

Architectures client-serveur orientées Web et données

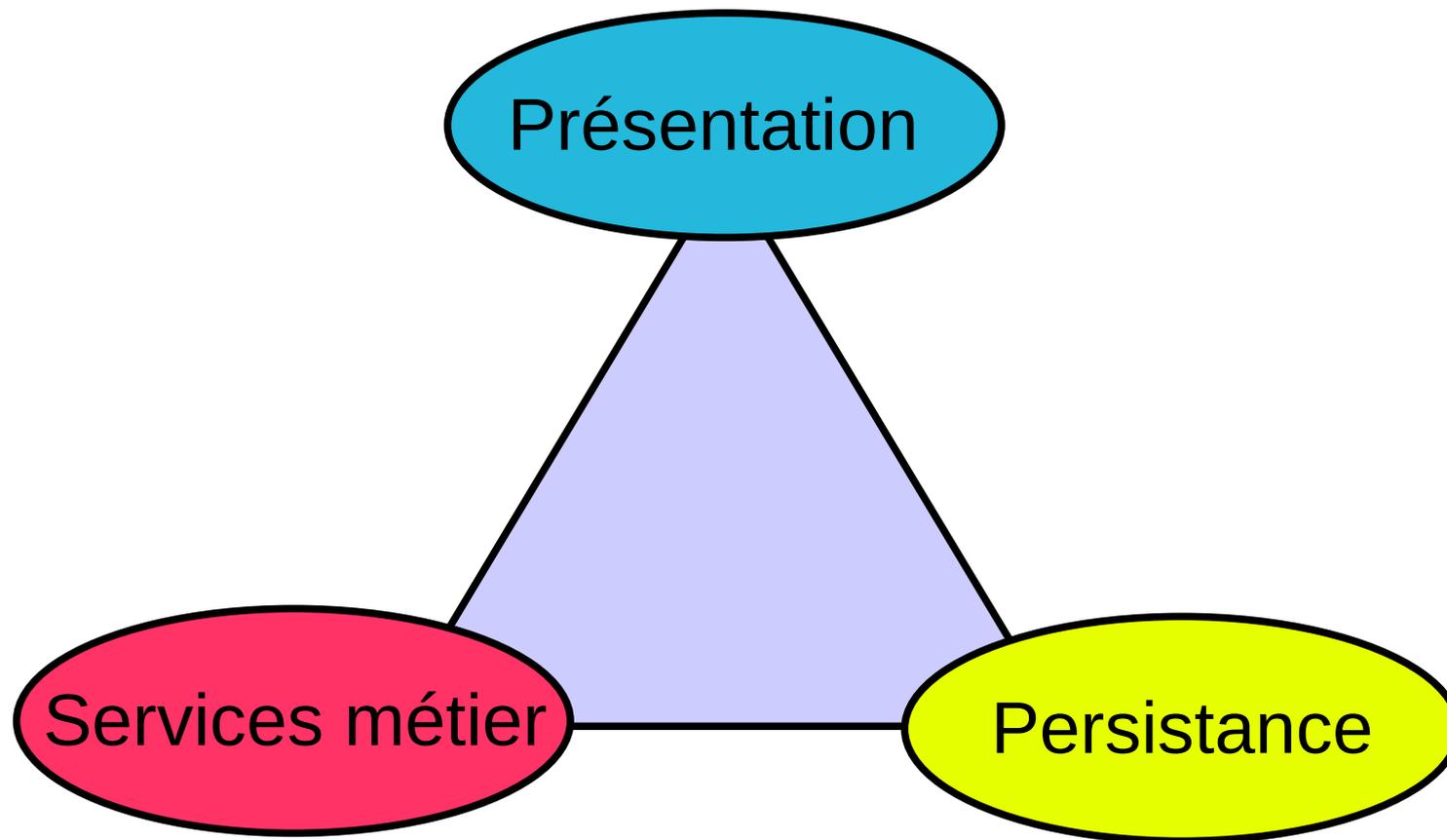
Master TIIL & ILIADE 1^{ère} année

Eric Cariou

*Université de Bretagne Occidentale
UFR Sciences et Techniques – Département Informatique*

Eric.Cariou@univ-brest.fr

Triptyque d'une application



Triptyque d'une application

◆ Présentation

- ◆ Interface utilisateur pour interagir avec l'application
 - ◆ Interface classique type GUI (ex : traitement de texte)
 - ◆ Interface Web, plus légère

◆ Persistance

- ◆ Enregistrement sur support physique des données de l'application
 - ◆ Fichiers (texte, binaire, XML, JSON ...)
 - ◆ Base de données relationnelles
 - ◆ Simple
 - ◆ Avec redondance pour fiabilité
 - ◆ Multiples : fédération de bases de données
 - ◆ Base de données noSQL/newSQL
 - ◆ Orientées documents, graphes ..

