UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR

ECOLE SUPÉRIEURE POLYTECHNIQUE

DÉPARTEMENT GÉNIE INFORMATIQUE

ARCHITECTURE LOGICIELLE

COURS INTRODUCTIF

Formateur

Dr Mouhamed DIOP

mouhamed.diop@esp.sn

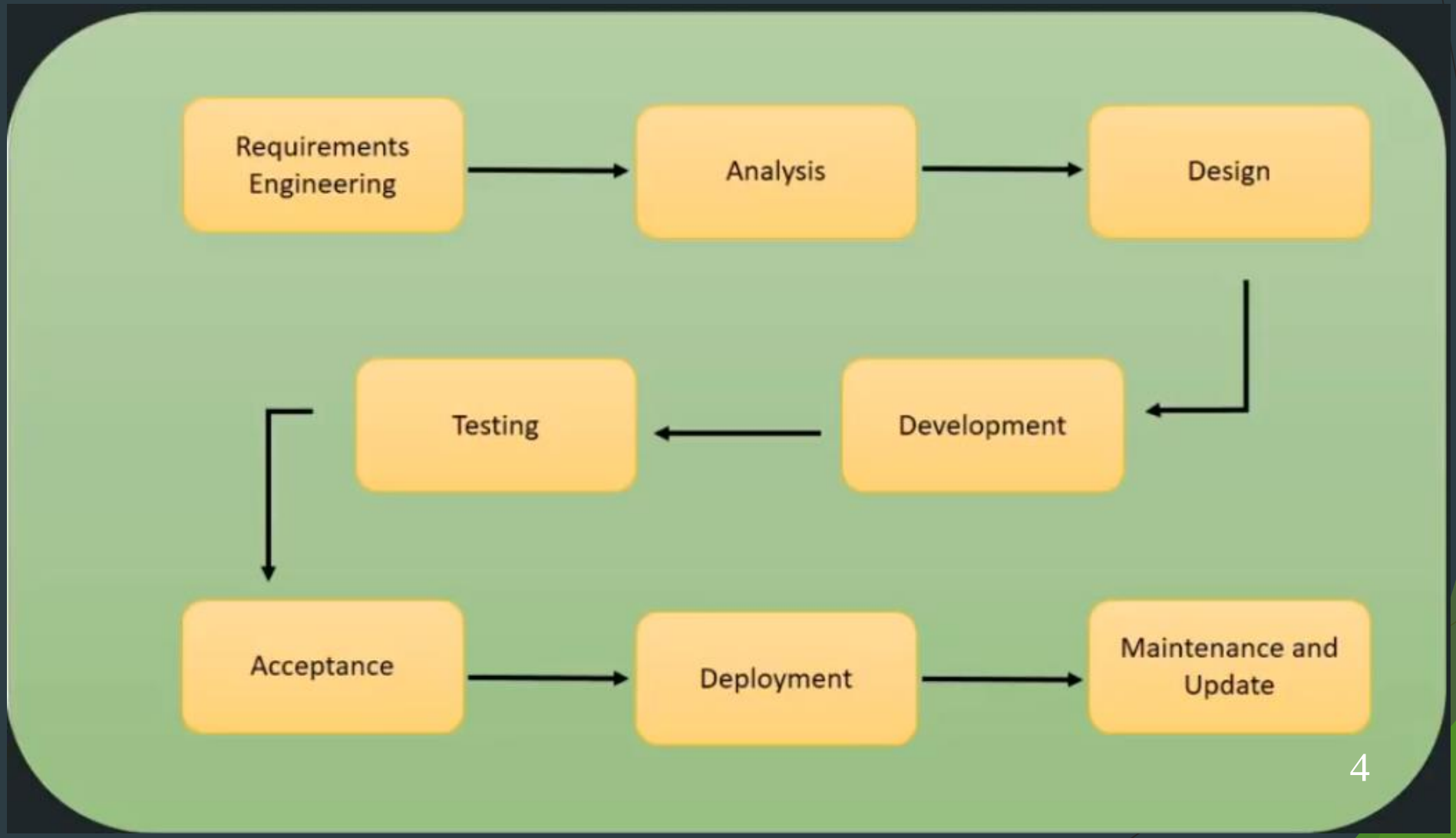
Objectifs

- A l'issue de ce cours, l'étudiant devra :
  - Comprendre la notion d'architecture logicielle
    - Pouvoir situer sa définition dans le processus de développement logiciel
  - Connaitre les différentes couches d'une architecture d'application
  - Connaitre la notion de style architectural (architecture pattern)
    - Pouvoir donner un aperçu des principaux styles architecturaux
  - Connaitre la différence entre patrons de conception (design patterns) et style architectural
  - Savoir découper une application en couches (vue logique) et en niveaux (vue physique)

Introduction

- La construction d'un logiciel ne se limite pas au codage
  - Le développement d'un logiciel se fait en plusieurs étapes
    - La programmation n'est qu'une partie du processus de développement logiciel
- Plusieurs modèles de développement existent…
  - Modèles en V ou en cascade
  - Modèles agiles
  - Etc.
- … et se partagent principalement un certain nombre de phases

Les principales phases de développement logiciel



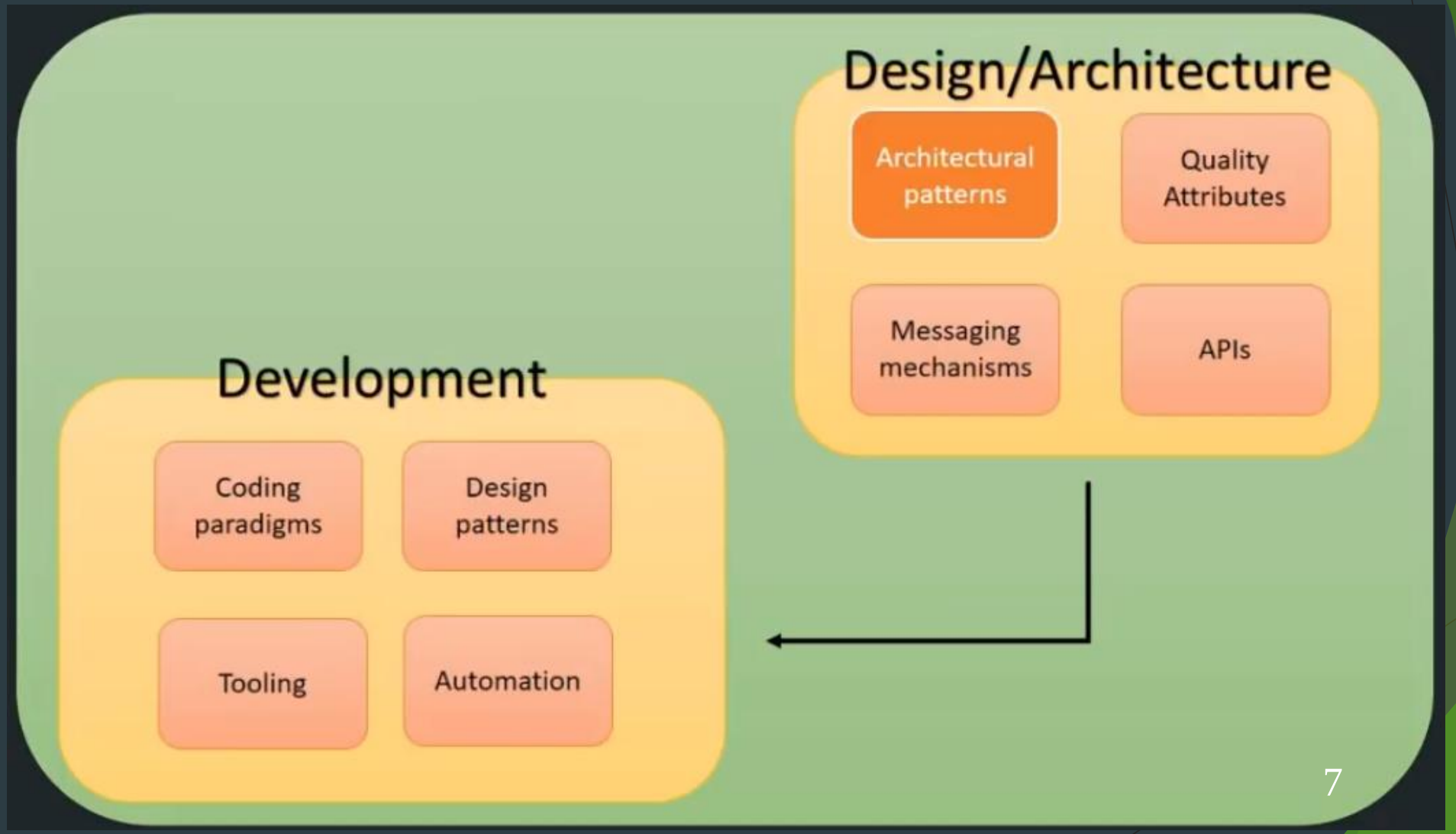
L'architecture logicielle (1/2)

- La spécification de l'architecture logicielle intervient durant la conception du logiciel et définit :
  - Le style architectural
    - Quels sont les composants du logiciel et comment ils sont organisés et assemblés ?
  - Les interfaces de communication (API)
    - Comment les composants communiquent entre eux ?
  - Les contraintes de qualité
    - Les contraintes non fonctionnelles que le système doit satisfaire
      - portabilité, robustesse, montée en charge, etc.

L'architecture logicielle (2/2)

- L'architecture d'un logiciel définit donc :
  - comment les différents composants d'un système logiciel sont organisés et assemblés,
  - comment ces composants communiquent-ils ensemble,
  - quelles sont les contraintes de qualité qui pèsent sur le système
- Les livrables obtenus durant la conception sont les entrées (input) de la phase de développement
  - L'architecture va servir de guide au développeur durant la phase de développement

Conception de l'architecture VS Implémentation



DEFINITION : sens littéral (1/2)

- Dans le sens originel qu'il a en construction civile, l'architecture nécessite :
  - Une compréhension profonde des besoins du bâtiment présents et futurs et des contraintes de l'environnement, y compris les contraintes sociales
  - Une connaissance approfondie des matériaux à assembler ainsi que leurs limites (usinage, vieillissement, rupture, etc.)
  - Une connaissance approfondie des procédés de construction (les méthodes du génie civil, la gestion de projet, l'acquisition) permettant d'organiser l'activité des différents corps de métiers qui concourent à la réalisation et au maintien en conditions du bâtiment.

DEFINITION : sens littéral (2/2)

- Dans ce contexte, l'architecte est :
  - Celui qui rassemble et sait mettre en œuvre ces différentes connaissances au service exclusif du but poursuivi par la maitrise d'ouvrage
  - Un chef d'orchestre qui doit savoir jouer de plusieurs instruments et qui doit surtout connaitre la musique

DEFINITION : en génie logiciel (1/2)

- L’architecture d’un programme ou d’un système informatique est la structure (ou les structures) du système qui comprend les éléments logiciels, leurs propriétés visibles et leur relations
- C’est l’organisation fondamentale d’un système représenté par ses composants, la relation des composants du système les uns avec les autres et avec l’environnement, et les principes et les directives pilotant leur conception et leur évolution [IEEE 1471]
- La phase de conception logicielle est l'équivalent, en informatique, à la phase de conception en ingénierie traditionnelle (mécanique, civile ou électrique)
  - cette phase consiste à réaliser entièrement le produit sous une forme abstraite avant la production effective.

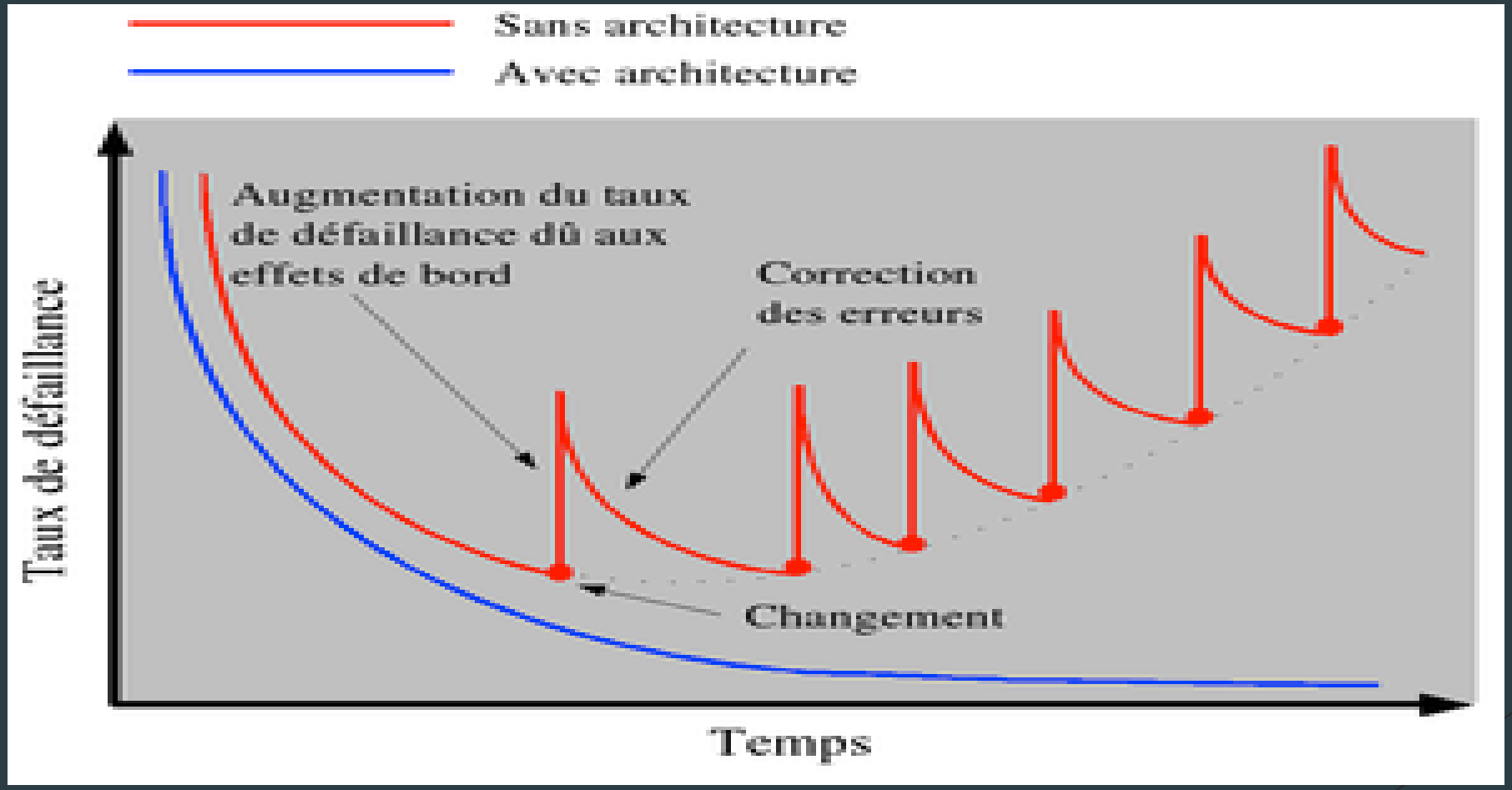
DEFINITION : en génie logiciel (2/2)

