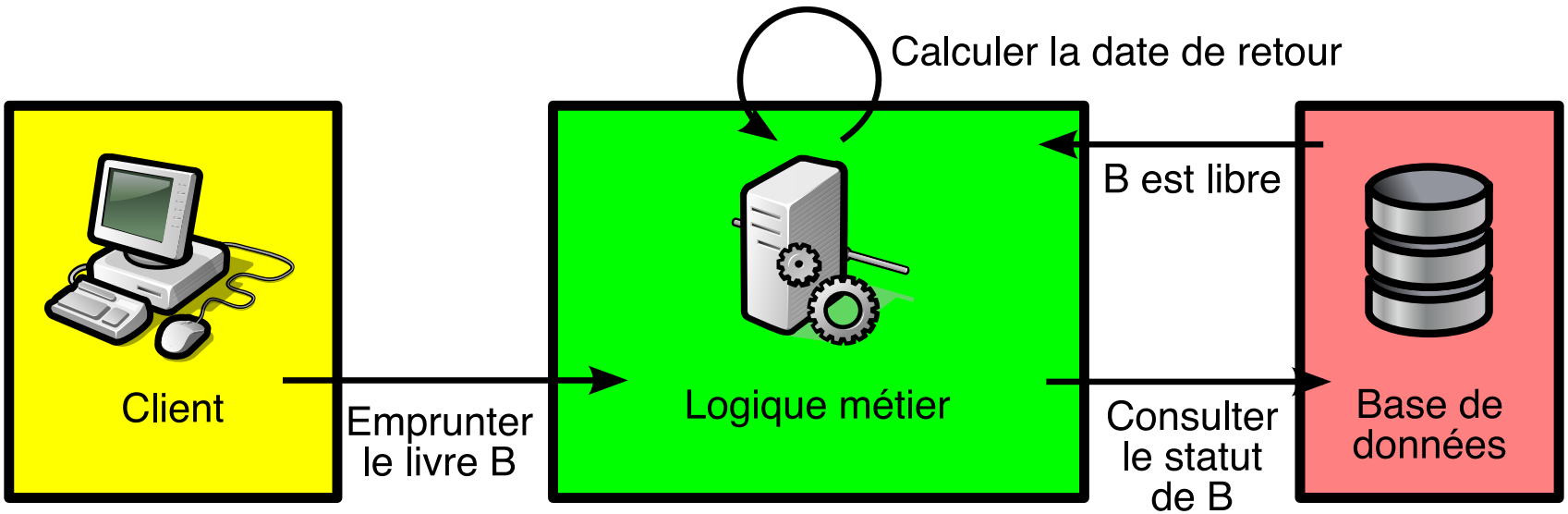
Exemple


Client
Emprunter le livre B
Logique métier
Consulter le statut de B
Base de données

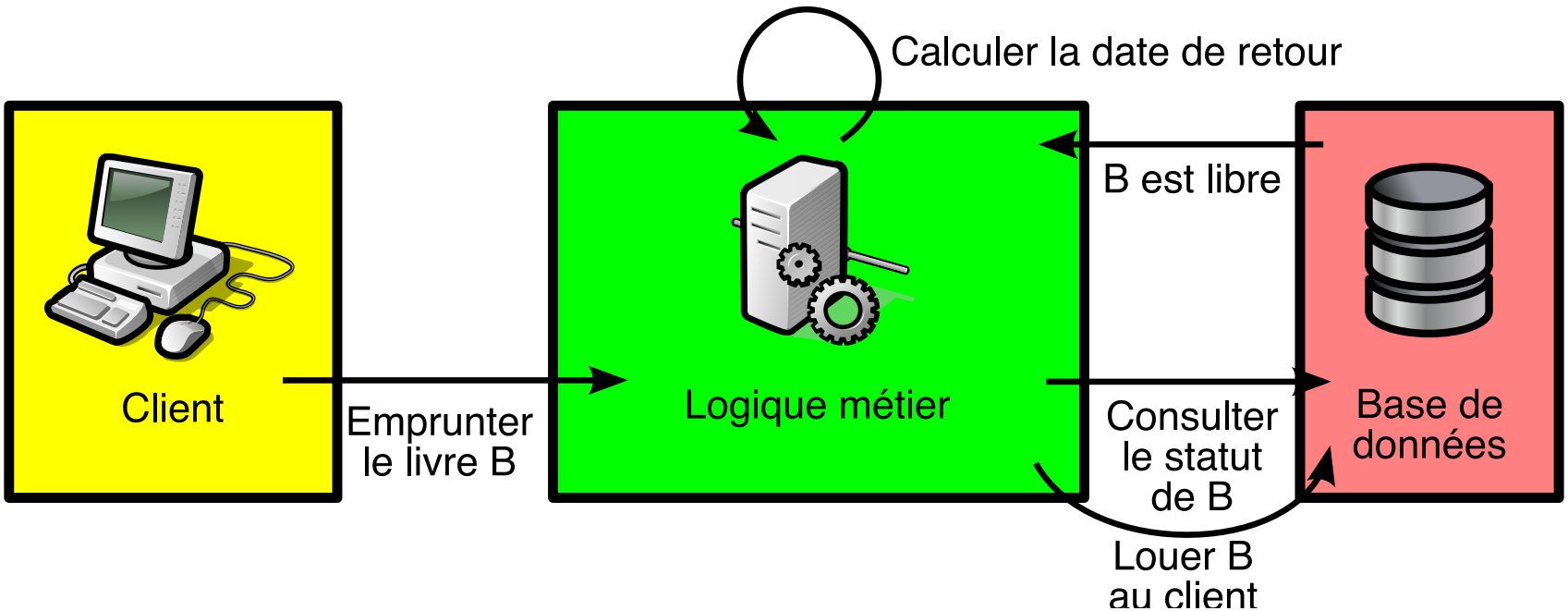
Exemple


Client
Emprunter le livre B
Logique métier
B est libre
Consulter le statut de B
Base de données