Triptyque d'une application

- ◆ Services métier
 - ◆ Partie applicative
 - ◆ Intègre la logique métier
 - ◆ Ex: un document est composé de sections, elles mêmes composées de sous-sections ...
 - ◆ Services offerts aux utilisateurs
 - ◆ Ex: créer un document, le modifier, ajouter des sections, l'enregistrer ...
- ◆ Trois parties
 - ◆ Sont intégrées et coopèrent pour le fonctionnement de l'application
 - ◆ En anglais, on les appelle aussi des « tiers » (étages)
 - ◆ Application « 3 – tiers » quand les 3 parties sont clairement distinctes

Triptyque d'une application

- ◆ Dans un contexte distribué
 - ◆ Les tiers sont / peuvent être exécutés sur des machines différentes
 - ◆ Certains tiers peuvent être sous-découpés
 - ◆ De nombreuses variantes de placement des tiers et de leur distribution
- ◆ Modèle centralisé
 - ◆ Tout est sur la même machine

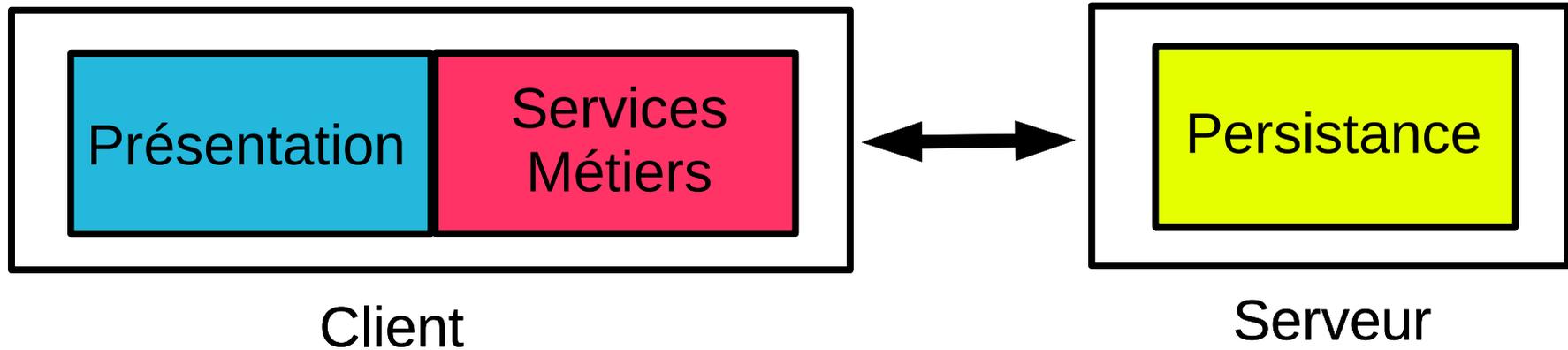


Architecture 2 – tiers

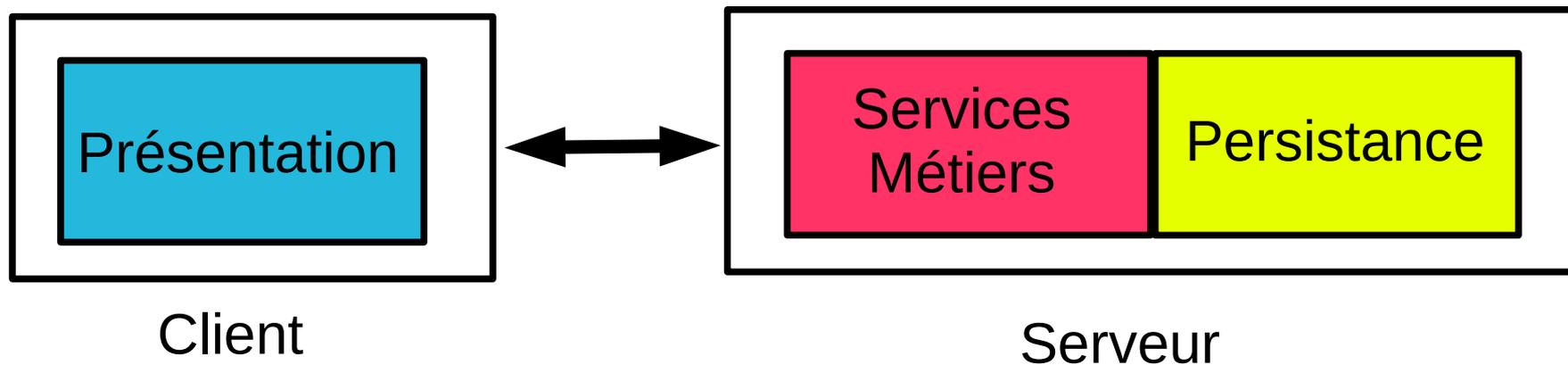
- ◆ Architecture 2 – tiers
 - ◆ Client / serveur de base, avec 2 éléments
 - ◆ Client : présentation, interface utilisateur
 - ◆ Serveur : partie persistance, gestion physique des données
 - ◆ Les services métier / la partie applicative peuvent être
 - ◆ Soit entièrement coté client, intégrés avec la présentation
 - ◆ La partie serveur ne gère que les données
 - ◆ Ex : traitement de texte avec serveur de fichiers distants
 - ◆ Ex : application accédant à une BDD distante
 - ◆ Soit entièrement coté serveur
 - ◆ La partie client ne gère que l'interface utilisateur
 - ◆ L'interface utilisateur peut même être exécutée sur le serveur
 - ◆ Fonctionnement mode terminal / mainframe
 - ◆ L'utilisateur a simplement devant lui écran / clavier / souris pour interagir à distance avec l'application s'exécutant entièrement sur le serveur
 - ◆ Soit découpés entre la partie serveur et la partie client