- Le modèle d'architecture est produit durant la phase de conception du logiciel
- Il ne décrit pas ce que doit réaliser un système (analyse fonctionnelle), mais plutôt comment il doit être conçu de manière à répondre aux spécifications
- L'analyse fonctionnelle décrit le « quoi faire » tandis que l'architecture logicielle décrit le « comment faire »

Objectifs visés

- Les deux objectifs principaux de toute architecture logicielle :
  - la réduction des coûts, réalisée par la réutilisation de composants logiciels (correction d'erreurs et adaptation du logiciel).
  - l'augmentation de la qualité du logiciel, se trouve distribuée à travers plusieurs critères
    - Portabilité, validité, maintenabilité, réutilisabilité, etc.
- Plus d’exigences dans la question posée
  - Non pas comment créer un logiciel qui fonctionne (débutant)
  - Mais, comment créer un logiciel qui fonctionne de manière optimale, tout en respectant les critères de qualité (expert)

Objectifs visés



Architecture d'un système

- L'architecture d'un système est sa ***conception*** de haut niveau
- N'importe quel système complexe est composé de s***ous systèmes*** qui interagissent entre eux
- La conception de haut niveau est le processus qui consiste à ***identifier*** ces sous-systèmes ainsi que les ***relations*** qu'ils entretiennent entre eux
- ***L'architecture*** d'un système est le résultat de ce processus

Architecture d'un système

- L'architecture implique plusieurs choix dont les ***technologies***, les ***produits*** et les ***serveurs*** à utiliser
- Il n'y a pas une architecture unique permettant de réaliser le système, il y en a plusieurs.
- Le ***concepteur*** ou ***l'architecte*** tâchera de choisir la meilleure architecture possible selon plusieurs critères dont la nature du projet, les compétences de l'équipe, les budgets et outils disponibles, etc.

Représentation des architectures

- Il existe ***plusieurs*** représentations graphiques des architectures
- Une des représentations utilisées est la représentation ***C&C*** : ***Composants*** et ***Connecteurs***
  - Un composant est un ***module logiciel*** (application, bibliothèque, module, etc.) ou un ***entrepôt de données*** (base de données, système de fichiers, etc.)
  - Le connecteur représente les ***interactions*** entre les composants
- La représentation C&C est un ***graphe*** contenant des composants et des connecteurs

Composants

- Un composant est un module logiciel ou un entrepôt de données
- Un composant est identifié par son ***nom*** qui indique son ***rôle*** dans le système
- Les composants communiquent entre eux en utilisant des ***ports*** (ou ***interfaces***)
- Les architectures utilisent des composants standards : ***serveurs***, ***bases de données***, ***application***, ***clients***, etc.

Serveur et Client

- Un serveur est un ***module*** logiciel qui répond aux requêtes d'autres modules appelés ***clients***
- Généralement, les services et les clients sont hébergés dans des machines différentes et communiquent via le ***réseau*** (intranet / internet)
- Par exemple, un service HTTP répond aux requêtes des clients qui sont les navigateurs web

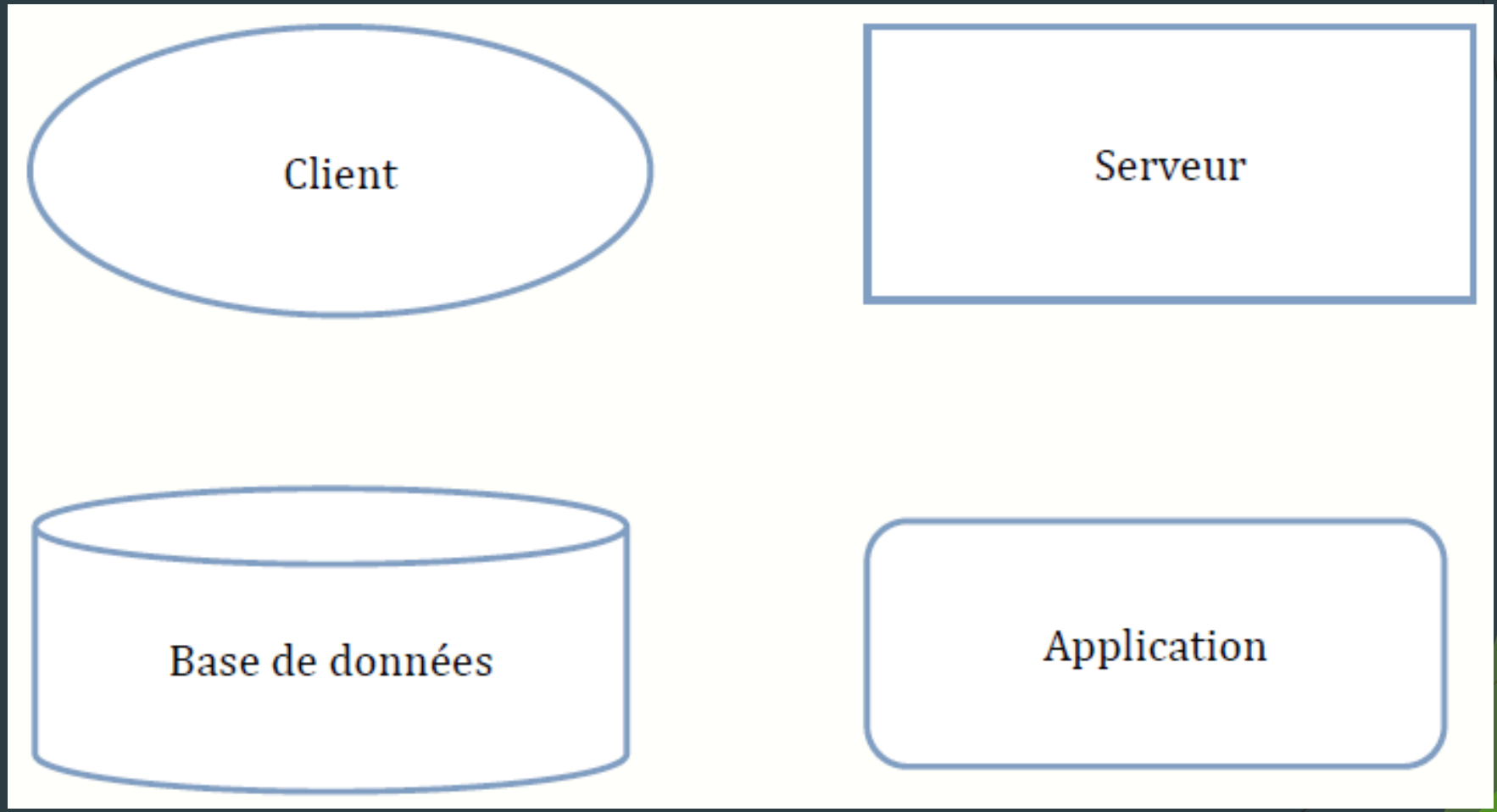
Application

- Une application est un ***module logiciel*** qui a un rôle bien défini dans le système logiciel
- Par exemple, une application d'envoi de mails

Base de données

- Une base de données est un ***entrepôt*** stockant les données sous un format ***normalisé***
- L'interrogation et la modification des données se fait en utilisant un langage de requêtage spécialisé (généralement du ***SQL***)
- La plupart des bases de données obéissent au modèle ***relationnel***
- Un SGBD (Système de Gestion de Base de Données) est un logiciel puissant, généralement accessible sur le ***réseau et*** conçu pour les gros systèmes
- SQL Server, Oracle, MySQL, PostgreSQL sont des exemples de SGBD connus sur le marché

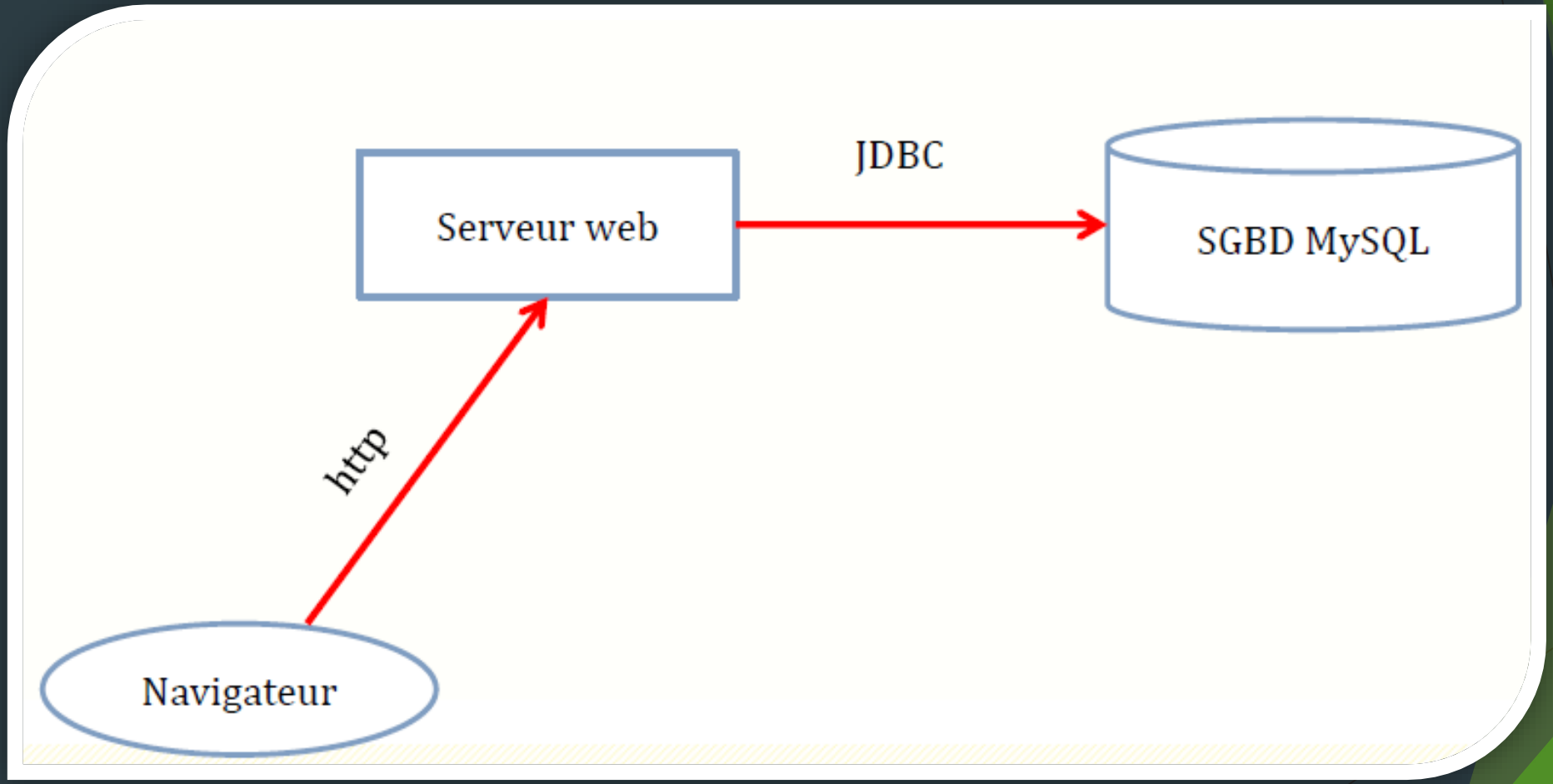
Vue C&C



Connecteurs

- Le connecteur modélise une ***interaction*** entre deux composants
- Un connecteur peut modéliser une interaction ***simple*** (appel de procédure) ou une interaction ***complexe*** (par exemple utilisation d'un protocole comme http)