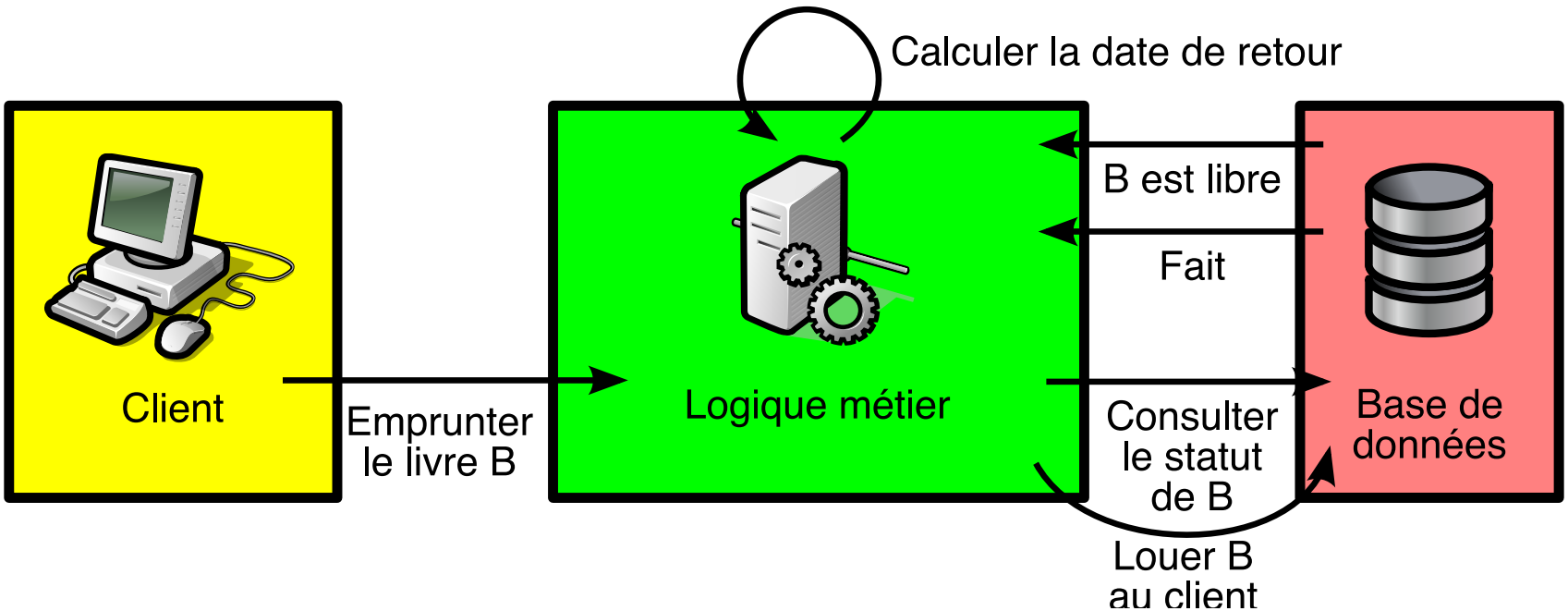
Exemple


Calculer la date de retour
Client
Emprunter le livre B
Logique métier
B est libre
Consulter le statut de B
Base de données

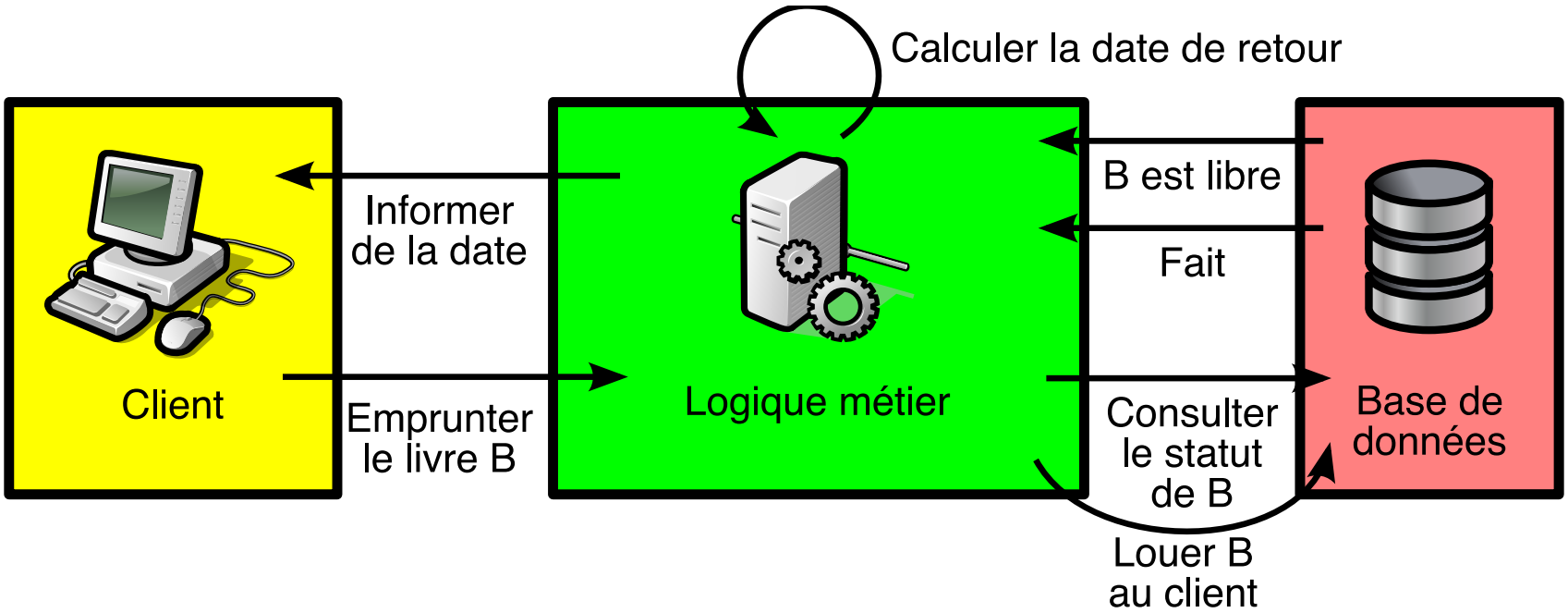
Exemple


Calculer la date de retour
Client
Emprunter
le livre B
Logique métier
B est libre
Consulter
le statut
de B
Louer B
au client
Base de
données