Architecture 2 – tiers

- ◆ Client : présentation + applicatif



- ◆ Serveur : applicatif + gestion données

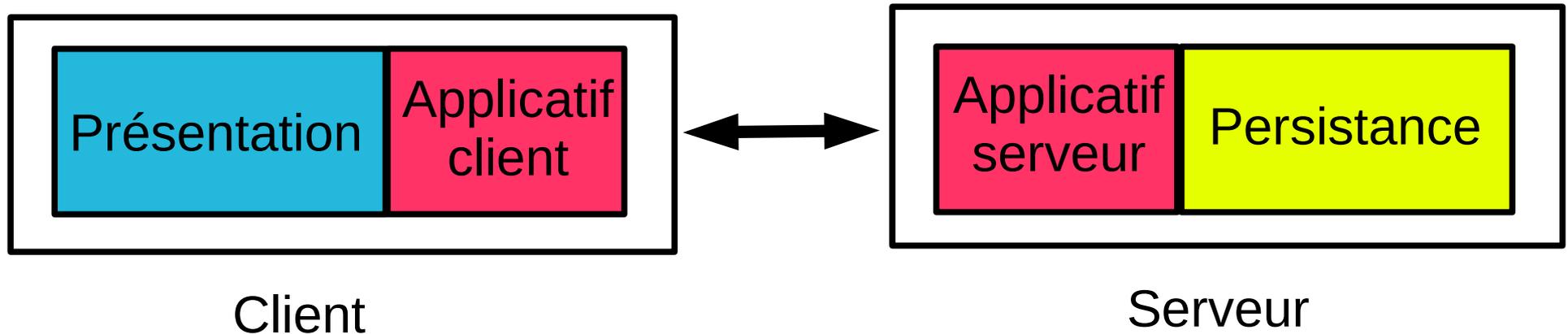


Architecture 2 – tiers

- ◆ Terminal : client intègre un minimum de la partie présentation

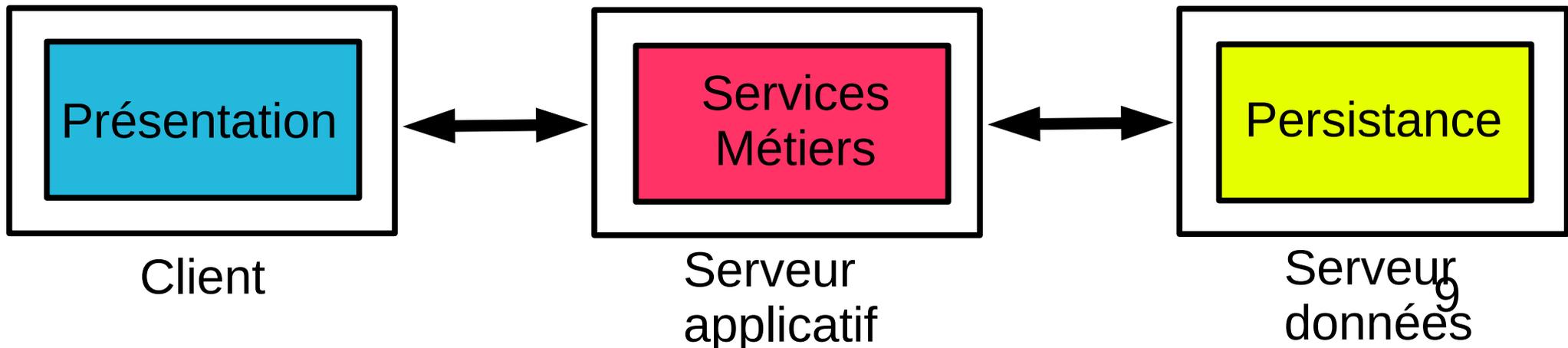


- ◆ Applicatif : découpé entre client et serveur



Architecture 3 – tiers

- ◆ Architecture 3 – tiers
 - ◆ Les 3 principaux tiers s'exécutent chacun sur une machine différente
 - ◆ Présentation
 - ◆ Machine client
 - ◆ Applicatif / métier
 - ◆ Serveur d'applications
 - ◆ Persistance
 - ◆ Serveur de (base de) données