Exemple

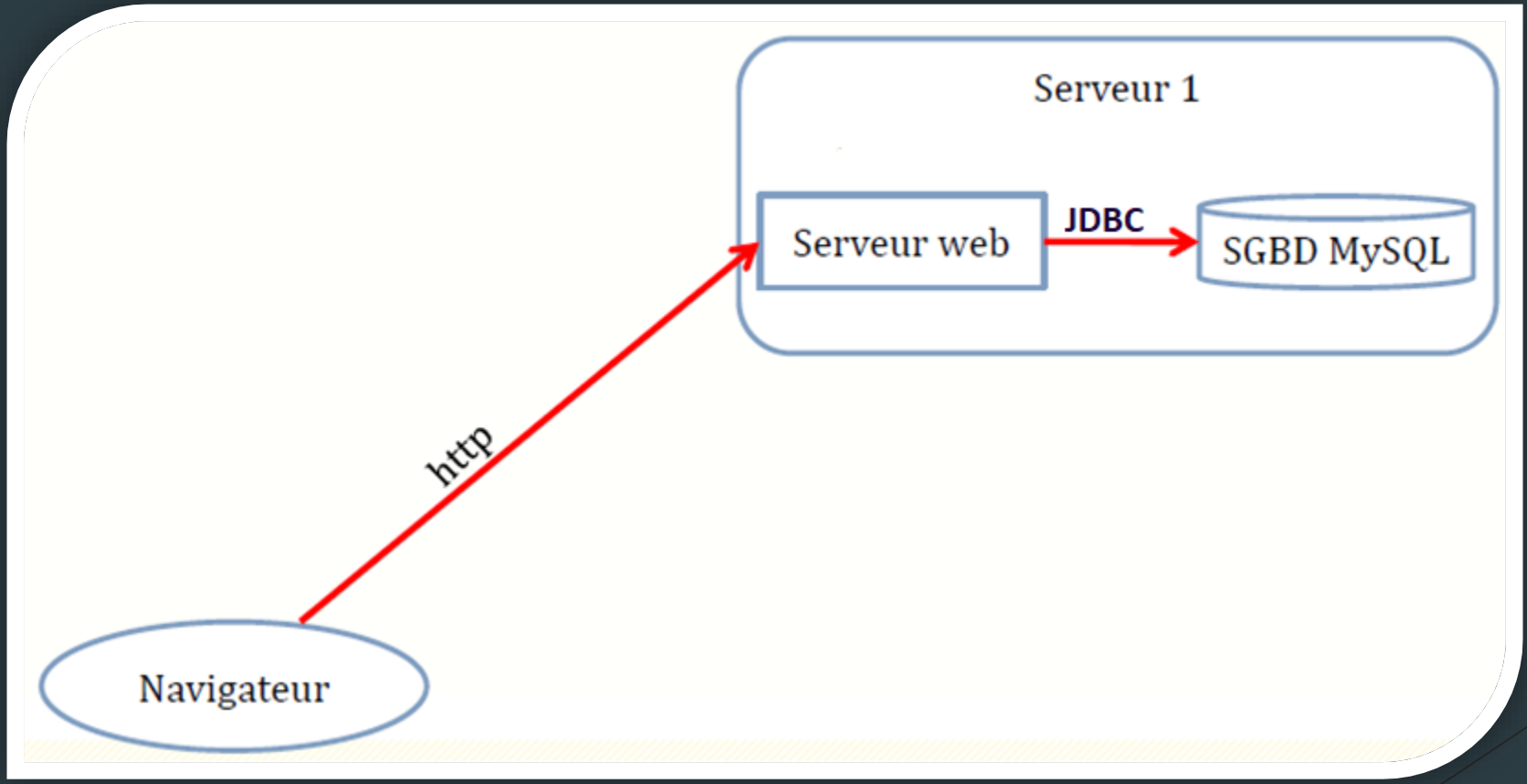


Vue physique et vue logique

- La vue logique d'une architecture logicielle définit les principaux ***composants*** d'une architecture sans se soucier des détails physiques (équipements, machines, etc.)
- La vue physique s'intéresse au ***déploiement physique*** des différents services
- La vue physique est peu précise lors de la conception.
  - Elle devient concrète lors de la phase de déploiement.

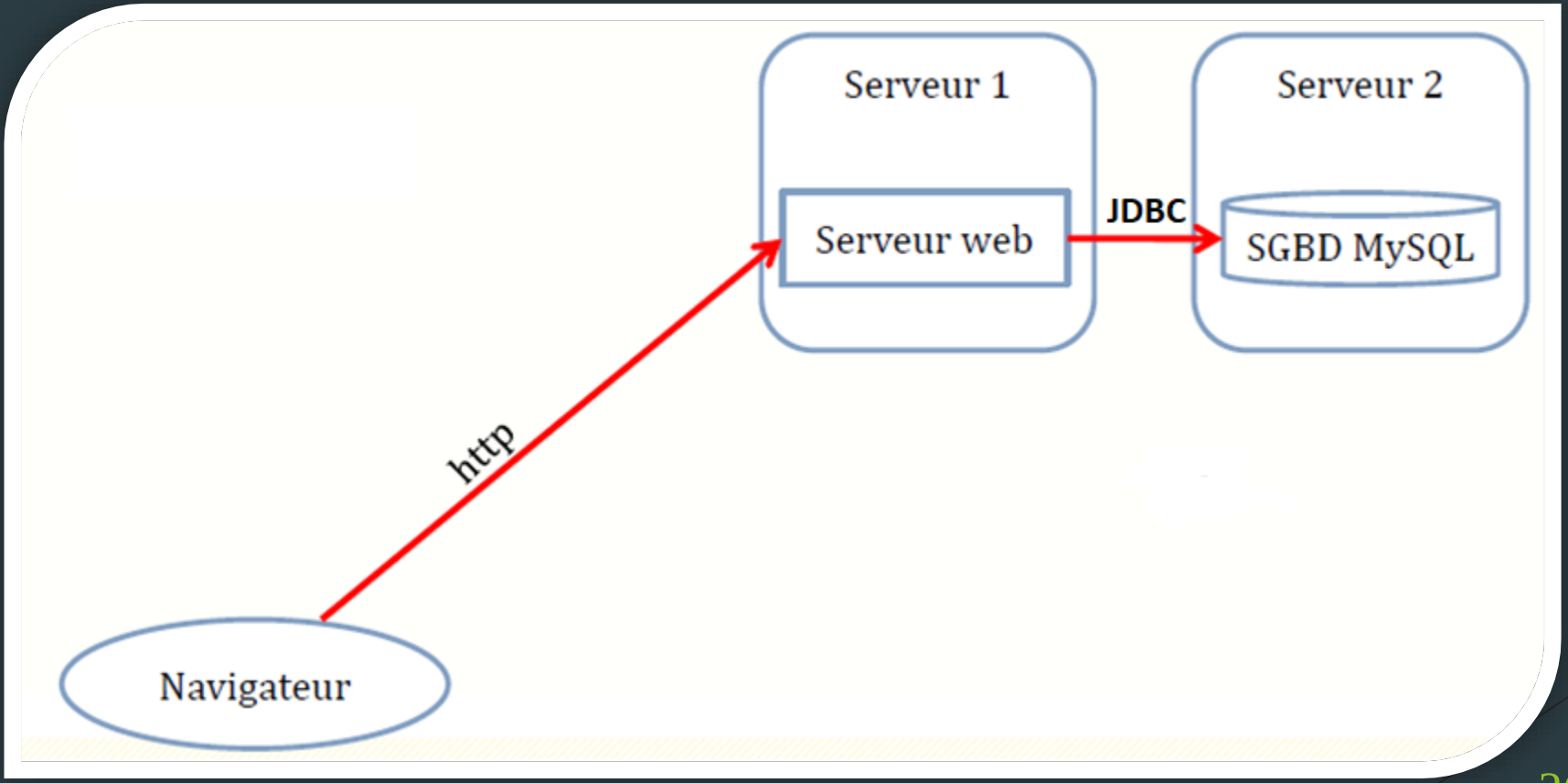
Vue physique et vue logique

- Exemple 1 : services déployés dans le même serveur



Vue physique et vue logique

- Exemple 2 : services déployés dans des serveurs différents



Utilisation de l'architecture

- Donne un aperçu de haut niveau du système qui va faciliter la communication et la compréhension
- Aide à comprendre des systèmes existants
- Décompose le système en sous-systèmes et sous-modules ce qui réduit la complexité et facilite la distribution des tâches
- Facilite l'évolution du système en remplaçant uniquement le sous-système désiré

UML et les architectures logicielles

- Plusieurs formalismes peuvent décrire une architecture logicielle
- UML 2 est un bon moyen de représenter une architecture logicielle
- Le ***diagramme de composants*** peut servir à représenter la ***vue logique*** d'une architecture
- Le ***diagramme de déploiement*** peut servir à représenter la ***vue physique*** d'une architecture

DIAGRAMME DE COMPOSANTS

Composant

- Un composant est une unité autonome dans un système
- Un composant définit un système ou un sous-système de n'importe quelle taille ou complexité
- Les diagrammes de composants permettent de modéliser les composants et leurs interactions
- Les composants d'un système sont facilement réutilisés ou remplacés

Caractéristiques d'un composant

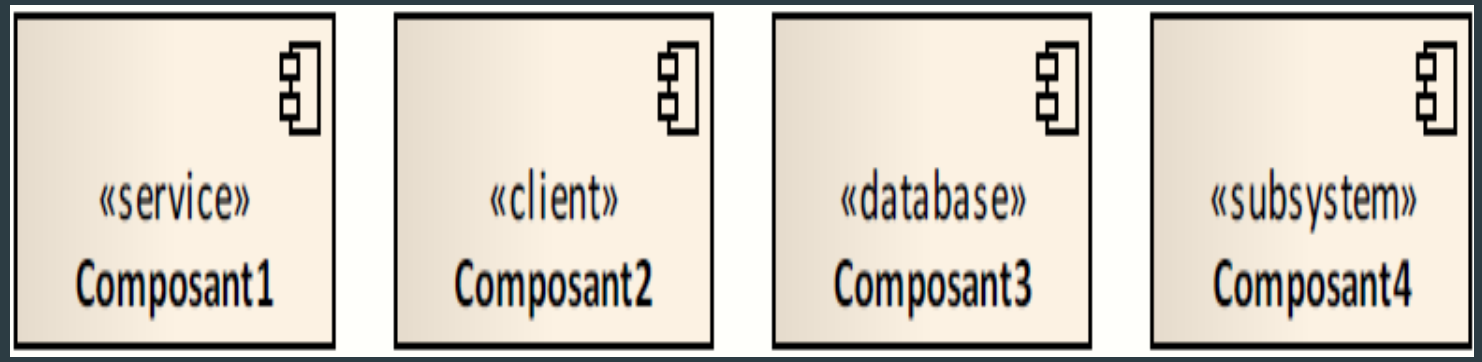
- Un composant est une ***unité modulaire*** avec des interfaces bien définies
- Les interfaces définissent comment le composant peut être ***appelé*** ou ***intégré***
- Le composant est ***remplaçable*** et ***autonome***
- L'implémentation du composant est ***cachée*** (encapsulée) aux entités externes

Représentation UML

Caractéristiques d'un composant

- Un composant a un nom ***unique*** dans son contexte
- Un composant peut être étendu par un ***stéréotype***
- Il existe des stéréotypes standard pour les composants comme « subsytem », « database » ou « executable »
- L'utilisateur peut ajouter ***ses propres stéréotypes*** à condition que ça soit consistant avec l'objectif du diagramme
- Dans le cadre d'une architecture logicielle, ces stéréotypes peuvent être utilisés : « service », « client », etc.