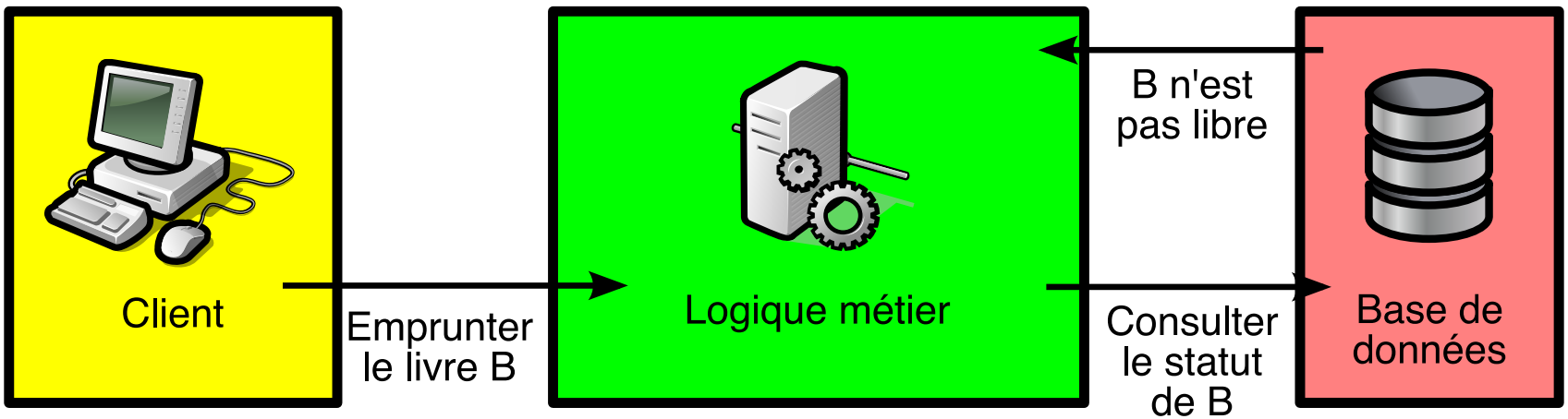
Exemple


Calculer la date de retour
Client
Emprunter
le livre B
Logique métier
B est libre
Fait
Consulter
le statut
de B
Louer B
au client
Base de
données

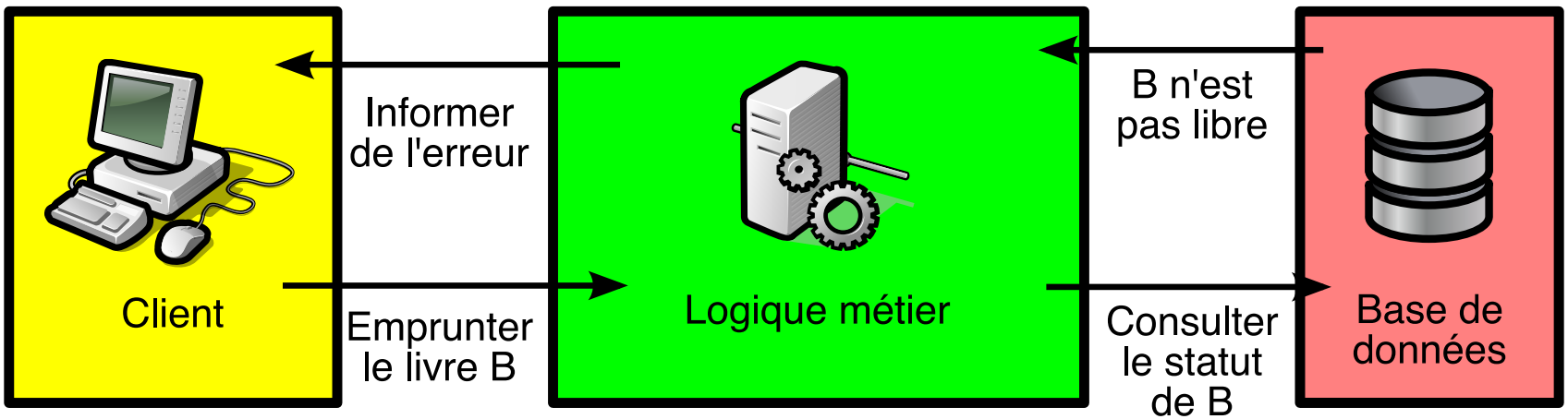
Exemple


Calculer la date de retour
Client
Informer
de la date
Emprunter
le livre B
Logique métier
B est libre
Fait
Consulter
le statut
de B
Louer B
au client
Base de
données