Architecture 3 – tiers sur le web

- ◆ Architecture 3 – tiers
 - ◆ Très développée de nos jours
 - ◆ Avec généralement un fonctionnement au dessus du Web
 - ◆ Couche présentation
 - ◆ Navigateur web sur machine cliente
 - ◆ Client léger
 - ◆ Affichage de contenu HTML
 - ◆ Couche applicative / métier
 - ◆ Serveur d'applications
 - ◆ Serveur HTTP exécutant des composants / éléments logiciels qui génèrent dynamiquement du contenu HTML ou des données
 - ◆ Via des requêtes à des BDD distantes
 - ◆ Couche persistance
 - ◆ Serveur(s) de BDD de données

Architecture n – tiers

- ◆ Architecture n – tiers
 - ◆ Rajoute des étages / couches en plus
 - ◆ La couche applicative n'est pas monolithique
 - ◆ Peut s'appuyer et interagir avec d'autres services
 - ◆ Composition horizontale
 - ◆ Service métier utilise d'autres services métiers
 - ◆ Composition verticale
 - ◆ Les services métiers peuvent aussi s'appuyer sur des services techniques
 - ◆ Sécurité,
 - ◆ Transaction
 - ◆ ...
 - ◆ Chaque service correspond à une couche
 - ◆ D'où le terme de N-tiers

Architecture n – tiers

- ◆ Intérêts d'avoir plusieurs services / couches (3 ou plus)
 - ◆ Réutilisation de services existants
 - ◆ Découplage des aspects métiers et technique et des services entre eux : meilleure modularité
 - ◆ Facilite évolution : nouvelle version de service
 - ◆ Facilite passage à l'échelle : évolution de certains services
 - ◆ On recherche en général un couplage faible entre les services
 - ◆ Permet de faire évoluer les services un par un sans modification du reste de l'application
- ◆ Inconvénients
 - ◆ En général, les divers services s'appuient sur des technologies très variées : nécessite de gérer l'hétérogénéité et l'interopérabilité
 - ◆ Utilisation de framework / outils supplémentaires
 - ◆ Les services étant plus découpés et distribués, pose plus de problèmes liés à la distribution

Logique de présentation

- ◆ Les tâches liées à la présentation requièrent
 - ◆ L'interaction avec l'utilisateur
 - ◆ La logique de présentation
 - ◆ Le traitement des données retournées par les services métiers et leur présentation à destination de l'utilisateur
- ◆ Pour un client lourd
 - ◆ Interaction avec utilisateur
 - ◆ Réalisée par la GUI (Graphic User Interface) d'un client lourd : boutons, listes, menus, zones graphiques ...
 - ◆ Toute la puissance d'une librairie de widgets dédiée dans un langage de programmation
 - ◆ Ex : Java FX
 - ◆ Logique de présentation
 - ◆ Client lourd fait directement l'appel des services métiers sur le serveur et met en forme les données retournées dans la GUI

Logique de présentation

- ◆ Pour un client Web
 - ◆ Fonctionnement basique
 - ◆ Interaction avec l'utilisateur : liens vers des URLs, formulaires ...
 - ◆ Logique de présentation
 - ◆ Navigateur se contente d'afficher du code HTML qui ne peut pas être statique ici vu que le contenu dépend des données récupérées en BDD
 - ◆ Serveur appelle les services métiers qui renvoient les données et met lui-même en forme les données dans le code HTML retourné au client
 - ◆ Réalisé de préférence dans une couche à part sur le serveur
 - ◆ Fonctionnement évolué grâce à la généralisation de Javascript et nouvelles normes HTML/CSS
 - ◆ Possibilité d'exécuter du code coté client et interactions plus riches
 - ◆ Affichage de la page peut varier selon l'interaction avec l'utilisateur
 - ◆ Client peut exécuter une partie de la logique de présentation
 - ◆ Y compris de manière dynamique : requêtes pour récupérer des données sur le serveur et les insérer dans la page affichée
 - ◆ Ex : technologies AJAX ou WebSocket
 - ◆ Nombreux frameworks JS : Angular, React, Vue ...