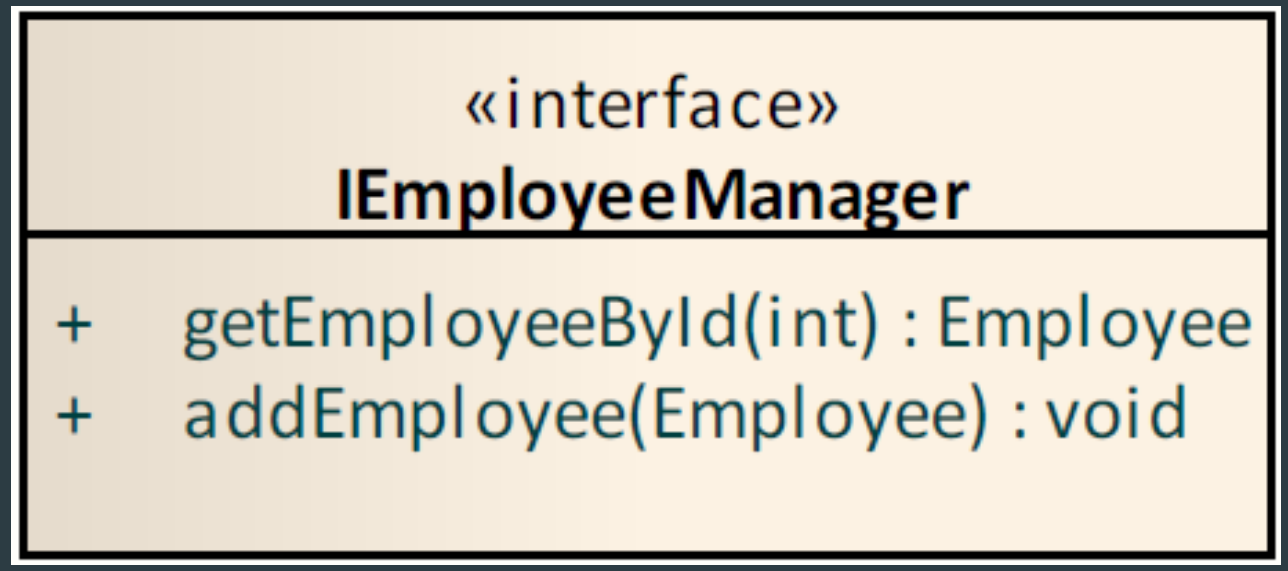
Exemples de stéréotypes



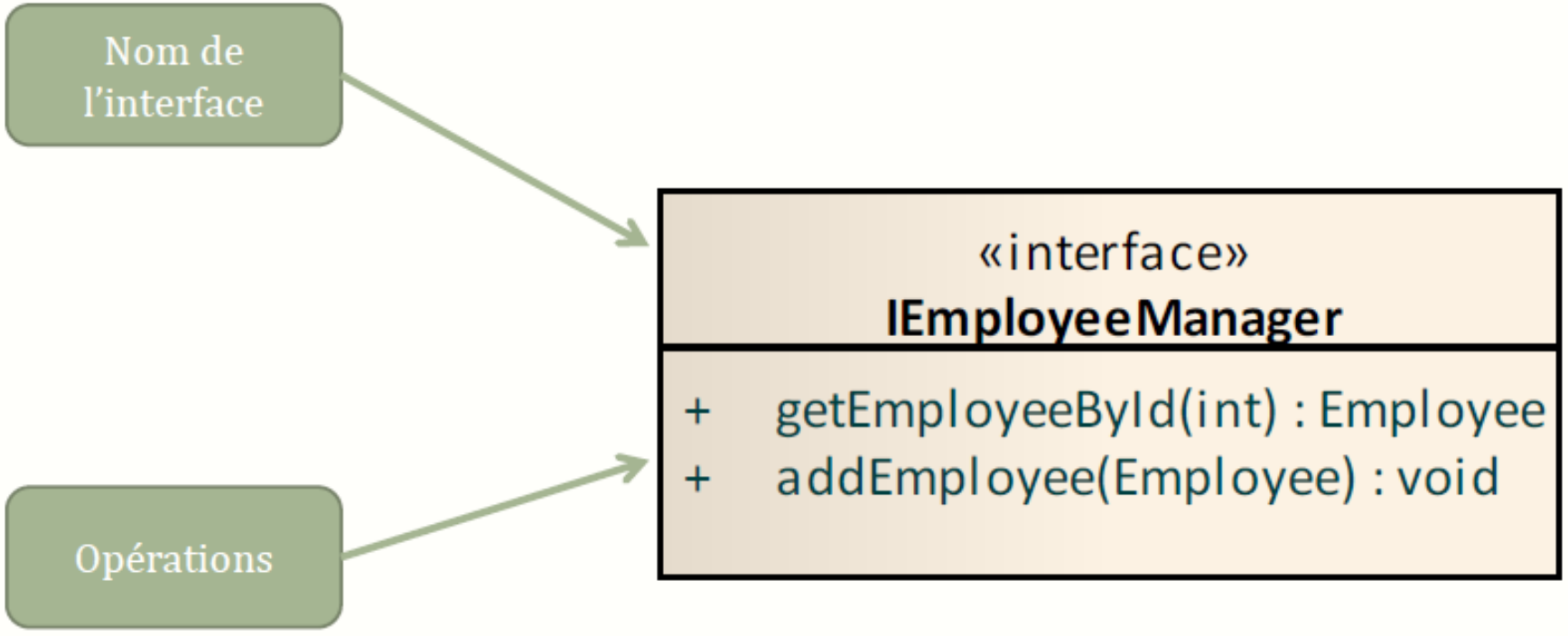
Interfaces

- Un composant définit son comportement en terme d'***interfaces fournies*** et ***interfaces requises***
- Une interface est une ***collection d'opérations*** ayant un ***lien sémantique*** et qui ***n'ont pas d'implémentation***
- L'implémentation des interfaces se fait par une ou plusieurs ***classes*** implémentant le composant
- Les noms d'interface commencent par « I » (convention)
- Un ***contrat*** entre C1 et C2 est défini quand C1 fournit une interface I qui est requise par C2

Exemple d'interface

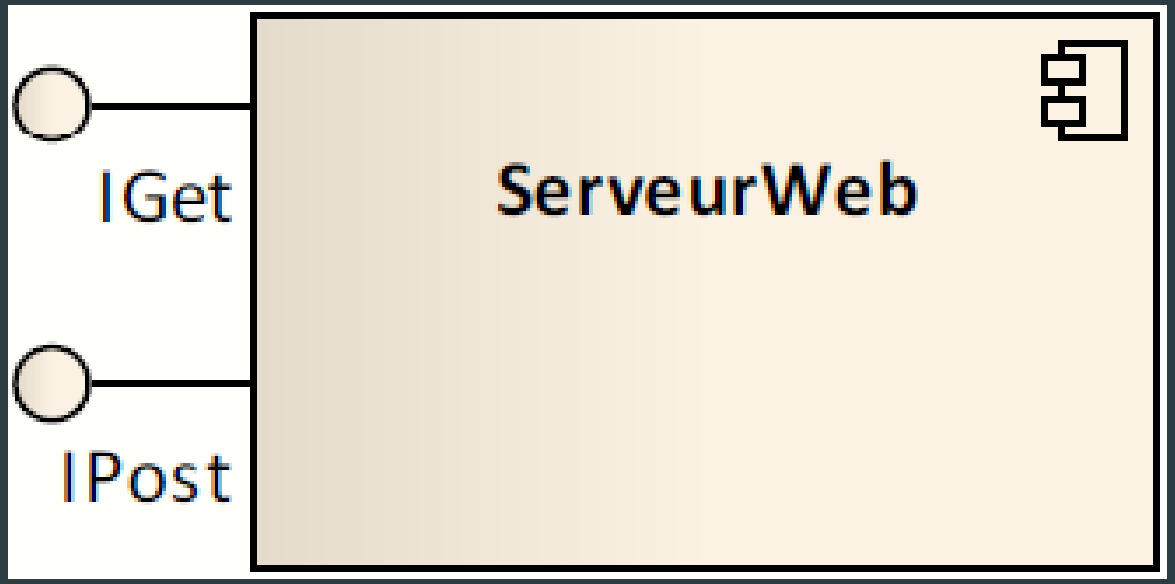


Exemple d'interface



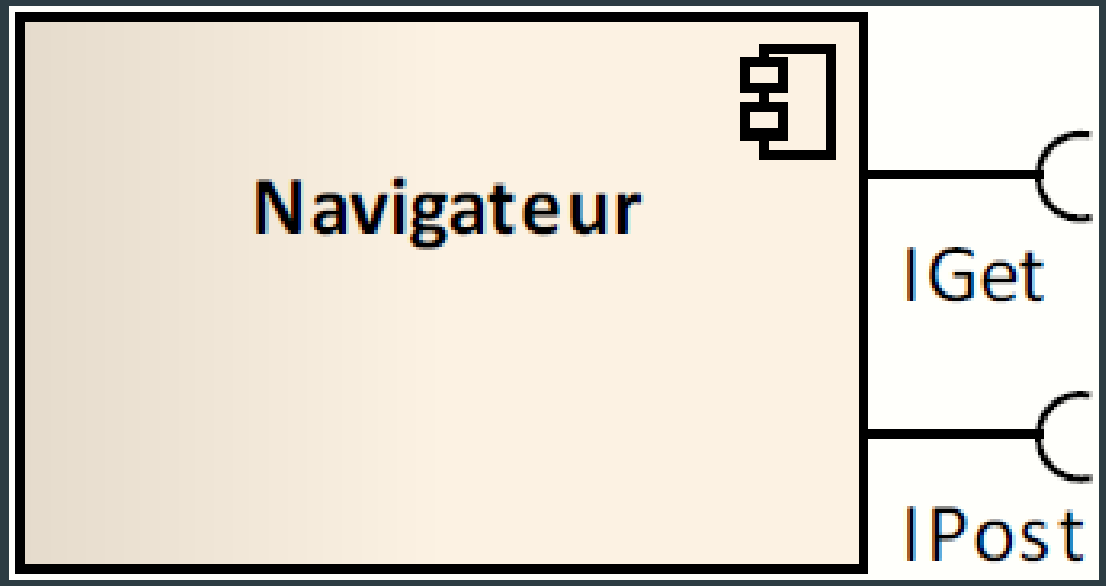
Interfaces fournies

- Une interface fournie définit les fonctions qu'un composant ***pourrait faire***
- Exemple : un serveur web peut gérer les requêtes HTTP de type Get ou Post



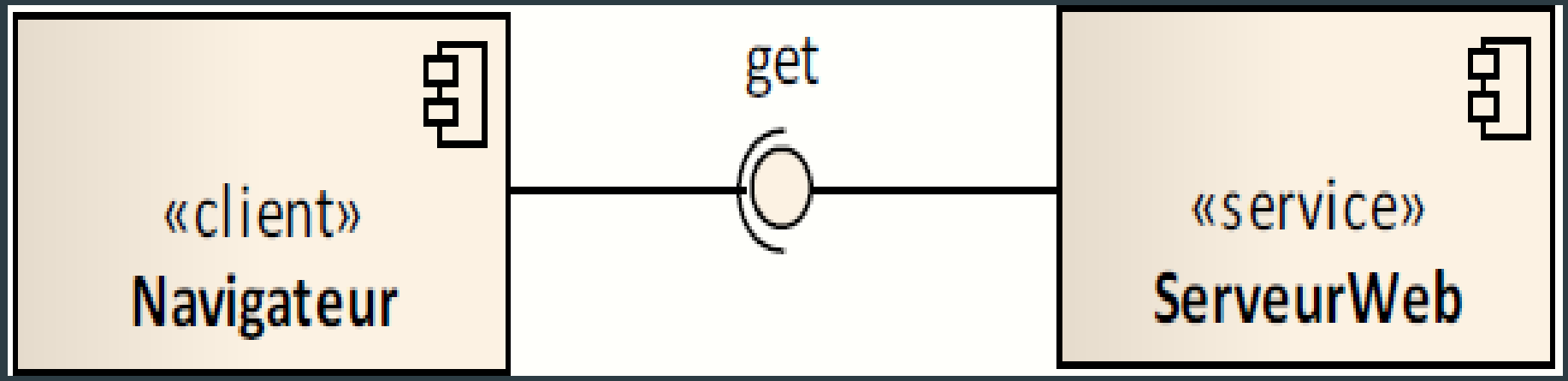
Interfaces requises

- Définit la (ou les interfaces) qu'un composant ***attend*** de son environnement



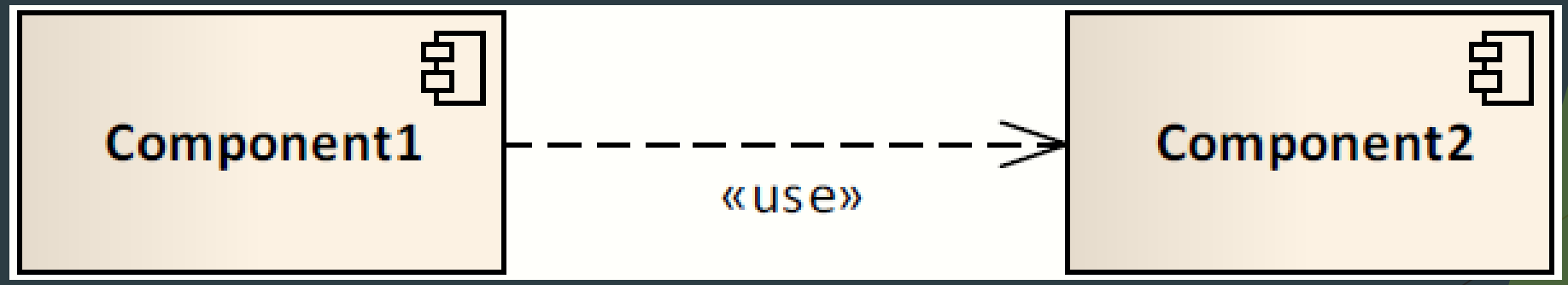
Assemblage

- Un assemblage entre deux composants est lorsqu'une même interface est requise pour un composant et fournie par l'autre



Utilisation

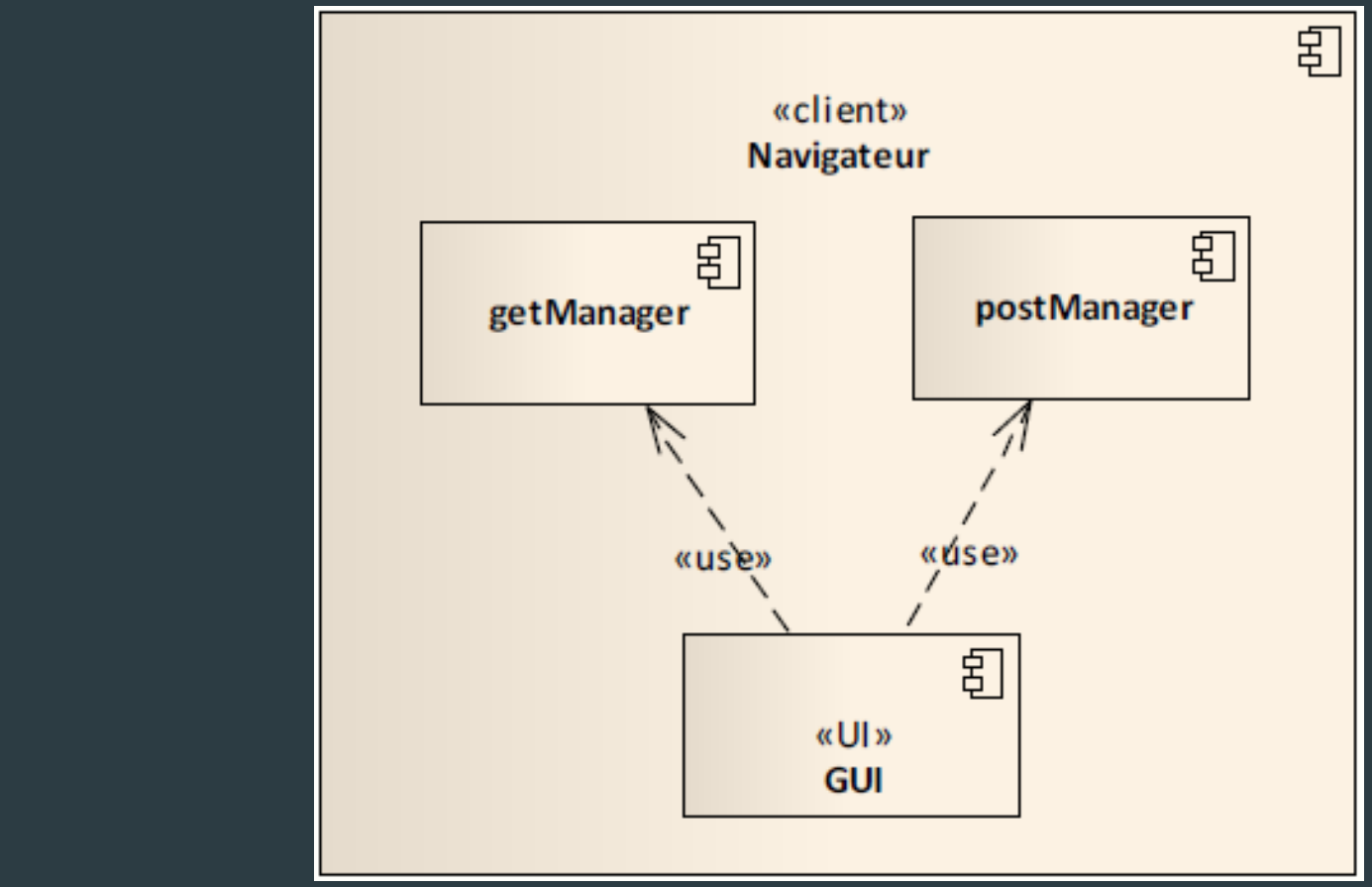
- Un composant C1 dépend d'un autre composant C2 lorsque C1 requiert C2 pour son implémentation (C1 appelle un des services de C2)
- En d'autres mots, l'exécution de C1 requiert la ***présence*** de C2