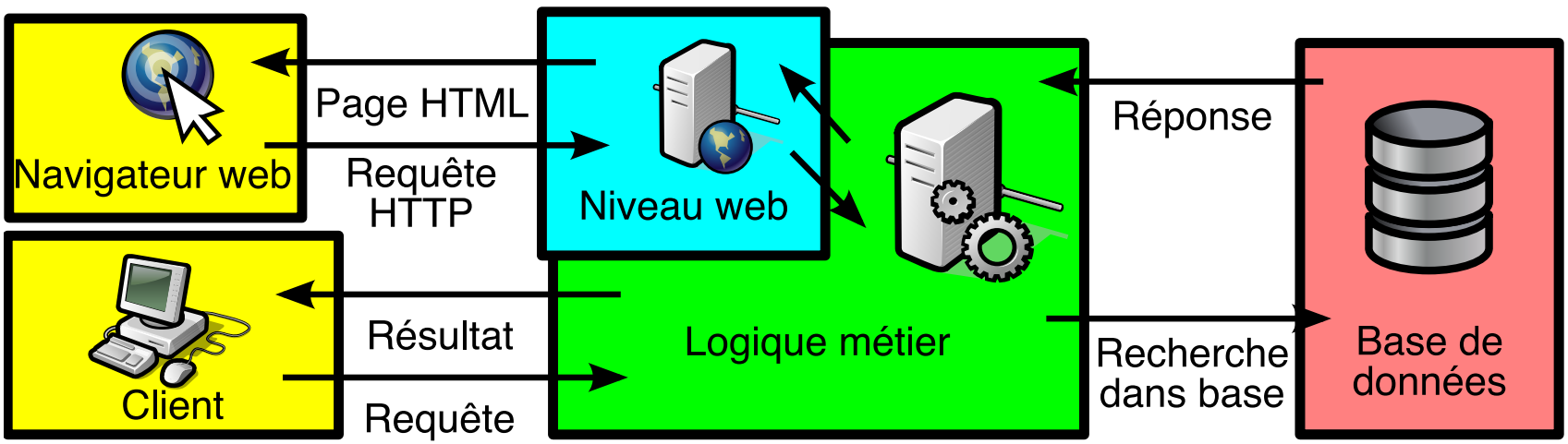
Exemple


Client
Emprunter le livre B
Logique métier
B n'est pas libre
Consulter le statut de B
Base de données

Exemple


Client
Informer
de l'erreur
Emprunter
le livre B
Logique métier
B n'est
pas libre
Consulter
le statut
de B
Base de
données

Niveau web


Navigateur web
Client
Page HTML
Requête HTTP
Résultat
Requête
Niveau web
Logique métier
Réponse
Recherche dans base
Base de données

Couche web

Création de ressources web

- à l'aide de programmes spécifiques (Servlets java)
- à l'aide de langages étendant HTML pour ajouter des parties dynamiques calculées par le serveur, type JSP et ASP.NET

Les ressources sont accessibles pour le client à l'aide d'un simple navigateur $\rightarrow$ portabilité, simplicité

Interfaces

Offre une interface de requêtes pour les clients
Opérations de base : création de compte, ajout d'une transaction, recherche de la liste des comptes, etc.

Améliore la réutilisabilité et la maintenance : possibilité de changer toute la logique métier sans changer les clients, à condition que l'interface reste la même

Base(s) de données

Communique avec la base de données (BD)
Exemple : à partir d'un nom d'utilisateur `?1`,
requête `SELECT * IN LIVRES l, UTILISATEURS u WHERE u.no = l.no_ut AND u.nom =?1`
= cherche les livres empruntés par `?1`

stocke les résultats de la recherche dans une structure manipulable par les programmes (tableau, liste, vecteur, etc.)
Intérêt : logique métier indépendante de la BD

Traitement des données

Dans base de données, données non formatées
$\longrightarrow$ traitement des données
Exemple : calcul de la durée de l'emprunt en fonction du statut de l'emprunteur
Transforme requêtes client complexes en requêtes BD plus simples
Gère requêtes asynchrones (envoi de mail, etc.)

Plan

- Architecture multi-niveaux
  - Architecture à 1 niveau
  - Architecture à 2 niveaux
  - Architecture à 3 niveaux
  - Architecture à n niveaux
- Les serveurs d'application
  - Description
  - Niveau web
  - Niveau métier
- Type de serveurs d'application
  - J2EE
  - .NET
  - Autres

Java Platform, Enterprise Edition (JEE ou J2EE)

Standard développé par Sun
Ensemble de spécifications que doit vérifier un serveur d'application

- architecture standardisée, plus facile à comprendre pour l'extérieur
- portabilité : peut passer d'un serveur d'applications J2EE à un autre sans trop de problème
- plus de détails dans la suite du cours

Critique

Avantages :

- standard
- nombreuses implantation avec différents coûts et performances
- dont implantations libres
- disponible sur différentes plateformes (Windows, Unix libre ou propriétaire, . . .)

Inconvénients :

- le code doit être écrit en Java
- la portabilité entre serveurs d'application J2EE n'est pas totale
- standard assez complexe $\rightarrow$ besoin de former les concepteurs du logiciel
- assemblage de briques parfois hétérogènes

Exemples d'implantations

- Sun Java System Application Server, GlassFish
- JBoss
- IBM WebSphere
- JOnAS (Bull, France Télécom, Inria)
- Apache Geronimo
- et d'autres...

Microsoft .NET

centre de développement complet proposé par Microsoft
disponible uniquement sous environnement Windows
plusieurs langages possibles : C#, Visual Basic, F#, J#, etc.
compilés dans un langage commun : "Common Language Infrastructure" qui est ensuite compilé en langage machine
ASP.NET permet de créer des pages web dynamiques

Critique

Avantages :

- intégration complète avec le système d'exploitation
- nombreux langage disponibles
- unité, cohérence

Inconvénients :

- spécifique à une plateforme
- non libre (sources ouvertes récemment)
- implantations libres (Mono, CrossNet) partielles