Persistence

- ◆ Couche de persistance
 - ◆ Stockage et manipulation des données de l'application
 - ◆ Supports physiques variés
 - ◆ Fichiers binaires
 - ◆ Fichiers textes « de base » ou structurés (JSON, XML)
 - ◆ Base de données
 - ◆ Relationnelles ou un ensemble de bases de données relationnelles
 - ◆ noSQL (orientées documents ou graphes)
 - ◆ Pour des BDD relationnelles
 - ◆ Nécessité d'envoyer à distance des requêtes de type SQL et d'en récupérer les résultats
 - ◆ Pour réaliser cela
 - ◆ Soit c'est natif dans le langage utilisé (ex : PHP)
 - ◆ Soit on passe par des frameworks ou des API dédiés

Persistance

- ◆ Quelques standards / outils d'accès à distance à des BDD
 - ◆ RDA (Remote Data Access) de l'ISO
 - ◆ ODBC (Open Data Base Connectivity) de Microsoft
 - ◆ JDBC (Java Data Base Connectivity) d'Oracle
 - ◆ Framework pour le langage Java
 - ◆ Fonctionnement général
 - ◆ Gestion de requêtes SQL mais avec indépendance du SGBDR utilisé (mySQL, PostgreSQL, Oracle ...)
 - ◆ En général, seule la phase de connexion au SGBDR est spécifique
- ◆ Pour des fichiers XML, outils gérant la navigation dans l'arbre de données
 - ◆ DOM : norme W3C permettant de naviguer et modifier un contenu XML ou HTML
 - ◆ Utilisé notamment coté client Web avec du Javascript
 - ◆ JAXP : framework JAVA pour lecture/édition fichiers XML

Persistence

- ◆ En programmation orientée objet, les données
 - ◆ Sont des objets avec des attributs valués et des références vers d'autres objets
 - ◆ Forment une structure globale sous forme de graphe
- ◆ Données stockées dans des supports externes
 - ◆ Schéma relationnel pour SGBDR
 - ◆ Arbre pour XML
- ◆ Problème de différence de format de stockage et de manipulation de données coté programme
 - ◆ Doit écrire du code qui permet de passer d'un objet au format de stockage physique
 - ◆ Peut être rapidement lourd et répétitif à faire

Persistence

- ◆ Peut à la place utiliser des frameworks de plus haut niveau
 - ◆ Correspondances objet-relationnel
 - ◆ ORM : Object-Relationnal Mapping
 - ◆ Sérialisation XML
- ◆ Principes
 - ◆ On définit la correspondance entre la structure des classes objet et le schéma relationnel/XML
 - ◆ On manipule directement des objets dans le code
 - ◆ Le framework fait la lecture/enregistrement du contenu des objets sur le support physique
 - ◆ Plus besoin de coder des requêtes SQL
 - ◆ Même si un langage de requête orienté objet reste utile
- ◆ Exemples
 - ◆ JPA (Java Persistence API) ou Hibernate pour ORM en Java
 - ◆ JAXB pour sérialisation XML en Java
 - ◆ JSONB pour sérialisation JSON en Java

Frameworks globaux

- ◆ Une application 3/N – tiers intègre un grand nombre de technologies
 - ◆ Présentation : HTML/CSS, framework graphiques Javascript...
 - ◆ Applicatif : objets, composants, scripts exécutables, services ...
 - ◆ Données : fichiers XML, JSON, BDD SQL, BDD noSQL...
 - ◆ Protocoles de communication : RPC/RMI, HTTP, messages, ...
- ◆ Pour faciliter l'intégration de ces technologies et le développement d'applications
 - ◆ Éditeurs offrent des frameworks globaux
 - ◆ Java EE chez Oracle
 - ◆ .Net chez Microsoft
 - ◆ Serveur d'application
 - ◆ Serveur permettant d'exécuter les parties applicatives dans le contexte de ces frameworks

Oracle Java EE

- ◆ Java EE : Java Entreprise Edition (devenue Jakarta EE)
 - ◆ Norme / standard défini par Oracle pour le langage Java
 - ◆ Technologies intégrées
 - ◆ Composants logiciels : EJB
 - ◆ Applications orientées Web : JSP, Servlet, JSTL
 - ◆ Communication à distance : Java RMI, IIOP, JMS (Java Message Service : communication par message), Web Services
 - ◆ Gestion données distantes : JDBC, JPA
 - ◆ Gestion d'annuaires (type LDAP) : JNDI
 - ◆ Interfaces graphiques : Java FX
 - ◆ Et bien d'autres ...
- ◆ Existe nombreux serveurs d'applications Java EE
 - ◆ Versions libres
 - ◆ GlassFish, JBoss, Apache Geronimo, Jonas ...
 - ◆ Versions d'éditeurs
 - ◆ Oracle GlassFish Server Open Source, IBM WebSphere, BEA WebLogic ...