Composition

- Un composant peut être lui-même composé d'autres composants. On parle alors de composition.
- Par exemple, un navigateur pourrait être composé de :
  - getManager (gestionnaire des requêtes get)
  - postManager (gestionnaire des requêtes POST)
  - GUI (interface)

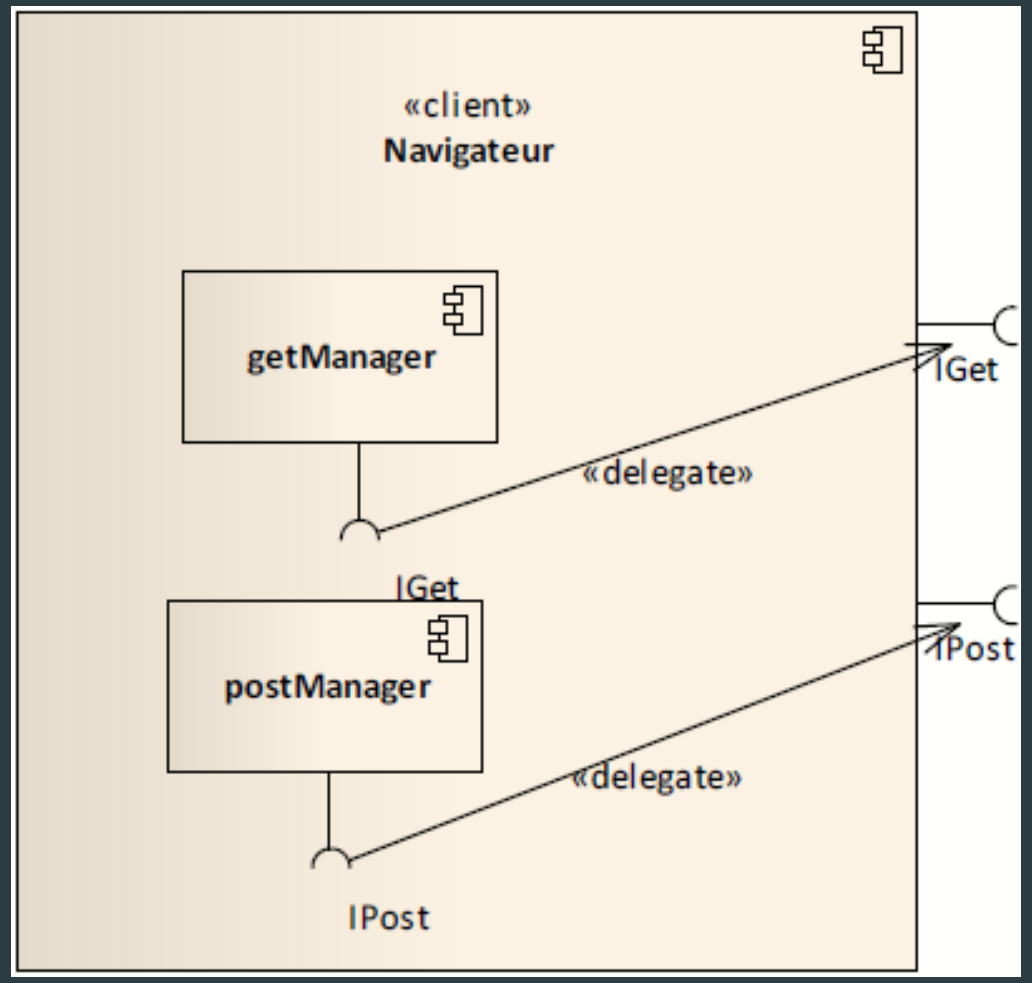
Composition



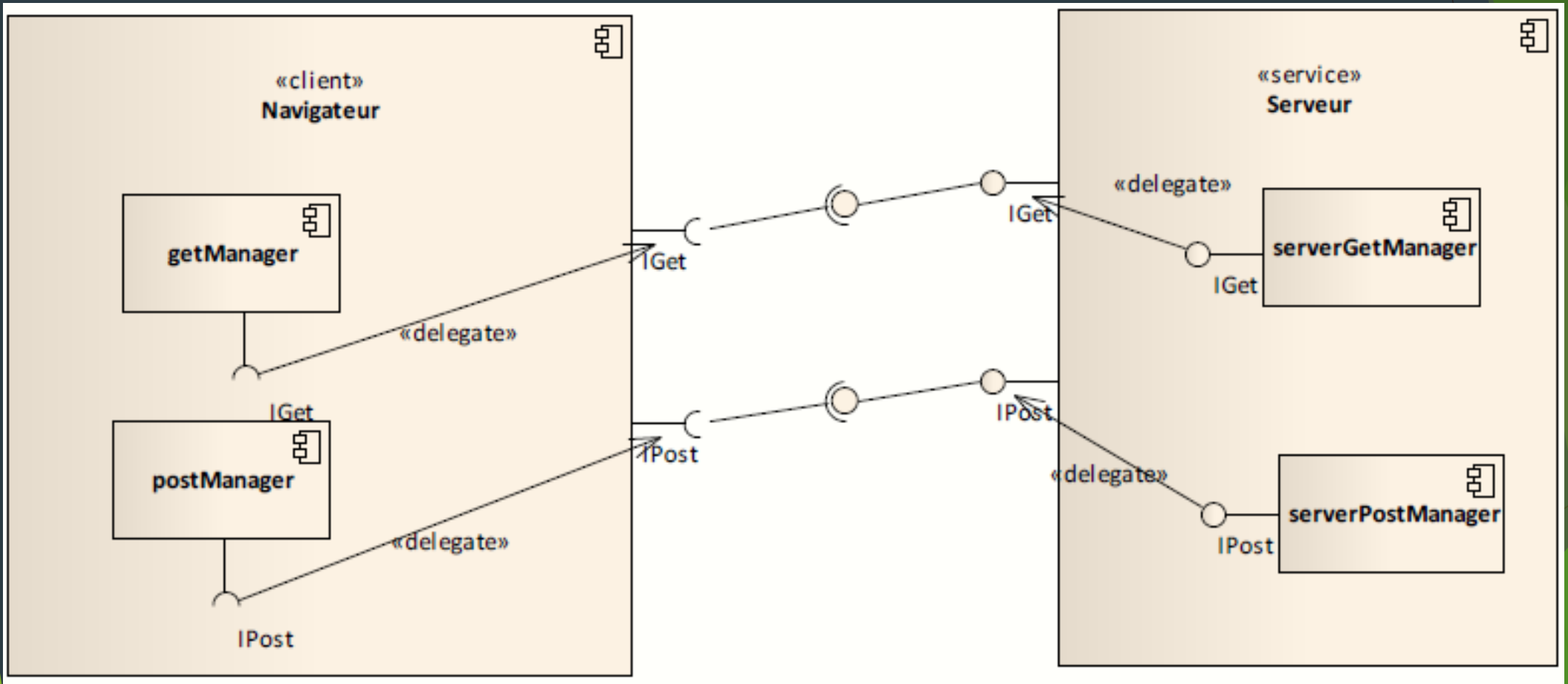
Délégation

- Un composant peut avoir des sous-composants qui incluent des interfaces fournies ou des interfaces requises
- La délégation consiste à ***transférer*** les interfaces fournies / requises du composant interne vers le composant externe

Délégation - Exemple



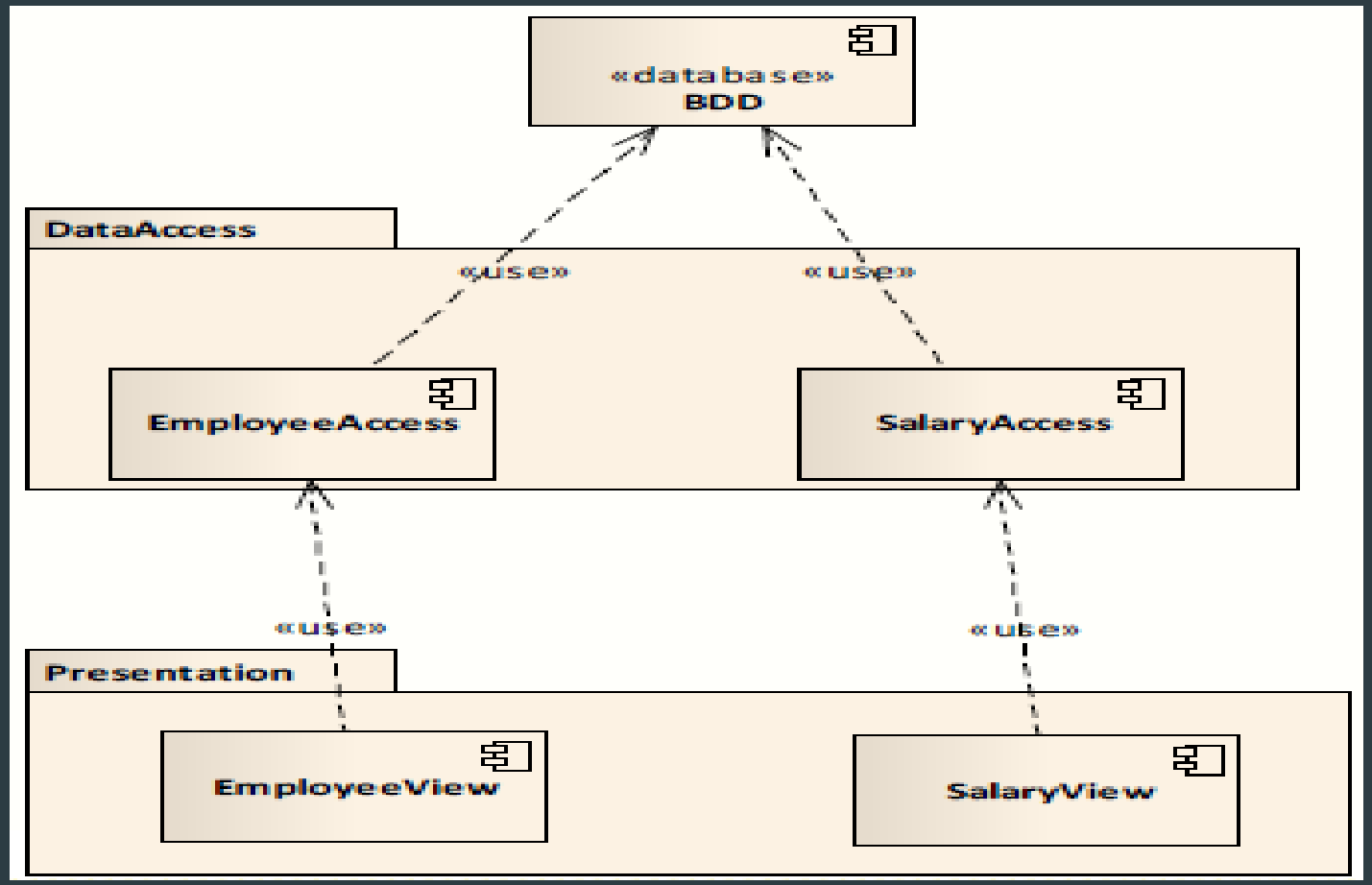
Délégation – Exemple 2



Paquets

- Les paquets peuvent être aussi utilisés dans les diagrammes de composants pour organiser les composants

Paquets


«database»
BDD
DataAccess
«use»
«use»
EmployeeAccess
SalaryAccess
«use»
«use»
Presentation
EmployeeView
SalaryView

Composants et classes

- Les classes sont « packagées » dans des bibliothèques
- Par exemple, une bibliothèque est un fichier jar (java) ou un fichier dll (.NET)
- Le diagramme de composants définit aussi le packaging des classes du système
- Le connecteur liant les classes aux composants est le connecteur « realize »

Composants et classes

- Un artifact est une ***pièce physique*** utilisée par le système
- Un artifact peut être un document, un fichier, un code source ou n'importe quel élément ayant une relation avec le système
- La dépendance avec le stéréotype « manifest » indique qu'un artifact est la représentation physique d'un composant.
- Par exemple, un fichier jar est une représentation physique d'une classe java