Zope

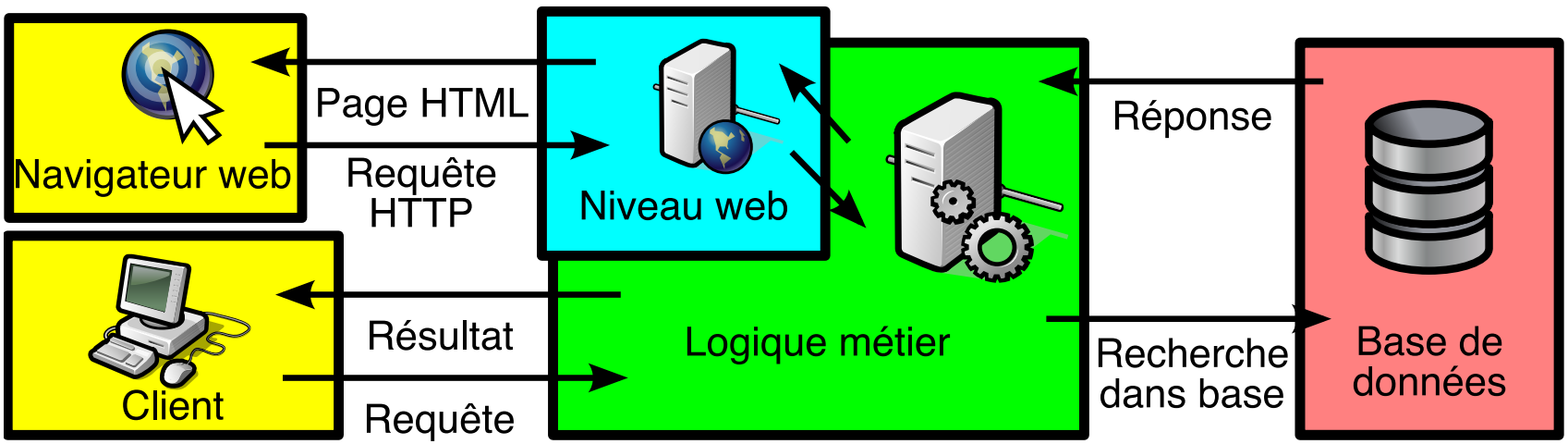
Serveur d'applications web libre
Langage utilisé : python
Rapide, faible technicité requise
Utilisé dans Plone : système de gestion de contenu