Microsoft .Net

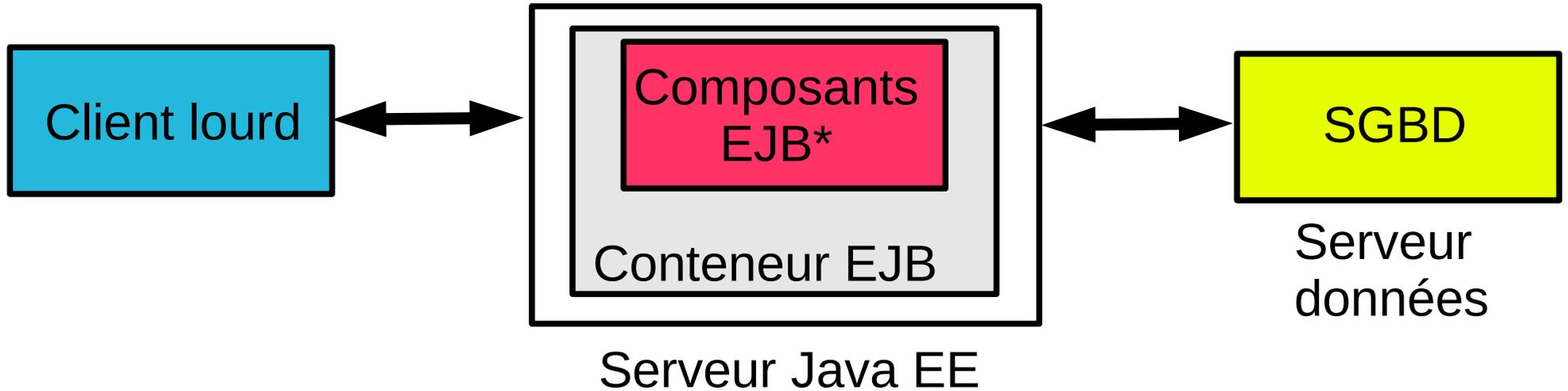
- ◆ Solution Microsoft similaire à Java EE
 - ◆ Particularité : multi-langage
 - ◆ Permet interopérabilité d'éléments écrits en C, Java, C#, J#, Eiffel, VB, ... (plus de 20 langages)
 - ◆ Traduction en code intermédiaire (MSIL) qui sera exécuté par la couche CLR (Common Language Runtime)
 - ◆ Coté Java, c'est le code Java qui était converti en byte code exécuté par la machine virtuelle Java (JVM)
 - ◆ C'est une norme également
 - ◆ La principale mise en œuvre est bien sûr de Microsoft et pour Windows, mais il existe quelques versions libres (implémentations souvent partielles)
- ◆ Technologies intégrées
 - ◆ Composants logiciels : COM+
 - ◆ Applications orientées Web : ASP .Net,
 - ◆ Communication à distance : .Net remoting, MSMQ, Web services
 - ◆ Accès données : ADO .Net, ODBC
 - ◆ ...