ARCHITECTURE DES APPLICATIONS

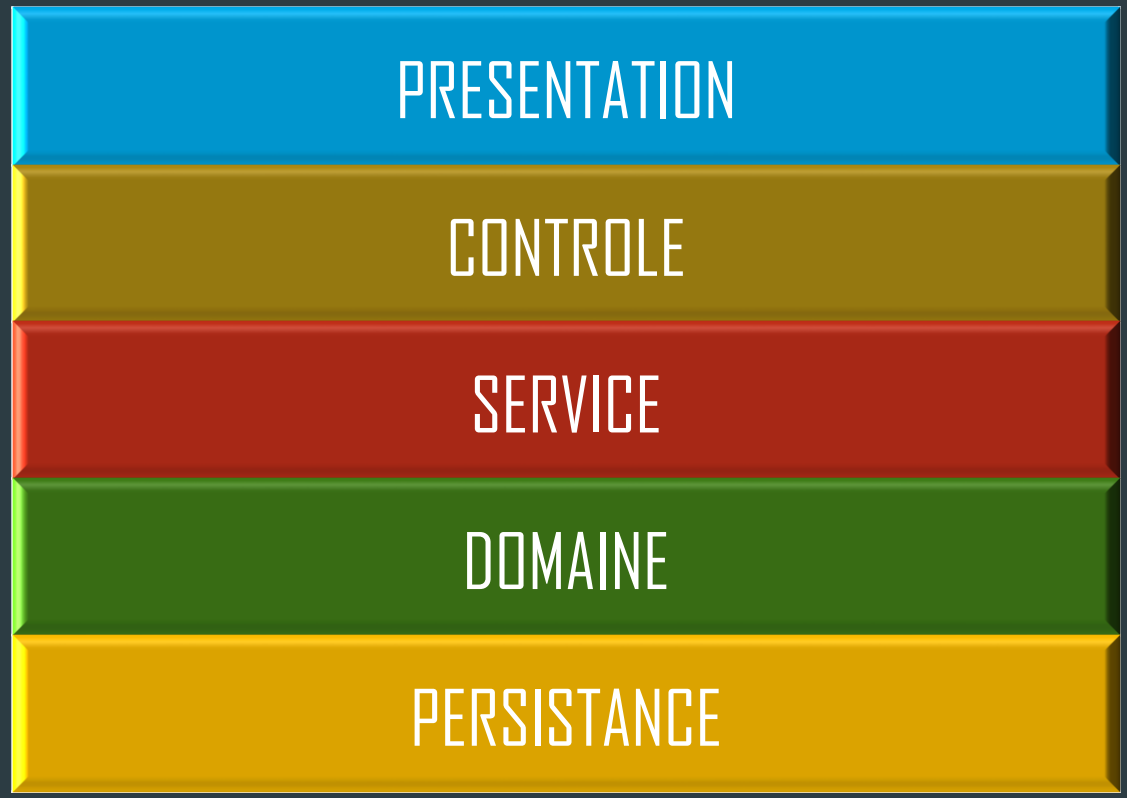
Introduction

- L'architecture d'une application est son découpage en couches ou vues logiques et en niveaux ou vues physiques (les tiers)
- Le découpage logique est effectué hors de tout contexte d'exécution (machines, OS et réseaux)
- Le découpage physique prend en compte le contexte d'exécution

Vue Logique (vue en couches)

- La vue logique montre le découpage des fonctions de l'application :
  - Cette vue est indépendante des considérations physiques
  - Le modèle de référence est le modèle à cinq couches qui s'applique aux applications munies d'une interface graphique manipulant des données persistantes.
- Cette vue permet de maitriser la complexité des applications
  - Séparation des types de développement (développements interface utilisateur, métier et persistance).
- Le passage d'une couche vers une autre se fait impérativement via des interfaces qui représentent chacune un service d'accès.

Modèle en cinq couches



Couche Présentation (1/2)

- Elle correspond à la partie de l'application visible et interactive avec les utilisateurs
  - On parle d'interface homme machine
  - Assure et gère l’interface permettant d’interagir avec l’utilisateur
- Elle peut se présenter sous plusieurs formes :
  - Elle peut être réalisée par une application graphique ou textuelle
  - Elle peut être représentée en HTML pour être exploitée par un navigateur
- Elle peut prendre plusieurs facettes sans changer la finalité de l'application
  - Une même fonctionnalité métier peut prendre différentes formes de présentation

Couche Présentation (2/2)

- On distingue trois catégories d'IHM pour les applications interactives :
  - Client léger : Les différents écrans de l'application sont générés en temps réels côté serveur et téléchargés par le poste client.
  - Client lourd : L'ensemble des écrans de l'application est généré et stocké sur le poste client. Communique avec le programme serveur via du code (application sous Swing par exemple)
  - Client riche : Ce modèle est un compromis entre le léger et le lourd (Exemple : JSF, Adobe Flex, SilverLight) qui permet d'exécuter du code directement dans le navigateur.

Couche Contrôle

- Gestion des requêtes du client
- Gestion des erreurs et exceptions qui peuvent être levées
- Gestion des sessions/espaces de travail utilisateur
- Gestion des droits d’accès

Couche Service

- Correspond à la partie fonctionnelle de l'application
  - Décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs (via la couche Présentation)
- Elle représente l’implémentation de la logique des cas d’utilisation
- Cette couche se charge donc de :
  - la logique métier
  - la sécurité applicative
  - les appels aux objets métiers de la couche Domaine.

Couche Domaine

- Recense les objets métiers manipulés par l'application (dans la couche service).

Couche Persistance

- Elle consiste en la partie gérant l'accès aux données du système
- Elle assure :
  - L’enregistrement sur support physique des données de l'application
    - Fichiers (binaires, XML, etc.)
    - Base de données (simple, avec redondance, multiple (fédération de BDD), …)
  - La création, la modification, la suppression d’occurrences des objets métiers.
  - Le mapping objet/relationnel et inversement.

Interaction entre les couches

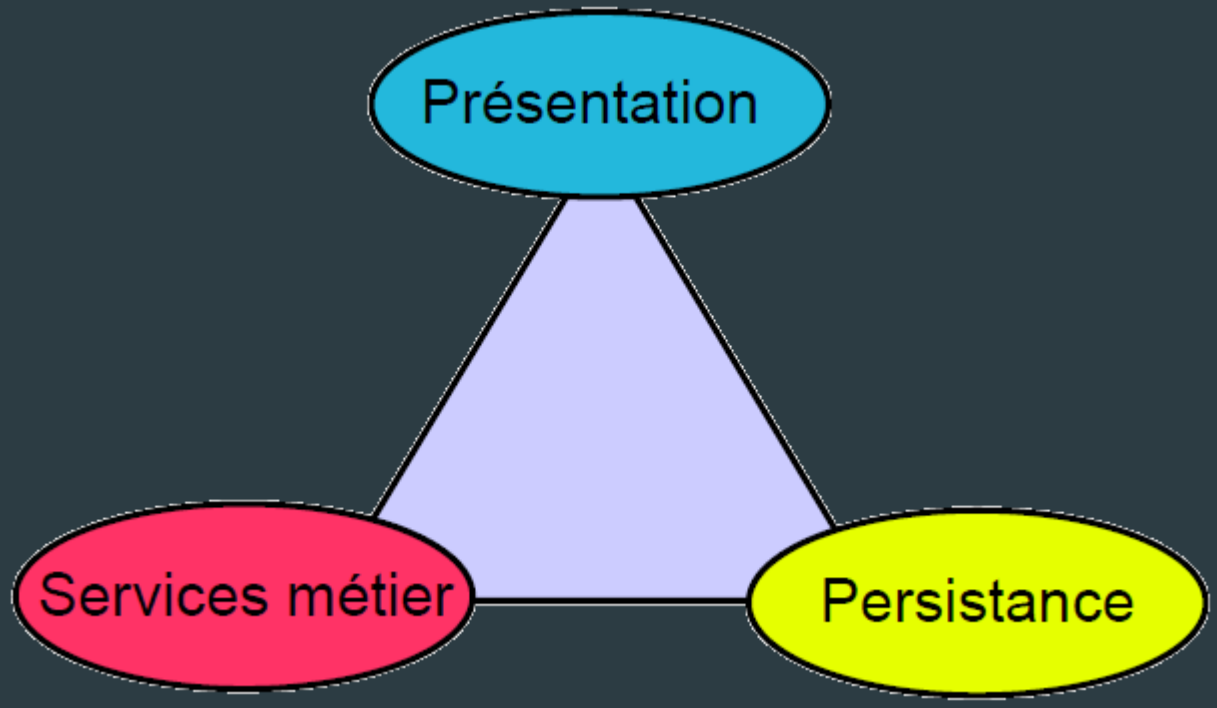
- Chaque couche a ses propres fonctions et utilise celle située en dessous d'elle.
  - Les couches sont intégrées et coopèrent pour le fonctionnement global de l'application
- Un couplage faible doit exister entre les couches
  - Le changement interne d'une couche ne devrait pas affecter les autres
    - Le changement du type de stockage (fichiers plats vers Base de données) ne devrait affecter que la couche de Persistance
  - Facilite la réutilisabilité et la maintenance

Vue Physique (tiers view)

- La vue physique (ou vue en niveaux) donne une vision plus physique de la structuration.
- Les couches peuvent être réparties physiquement sur différents composants matériels
  - On parle dans ce cas de niveaux (ou tiers)
  - De nombreuses variantes de placement de ces tiers existent
- En fonction de cette répartition, on peut obtenir :
  - Un modèle centralisé (architecture 1-tiers)
  - Une architecture non centralisée (architecture 2-tiers, 3-tiers, etc.)

Vue Physique (tiers view)

- Dans les architectures présentées dans la suite, nous supposerons que l'application dispose au moins de trois couches de base



Architecture 1-tiers

- Il est aussi appelé « Architecture monolithique » ou « Modèle 1-tiers »
  - Toutes les couches applicatives se trouvent dans un seul composant matériel
- Les applications utilisant une telle architecture
  - Sont des binaires sur lequel s'exécutent toutes les couches (de la présentation à la persistance).
  - C'est l'exemple des applications utilisées en monoposte (Word par exemple)

Architecture 1-tiers

Poste de travail ou ordinateur central

Architecture 2-tiers

- Il est aussi appelé « Architecture client/serveur »
- Le système est composé de deux composants principaux se trouvant généralement dans des machines séparées : le ***client*** et le ***serveur***
  - Le client envoie des ***requêtes*** au serveur
  - Le serveur réagit aux requêtes en renvoyant des ***réponses***
- Client / serveur de base, avec 2 éléments
  - Client : présentation, interface utilisateur
  - Serveur : partie persistance, gestion physique des données

Architecture 2-tiers : variantes (1/6)

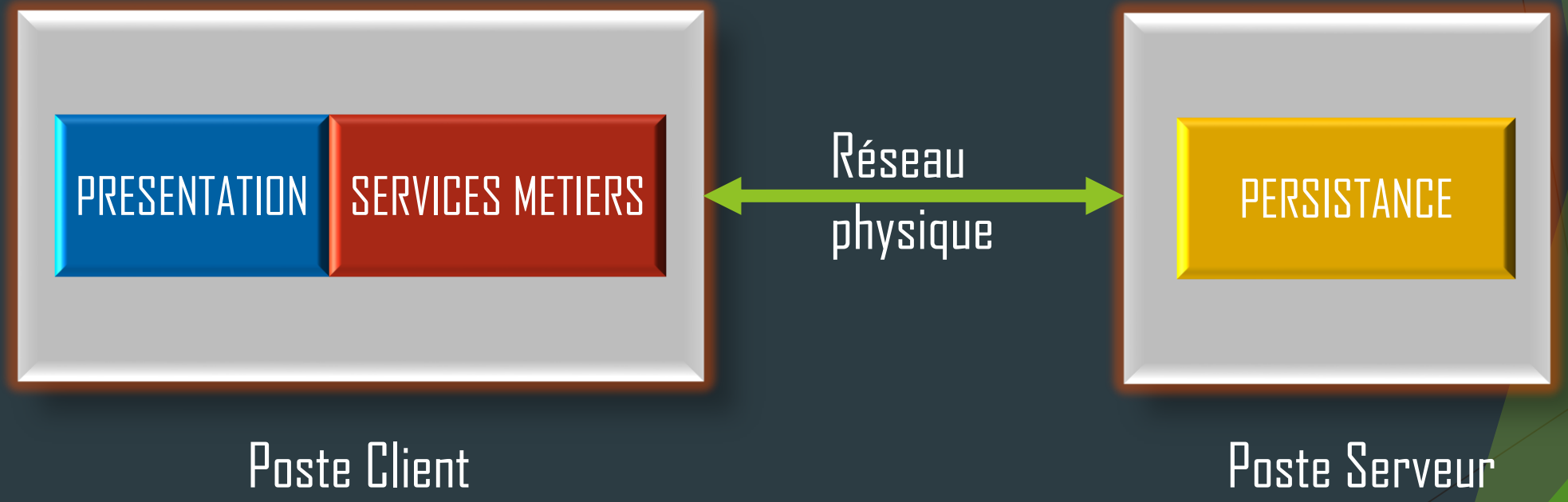
- Les services métier (la partie applicative) peuvent être :
- Soit entièrement coté client, intégrés avec la présentation
  - La partie serveur ne gère que les données
    - Ex : traitement de texte avec serveur de fichiers distants
    - Ex : application accédant à une BDD distante

Architecture 2-tiers : variantes (2/6)

- Les services métier (la partie applicative) peuvent être :
- 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 un 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

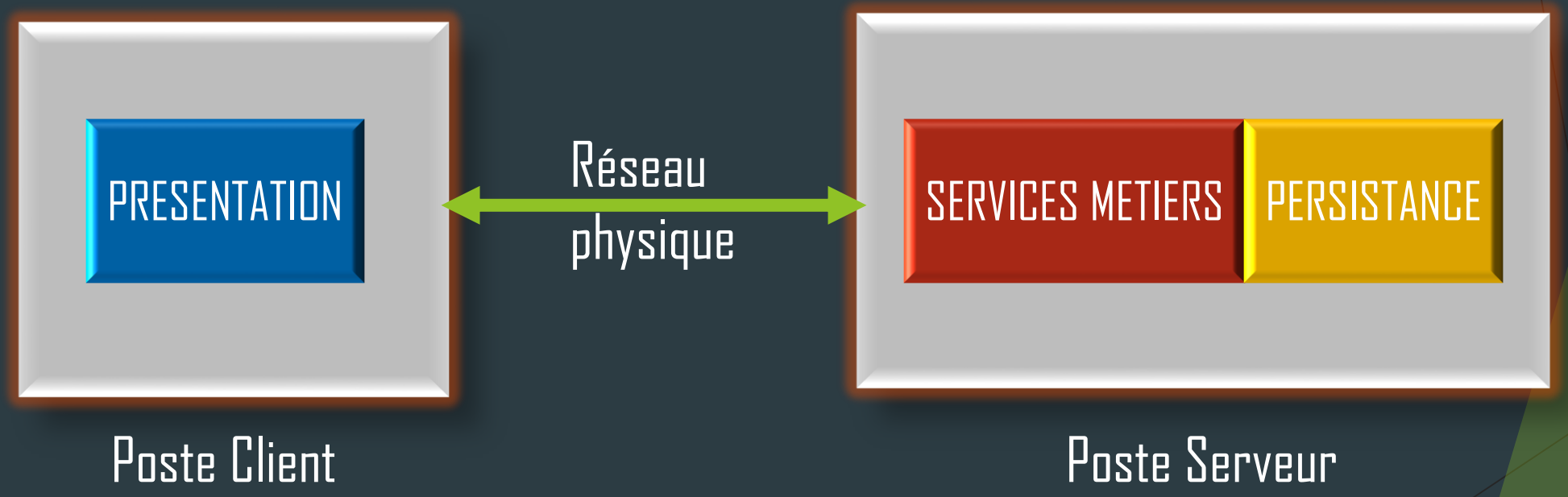
Architecture 2-tiers : variantes (3/6)