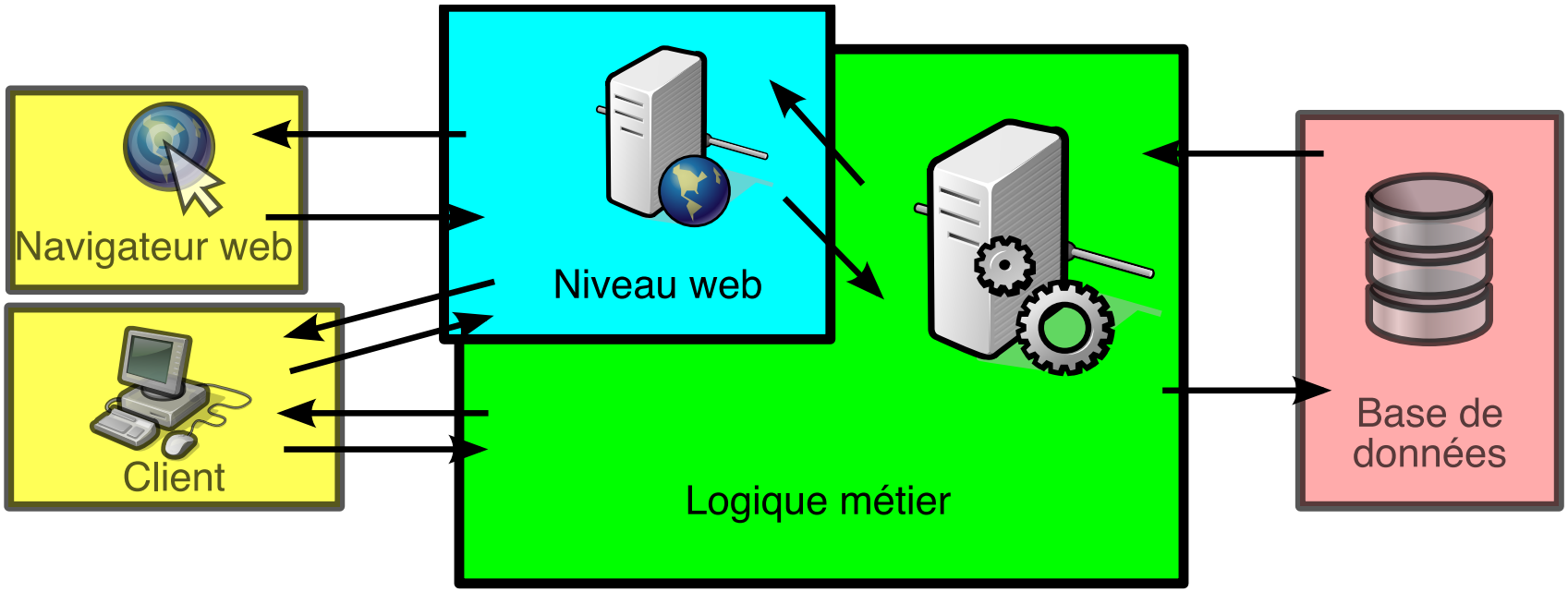
Deuxième partie II

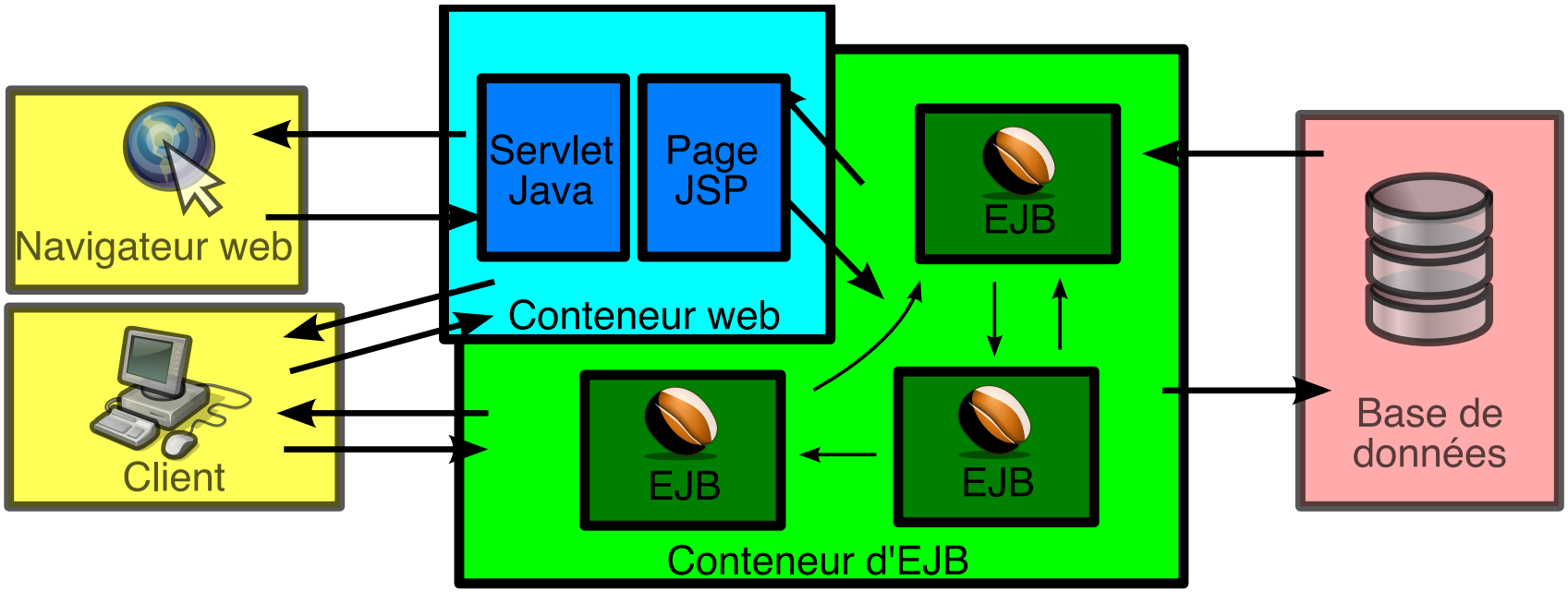
Mise en œuvre d'une application J2EE

Plan

- Généralités
- Niveau web
- Niveau métier
  - Les Enterprise Java Beans
  - Beans session
  - Beans entité
  - Bean contrôlés par messages
  - XDoclet
- Sécurité


Navigateur web
Client
Page HTML
Requête HTTP
Résultat
Requête
Niveau web
Logique métier
Réponse
Recherche dans base
Base de données


Navigateur web
Client
Niveau web
Logique métier
Base de
données


Navigateur web
Client
Servlet Java
Page JSP
Conteneur web
EJB
EJB
EJB
Conteneur d'EJB
Base de données

Plan

- Généralités
- Niveau web
- Niveau métier
  - Les Enterprise Java Beans
  - Beans session
  - Beans entité
  - Bean contrôlés par messages
  - XDoclet
- Sécurité

Servlets

Objets java qui gère les requêtes qu'on lui soumet
En particulier : servlets HTTP
gère les six requêtes du protocole HTTP : GET POST OPTIONS DELETE PUT TRACE
utilise un objet HttpServletResponse pour retourner la réponse (sous forme de page HTML en général)