Architecture 3/4-tiers, contexte Java EE

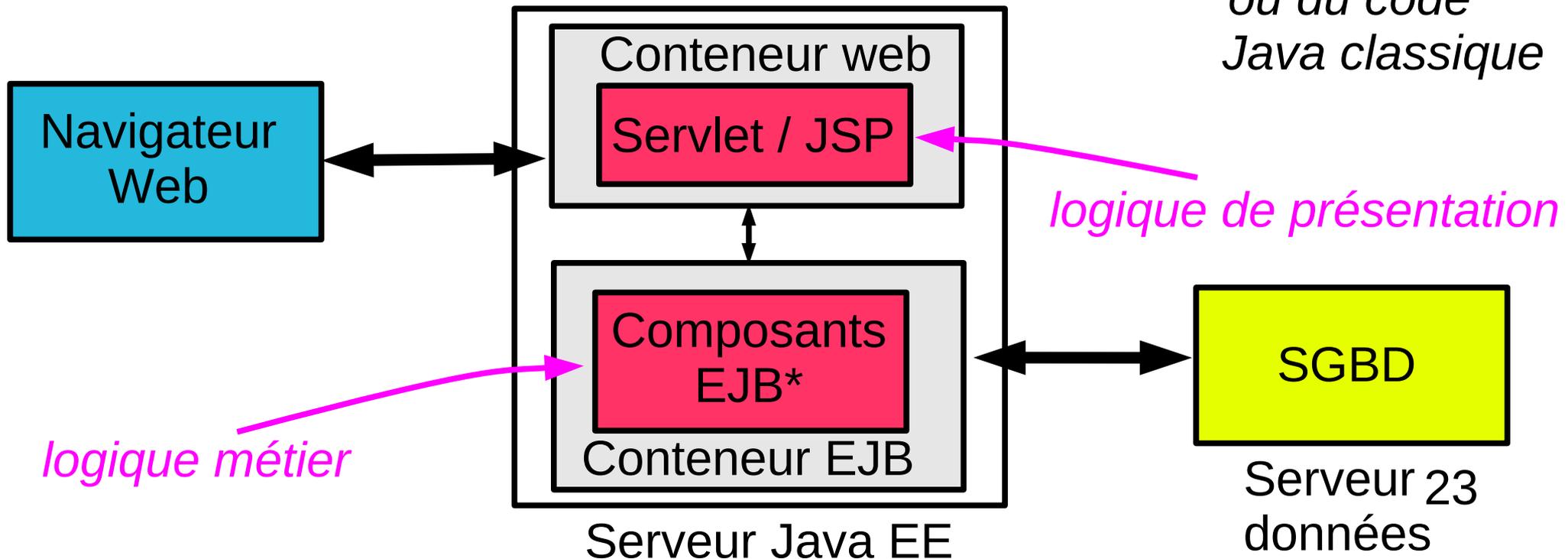
- ◆ Deux architectures générales contexte Java EE
 - ◆ Logique applicative
 - ◆ Réalisée par composants EJB (ou du code Java classique)
 - ◆ Communication via JPA ou JDBC pour attaquer BDD distante
 - ◆ Présentation
 - ◆ Avec client léger ou client lourd
 - ◆ Client léger : navigateur web
 - ◆ Intérêt : simplifie la présentation (suffit d'un navigateur)
 - ◆ Inconvénient : limite de l'interaction via HTML même si de plus en plus puissant grâce à Javascript et librairies associées
 - ◆ Client lourd
 - ◆ Application « standard » coté client, gère la logique de présentation
 - ◆ Intérêt : plus grande possibilité en terme de présentation et d'interaction
 - ◆ Inconvénient : nécessite un développement dédié via des API de widgets
 - ◆ Interaction avec la partie applicative sur le serveur
 - ◆ Via JSP / Servlet pour un client léger pour gérer les requêtes HTTP
 - ◆ Direct si client lourd (via un middleware type RMI)

Architecture 3/4-tiers, contexte Java EE

◆ Client lourd (3-tiers)



◆ Client léger (4-tiers)