- Client : présentation + applicatif (client lourd)



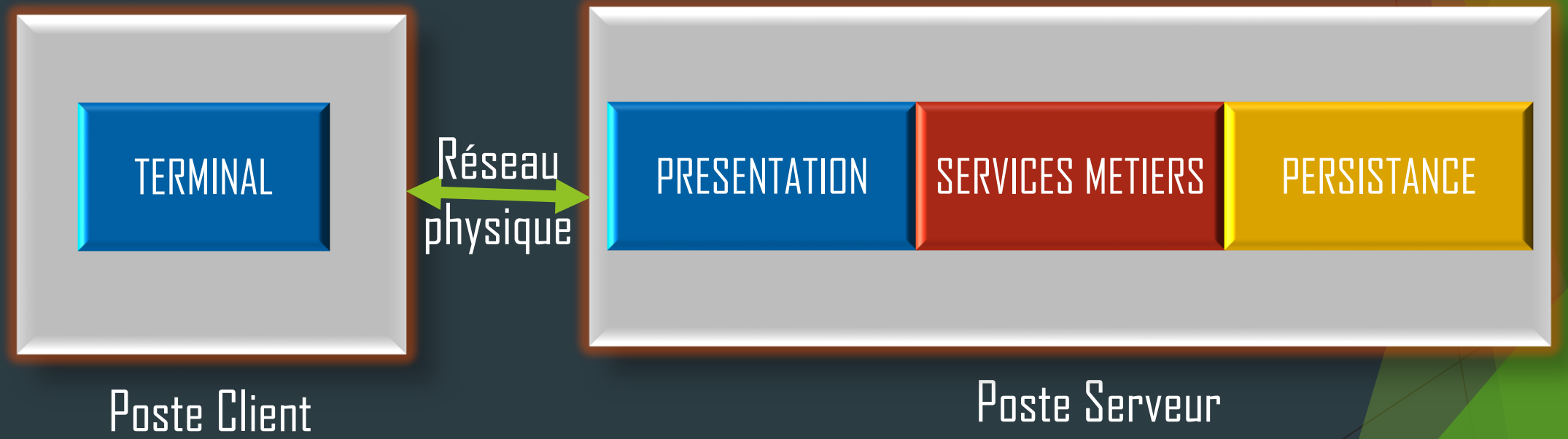
Architecture 2-tiers : variantes (4/6)

- Serveur : applicatif + gestion des données (client léger)



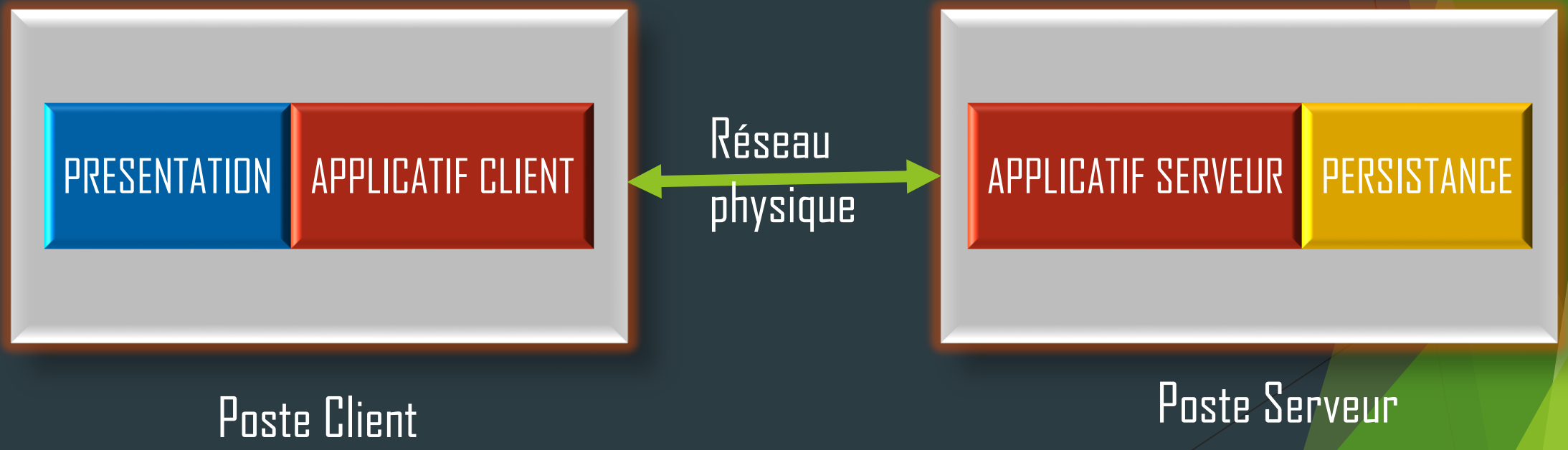
Architecture 2-tiers : variantes (5/6)

- Terminal : client intègre un minimum de la partie présentation (client léger)



Architecture 2-tiers : variantes (6/6)

- Applicatif découpé entre client et serveur (client riche)
- Exemple: Gmail, Google Maps



Architecture 2-tiers : avantages et inconvénients

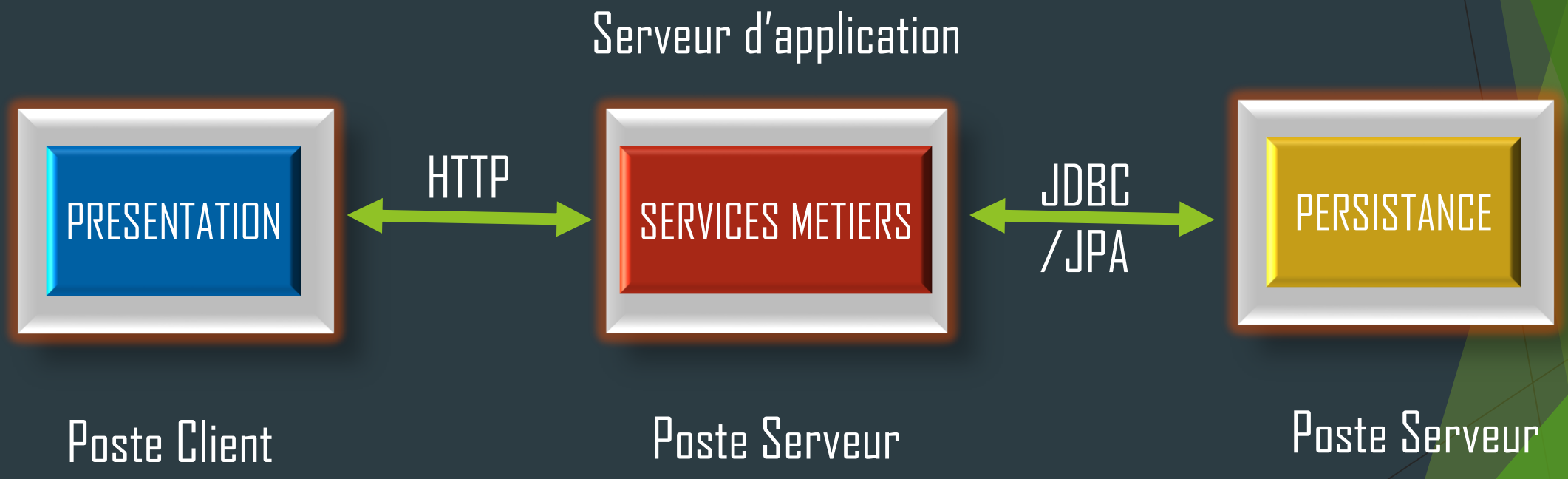
- ***AVANTAGES***
  - Séparation des tâches
  - Simple à utiliser
- ***INCONVÉNIENTS***
  - Souvent insuffisant pour des cas complexes

Architecture 3-tiers

- Les 3 principaux tiers s'exécutent chacun sur une machine différente
  - Très développée de nos jours dans le contexte du web
- Couche présentation
  - Affichage de contenu HTML par le navigateur (client léger)
- Couche applicative / métier
  - Serveur d'applications
    - Serveur web + conteneur web (JSP, servlets, JSF, etc.) ➔ génération contenu dynamique
- Couche persistance
  - Serveur(s) de base de données

Architecture 3-tiers

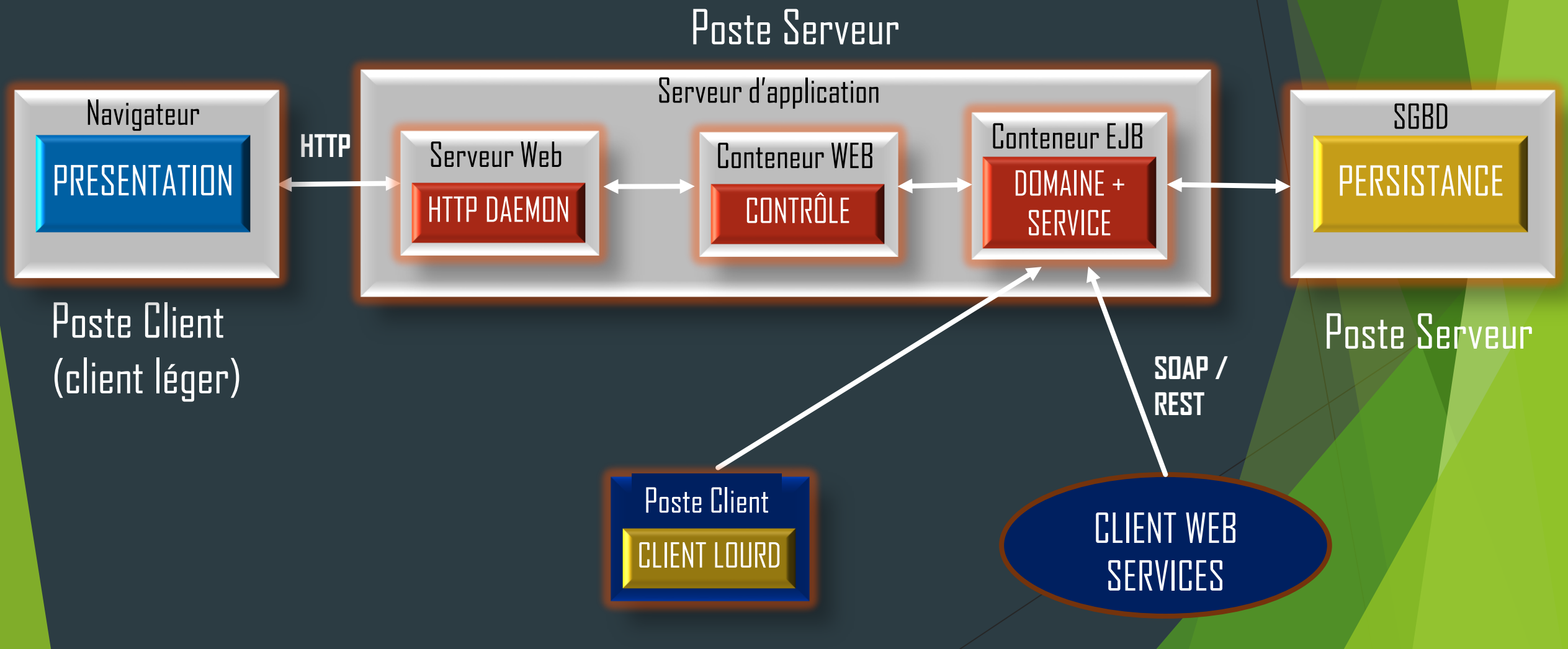
- Les 3 principaux tiers s'exécutent chacun sur une machine différente



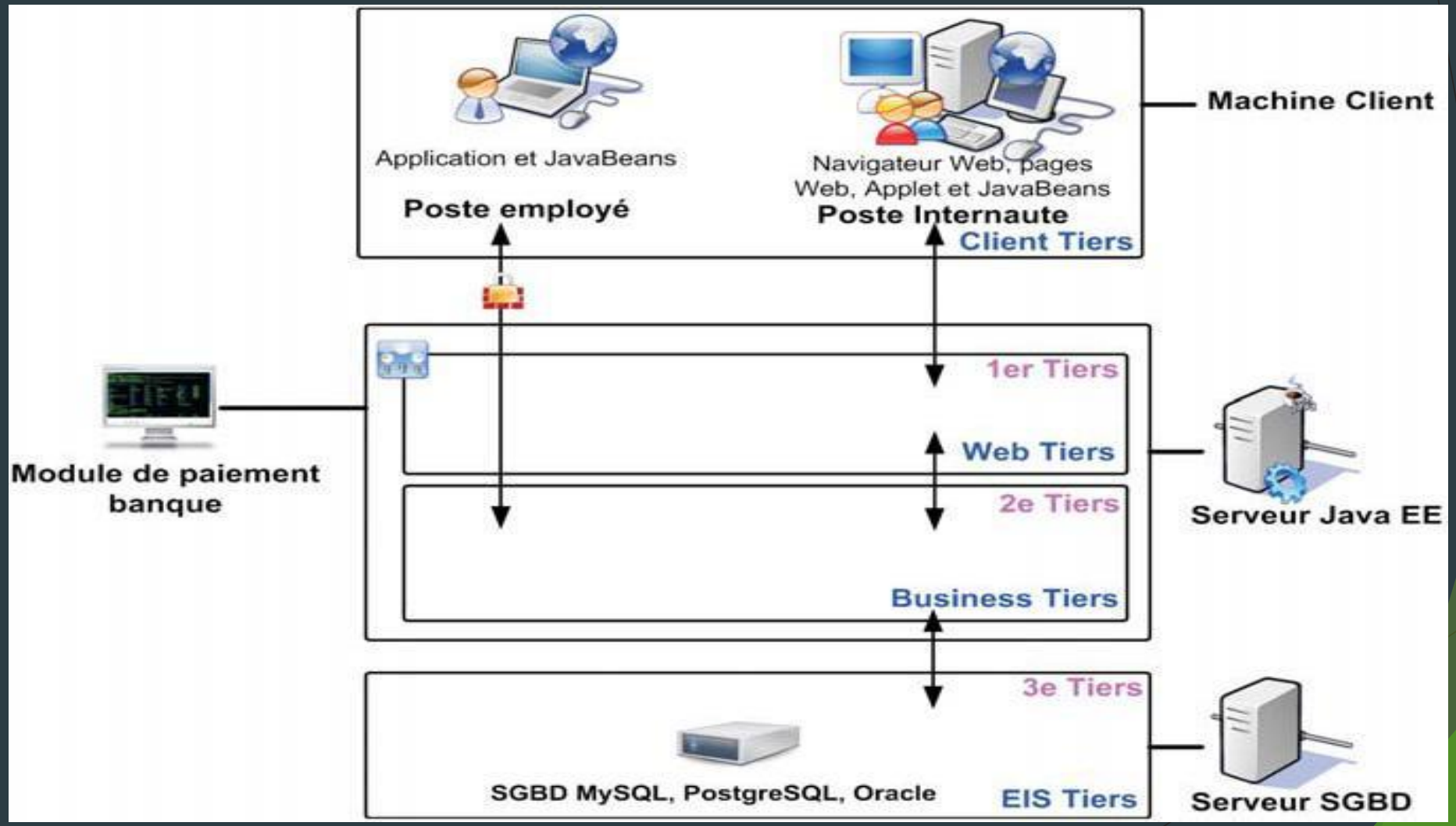
Architecture N-tiers (ou multi-tier)