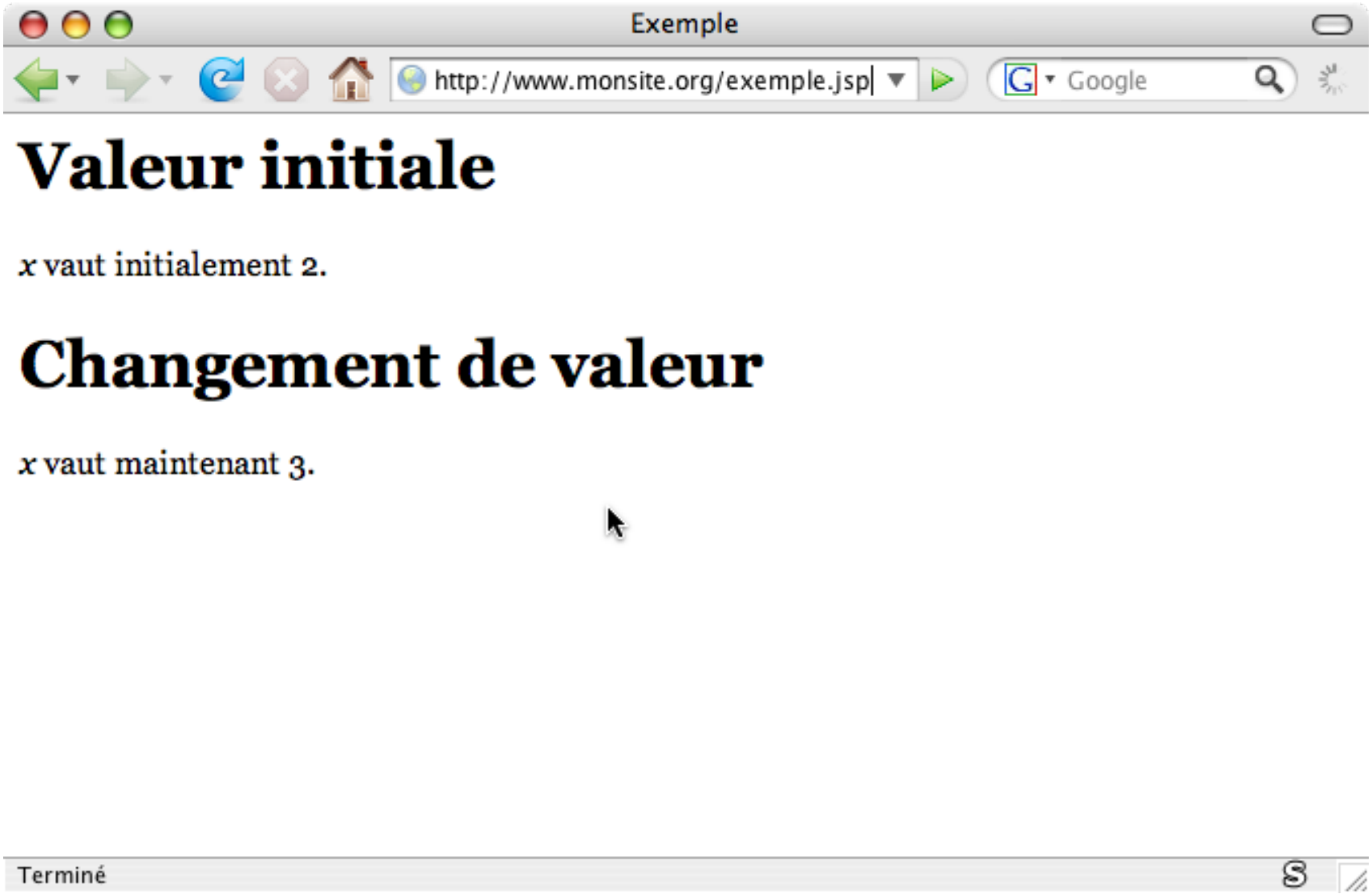
Pages JSP

permet de créer des pages HTML dynamiques
insertion de bouts de code java dans des pages HTML
code exécuté sur le serveur ($\neq$ javascript) : création d'un servlet correspondant

Exemple de page JSP

```
<%@ page language="java" %>
<html>
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <% int x = 2; %>
    <h1>Valeur initiale</h1>
    <p><var>x</var> vaut initialement <%=x%>.</p>
    <h1>Changement de valeur</h1>
    <% x = x + 1; %>
    <p><var>x</var> vaut maintenant <%=x%>.</p>
  </body>
</html>
```

Résultat


Exemple
http://www.monsite.org/exemple.jsp
Google
Valeur initiale
x vaut initialement 2.
Changement de valeur
x vaut maintenant 3.
Terminé

Plan

- Généralités
- Niveau web
- Niveau métier
  - Les Enterprise Java Beans
  - Beans session
  - Beans entité
  - Bean contrôlés par messages
  - XDoclet
- Sécurité

Introduction

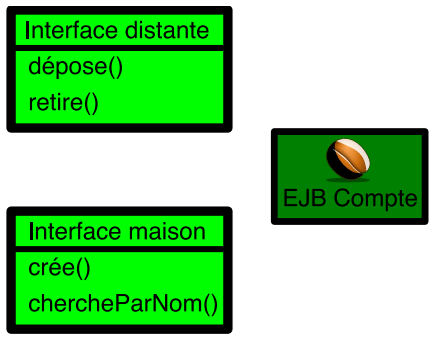
Chaque partie de la logique métier est confiée à un EJB
un EJB définit une interface à travers laquelle il peut être utilisé
3 types d'EJB suivant rôle

Types d'EJB

Bean session (*session bean*) :
effectue une tâche, implante une requête de l'interface avec les clients
Bean entité (*entity bean*) :
représente une donnée dans la base de données.
Bean contrôlé par messages (*message-driven bean*) :
gère les messages asynchrones (qui n'ont pas besoin d'être exécutés tout de suite)

Interfaces

Interface distante (*remote interface*) : fonctionnalités propres à l'EJB
Interface maison (*home interface*) : création, fonctionnalités communes à tous les EJBs correspondants



Deux autres interfaces (*local interface* et *local home interface*) : idem mais uniquement pour être utilisé par des éléments sur la même machine (plus rapide)
Les autres composants et les clients n'accèdent qu'aux fonctionnalités décrites dans les interfaces → fonctionnement interne facilement modifiable

Bean session

communique avec un client :

- ouvre une session interactive
- le client accède aux fonctionnalités de l'interface du bean session
- le bean lance les calculs permettant d'effectuer une requête
- et retourne le résultat au client

Utilisation

un seul client par bean session à la fois
l'état d'un bean session n'est pas persistant : ne contient pas les données à conserver plus de quelques heures
protège le client de la complexité des opérations internes à l'application sur le serveur
deux types de bean session : sans état et avec état

Bean session sans état

ne conserve pas de traces de la conversation avec le client
utilise uniquement les arguments passés lors de la requête
peut être utilisé par plusieurs clients successivement
pas besoin de stockage, même temporaire
peut implémenter un service web
passe plus à l'échelle qu'un bean session avec état

Bean session avec état

état = conversation avec le client
résultat des requêtes dépendant de l'historique des requêtes (ex. : metDansPanier(article) puis affichePanier())
quand le client se termine, le bean session disparaît

Bean entité

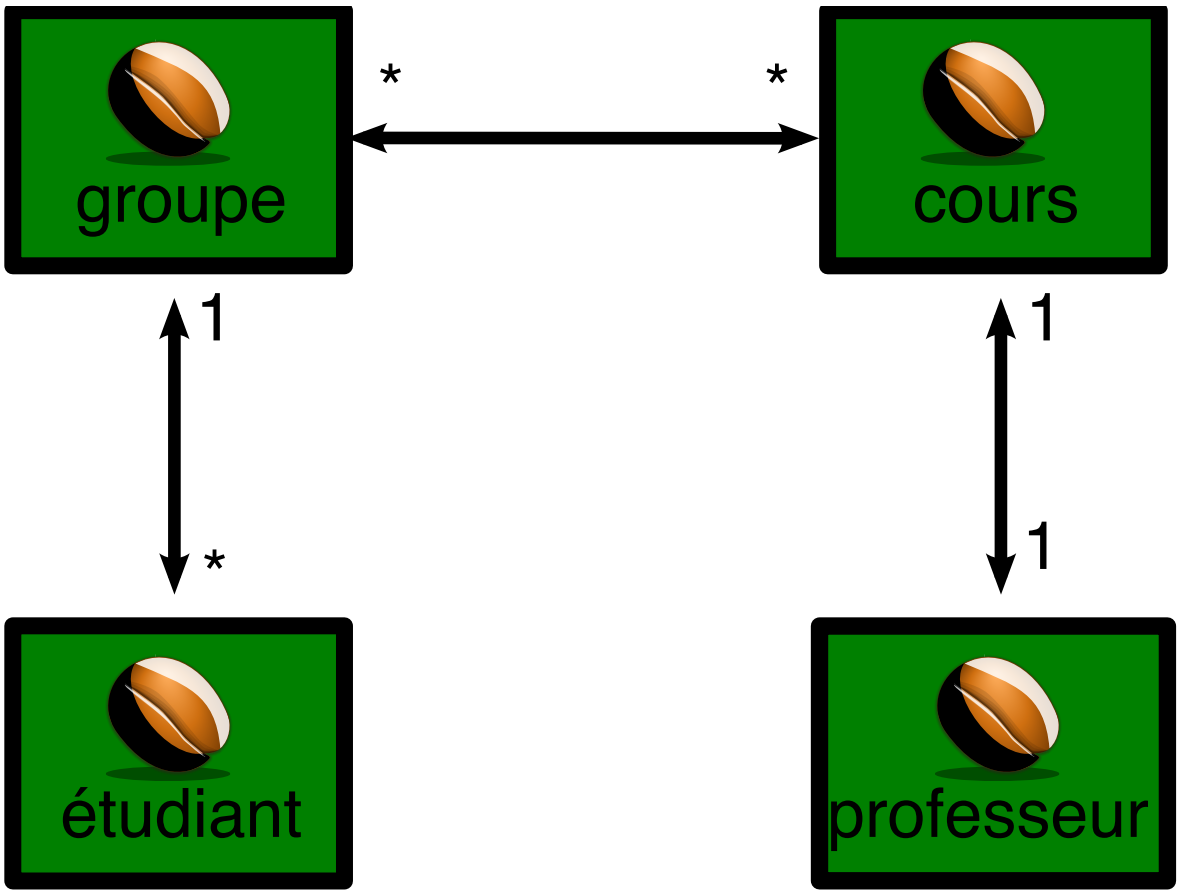
représente une donnée dans la base de données
$\longrightarrow$ persistance
possède un identifiant unique (clef primaire)
peut-être partagé entre plusieurs clients et lié à d'autres beans entité
propose des fonctionnalités de recherche dans la base