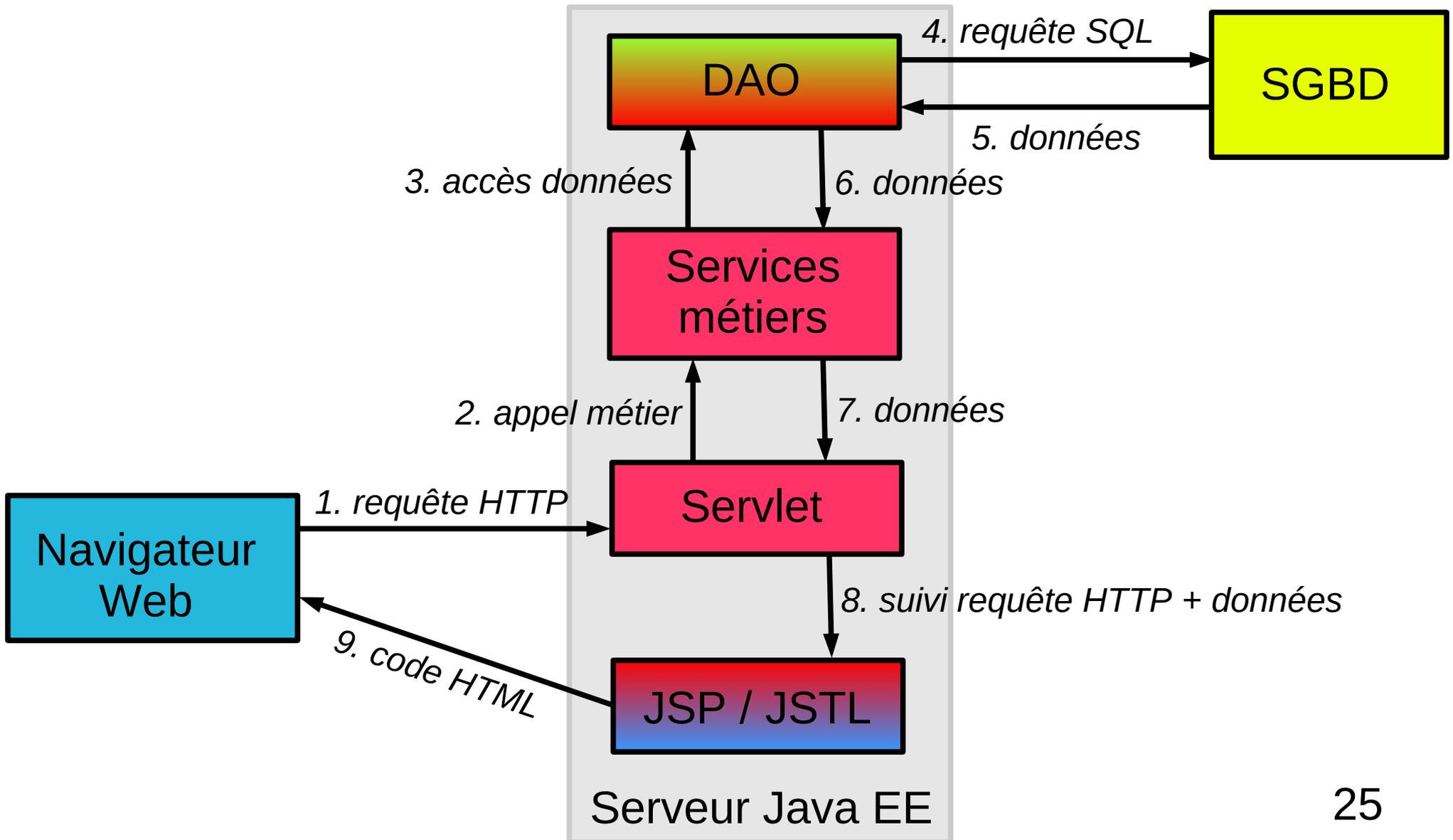


Architectures micro-services

- ◆ Application = ensemble de services interconnectés
 - ◆ Service léger : une fonctionnalité bien spécifique
 - ◆ Défini par une API : Application Programming Interface
 - ◆ Services faiblement couplés
 - ◆ Pas d'état/session entre les services
 - ◆ Services facilement substituables
 - ◆ Plus de flexibilité qu'avec une architecture N-tiers
- ◆ Utilise généralement le style architectural REST
 - ◆ Un service est identifié par une URL HTTP
 - ◆ Utilise les verbes HTTP pour définir l'action à réaliser
 - ◆ GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS, TRACE
 - ◆ L'URL ou la requête HTTP contient les éventuels paramètres
 - ◆ Ex : <http://www.annuaire.fr/cariou/eric>
 - ◆ Format des données de tout type mais souvent en JSON

Architecture Web type basique

- ◆ Architecture Web type avec des servlets



Architecture Web type basique

1. Client envoie une requête HTTP au serveur
 - ◆ Clic sur une URL, soumission d'un formulaire...
2. Quand la servlet reçoit la requête HTTP
 - ◆ Identifie la demande du client, récupère données du formulaire...
 - ◆ Puis appelle le service métier requis
3. Couche métier utilise un DAO pour accéder aux données
 - ◆ DAO : Data Access Object
 - ◆ Objet/couche dédié à l'accès aux données
 - ◆ On évite que la couche métier fasse directement des requêtes sur la BDD
4. DAO fait une requête SQL sur le SGBDR
 - ◆ Soit codée directement dans le DAO (si JDBC)
 - ◆ Soit réalisée indirectement par l'ORM (si JPA)

Architecture Web type basique

5. DAO récupère le résultat de la requête SQL
 - ◆ Retravaille au besoin le résultat pour notamment le mettre sous forme objet si usage de JDBC
6. DAO retourne les données à la couche métier
7. Couche métier retourne les données à la servlet
8. Servlet fait appel à une page JSP
 - ◆ Fait suivre la requête HTTP (pour pouvoir répondre au client)
 - ◆ Associe les données à la requête
9. Page JSP génère du code HTML renvoyé au client
 - ◆ Met en forme les données retournées dans du HTML
 - ◆ Beaucoup plus facile à faire dans une page JSP que par la servlet directement
 - ◆ Usage de JSTL pour faciliter traitement données

***Contenu du module
« Systèmes d'information »***

Dév. Java orienté services et données

- ◆ Persistence
 - ◆ Relationnel (SQL)
 - ◆ JDBC : accès basique en SQL à BDD relationnelles
 - ◆ JPA : framework de mapping objet-relationnel
 - ◆ noSQL : BDD orientée documents avec Mongo.db
 - ◆ Sérialisation d'objets en XML/JSON (avec JAXB et JSON-B)
- ◆ Serveur Web applicatif
 - ◆ Servlet : programme Java traitant des requêtes HTTP
 - ◆ JSP/JSTL : variante des servlets mixant code HTML et code Java
 - ◆ API REST
- ◆ Présentation, client Web
 - ◆ Framework Javascript Vue.js
- ◆ Spring Boot
 - ◆ Framework Java orienté composants pour intégration à plus haut niveau de plusieurs technologies (services, Web, données...)