- Rajoute des couches en plus à l'architecture 3-tiers
- 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, etc.
- Chaque service correspond à une couche, d'où le terme de N-tiers

Architecture N-tiers (plateforme Java EE)



Architecture N-tiers (plateforme Java EE)



Architecture 3/N-tiers : avantages

- Intérêts d'avoir plusieurs couches (3 ou plus)
  - Réutilisation de services existants
  - Découplage des aspects métier et technique et des services entre eux : meilleure modularité
  - Facilite l'évolution : nouvelle version de service
  - Facilite le 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

Architecture N-tiers : inconvénients

- ***INCONVÉNIENTS***
- En général, les divers services s'appuient sur des technologies très variées : nécessité 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
- Nécessite des ressources matérielles importantes

QUELQUES DESIGN PATTERNS

Le modèle MVC

- MVC est un motif d'architecture logicielle (design pattern) largement répandu
  - Il est principalement destiné aux applications disposant d'une GUI
  - Il a été lancé en 1978 et est très populaire pour les applications web.
- Il repose sur la volonté de séparer les données, les traitements et la présentation.
  - Découplage de l'affichage des informations, des actions de l'utilisateur et de l'accès aux données.
- Plus récemment, il a été recommandé comme modèle pour la plate-forme J2EE de Sun et gagne fortement en popularité auprès des développeurs, quelque soit le langage utilisé.

Le modèle MVC

- On peut trouver du MVC dans n'importe quelle architecture
  - Il concerne l'organisation du code
- C'est un motif qui est utilisé par de nombreux Framework pour applications web
  - Ruby on Rails
  - ASP.NET MVC,
  - Symfony
  - Laravel
  - AngularJs
  - Etc.

Le modèle MVC

- Le motif est composé de trois composants ayant chacun une responsabilité bien définie :
  - les modèles
  - les vues
  - les contrôleurs

Le modèle MVC

- **LE MODELE**
  - Représente la partie de l'application qui exécute la logique métier.
  - Il est responsable de la représentation et de la manipulation des données
    - Définition des données de l'application
    - Définition des fonctions de manipulation des données
    - Stockage et extraction des données de la BDD
  - Il effectue ses opérations sur ordre du contrôleur

Le modèle MVC

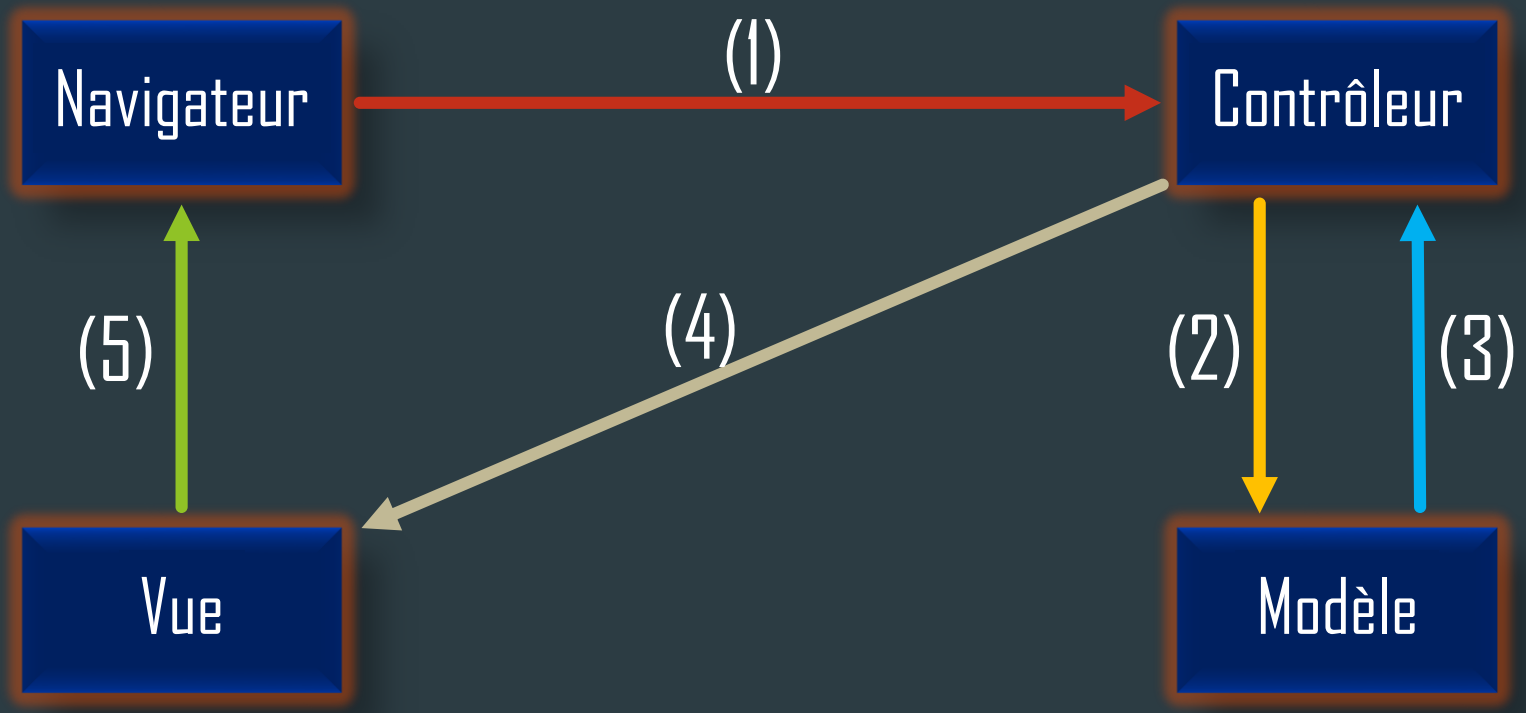
- **LA VUE**
  - Représente l'interface utilisateur.
  - Elle n'effectue aucun traitement
    - Elle notifie le contrôleur des actions de l’utilisateur
    - elle affiche les données que lui fournit le modèle.
  - Il peut y avoir plusieurs vues qui présentent les données d'un même modèle
    - Ex : Sous Windows, affichage des fichiers en mode liste, icone, détail, etc.

Le modèle MVC

- **LE CONTROLEUR**
  - Il gère la dynamique de l'application
    - fait le lien entre la vue et le modèle
  - Le contrôleur se charge :
    - d'intercepter les requêtes de l'utilisateur,
    - d'appeler le modèle,
    - puis de rediriger vers la vue adéquate.
  - Il ne doit faire aucun traitement
    - Il ne fait que de l'interception et de la redirection.

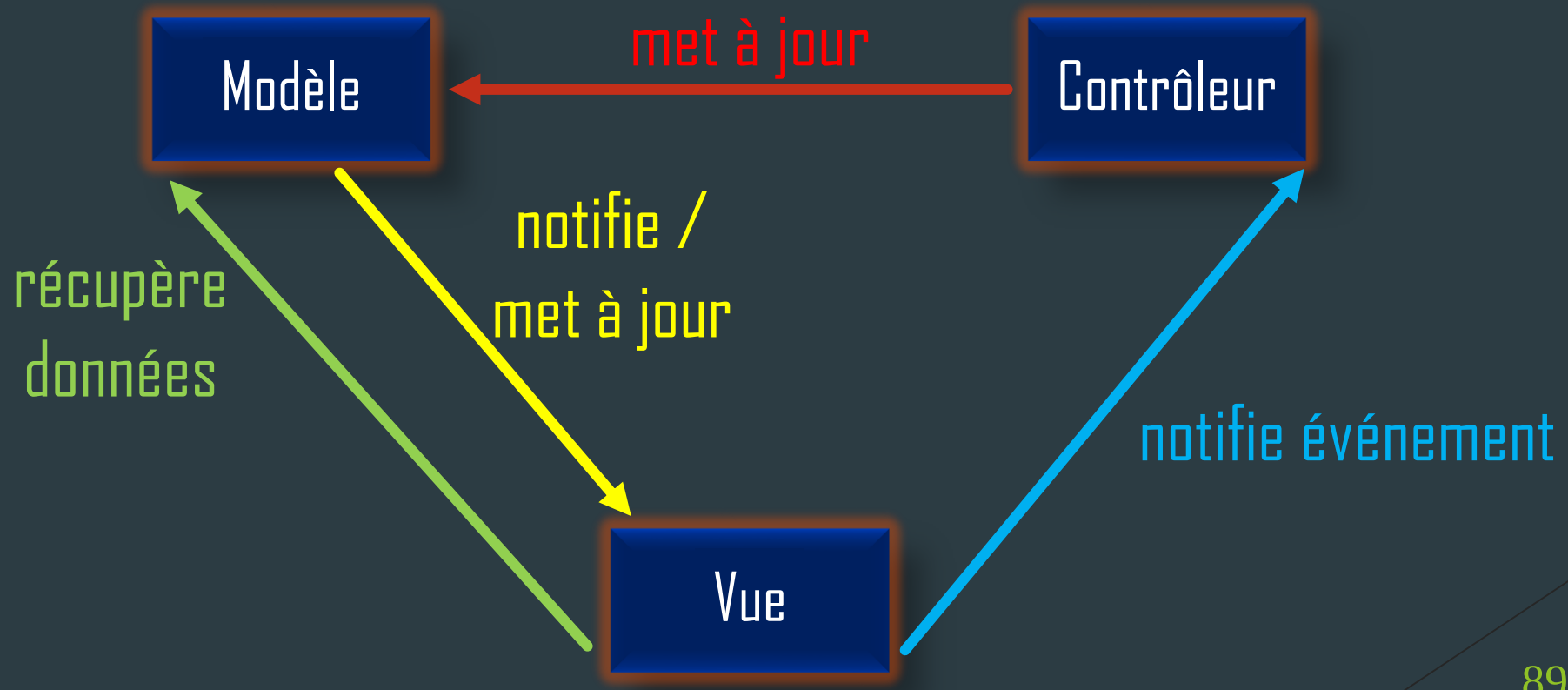
Le modèle MVC

- Dans le contexte du web



Le modèle MVC

- Dans le contexte d'une application GUI



Le modèle MVC

- **AVANTAGES**
  - Séparation des compétences (design, base de données, application)
  - Simplicité de mise à jour
  - Testabilité accrue (les tests unitaires se font plus facilement)
- **INCONVENIENTS**
  - Plus de ressources consommées.
  - Développement initial plus long
  - Plus d'efforts de développement car chaque tâche concerne les trois couches

<code>DEMO</code>

Le Pattern DAO

Le pattern DAO

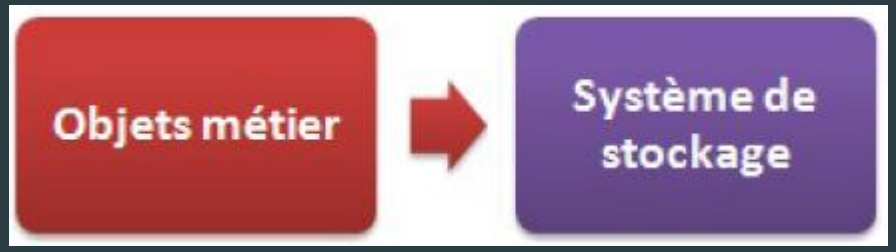
- DAO : Data Access Object / DAL : Data Access Layer
  - Constitué d'un ensemble de classes offrant des services d'accès au système de stockage
- Permet de découpler la couche Métier et la couche de stockage des données
  - Les classes de la couche métier n'interrogent plus directement le système de stockage
  - Elles passent par la DAO qui fait office d'interface entre la couche métier et celle de stockage des données

Le pattern DAO

- La couche DAO cache ainsi à la couche métier (et au reste de l'application) la manière dont les données sont stockées
  - Exemple : La classe UtilisateurDAO disposant des méthodes
    - ajouter(Utilisateur), modifier(Utilisateur), supprimer(Utilisateur)
    - Peut faire office d'intermédiaire pour toute manipulation d'utilisateur dans la base de données
- La couche DAO peut aussi créer ses propres classes d'exception qui encapsulent et ainsi masquer les exceptions liées à la technologie de stockage utilisée.

Le pattern DAO

- Sans DAO, on accède directement au système de stockage depuis la couche métier



- Avec la mise en œuvre du pattern, l'accès aux données se fait à travers la couche DAO

<code>DEMO</code>

MERCI DE VOTRE ATTENTION