Gestion de la persistance

Deux types de persistance :

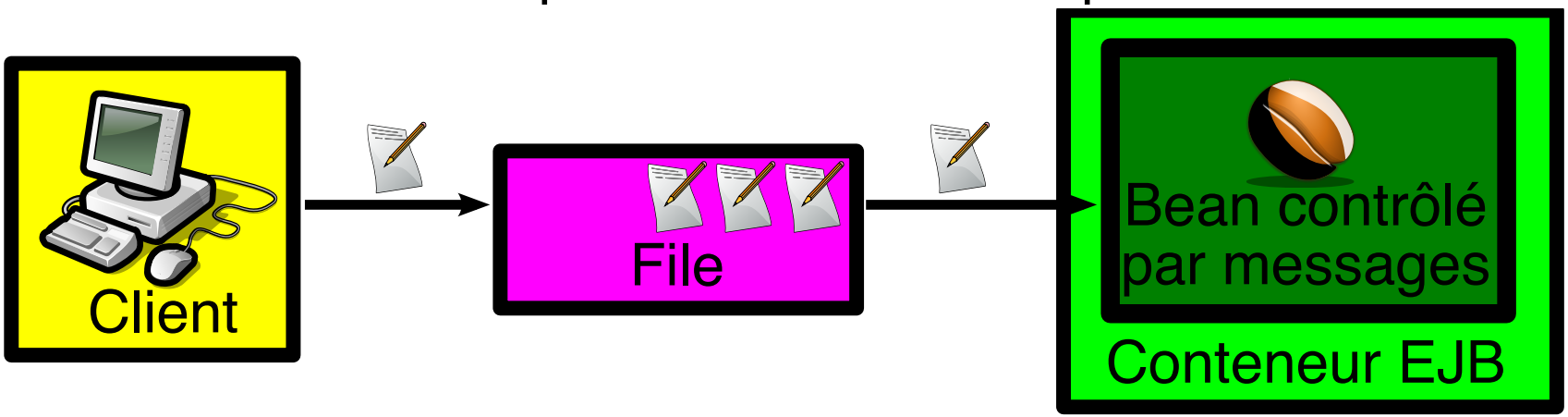
- Persistance gérée par le bean :
  Lien entre bean entité et donnée effective dans base géré par le bean lui-même.
  Code contient requêtes dans la base de données
  Plus de flexibilité
  Moins de portabilité/base de donnée
- Persistance gérée par le conteneur EJB
  Liens entre différents beans entité définis dans conteneur EJB
  Conteneur EJB fait correspondre ces liens avec base de données automatiquement
  pas de dépendances avec base de données effectivement utilisée

Relations entre beans entité



Bean contrôlé par messages

associé à une file (*queue*) qui entasse des messages envoyés par le client
traite les messages de façon asynchrone
plusieurs beans peuvent utiliser la même file (répartition de la charge), plusieurs clients peuvent utiliser la même file
exemples : envoi d'un mail de confirmation, commande à un fournisseur d'un produit bientôt en rupture de stock



Diminuer la redondance : XDoclet

pour définir un bean, besoin d'écrire 5 classes :
bean, interface distante, interface maison, interface locale, interface locale maison
beaucoup de redondances car fonctionnalités proposées par interfaces sont celles du bean
solution : XDoclet
seule la classe pour le bean est écrite, annotées avec des commentaires XDoclets
XDoclet génère ensuite les autres classes à partir des commentaires
permet également de gérer lien entre bean entité et base de données, relations entre beans entités

Exemple d'annotations

```
/**
 * @ejb.home-method view-type="remote"
 */
public void trouveParNom() {
   ...
}

/**
 * @ejb.interface-method view-type="local"
 */
public void depose(float somme) {
   ...
}
```

Plan

- Généralités
- Niveau web
- Niveau métier
  - Les Enterprise Java Beans
  - Beans session
  - Beans entité
  - Bean contrôlés par messages
  - XDoclet
- Sécurité

Domaines, utilisateurs, groupes, rôles

Utilisateurs et groupes à la UNIX
Exemple de groupes : étudiants, professeurs
Accès aux ressources : rôles
Exemple : administration, consultation
Le tout forme un domaine (*realm*)