Approche Composant

- Pour pallier les défauts de l'approche objet, l'approche composant est apparue.

- Cette approche est fondée sur des techniques et des langages de construction des applications qui intègrent d'une manière homogène des entités logicielles provenant de diverses sources.

- Un composant est une boite noire, communicant avec l'extérieur à travers une interface dédiée, permettant la gestion du déploiement, de la persistance, etc.

Un Composant : Qu'est-ce que c'est ?

Définition usuelle

- Une unité regroupant les fonctionnalités concernant une même idée
- Un module logiciel autonome pouvant être installé sur différentes plates-formes
  - qui exporte des attributs et des méthodes
  - qui peut être configuré (déploiement semi automatique)
  - capable de s'auto-décrire

Intérêt

Être des briques de base configurables pour permettre la construction d'une application par composition

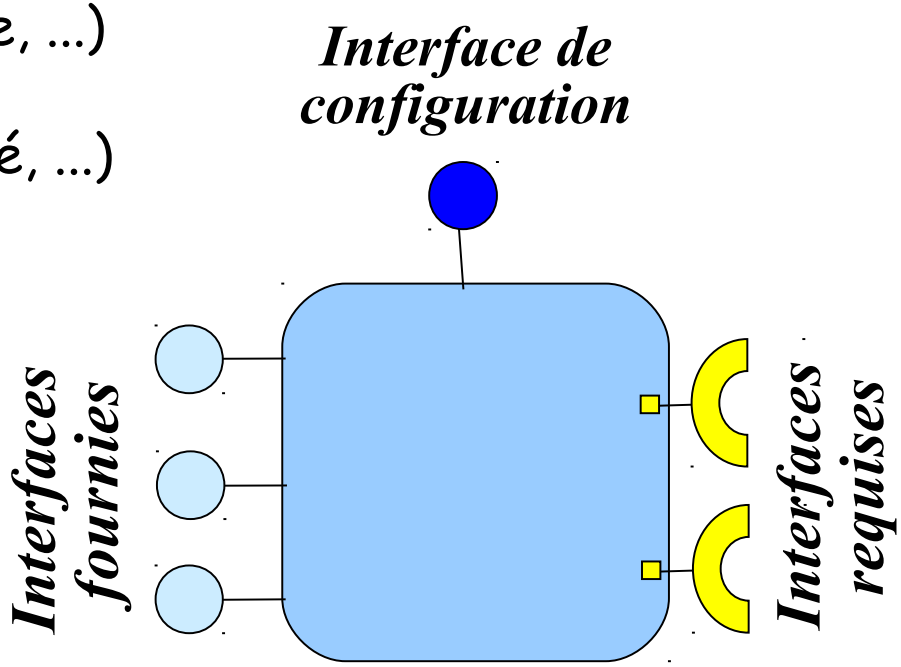
Structure d'un composant

Interactions avec un composant

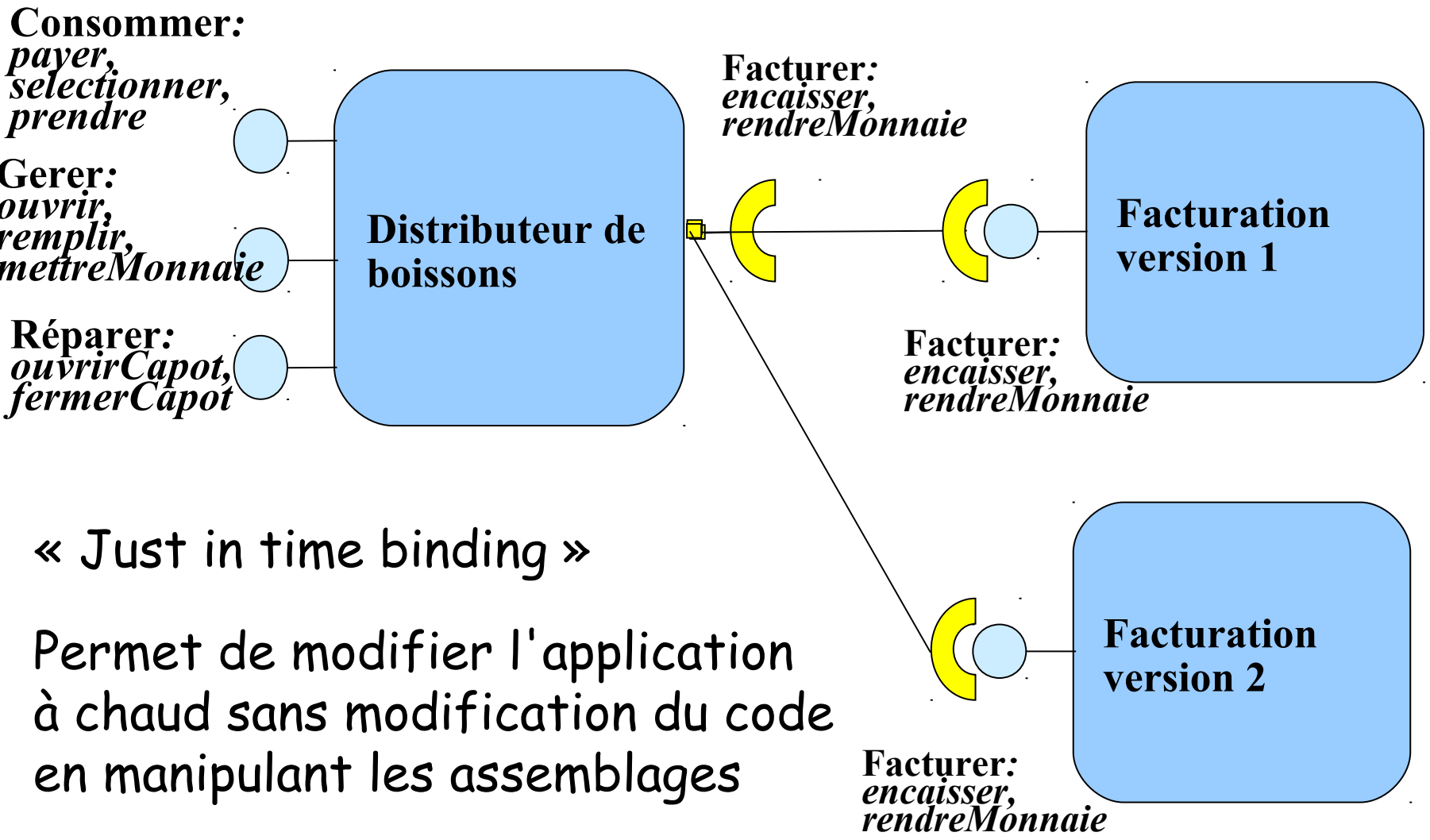
- ce qui est fourni par le composant
- ce qui est utilisé par le composant
- modes de communication

Configuration du composant

- propriétés (attributs publics)
- connexions
- cycle de vie (arret, redemarrage, ...)
- contraintes techniques (transaction, persistance, sécurité, ...)
- ...



Re-configuration dynamique



« Just in time binding »

Permet de modifier l'application à chaud sans modification du code en manipulant les assemblages

Approche Composant

- La distribution de composants fait naître de nouvelles difficultés qu'il convient de gérer efficacement afin de préserver la souplesse et de garantir l'évolutivité du système d'information.

- D'une part, l'interdépendance de composants distribués diminue la maintenabilité et l'évolutivité du système.

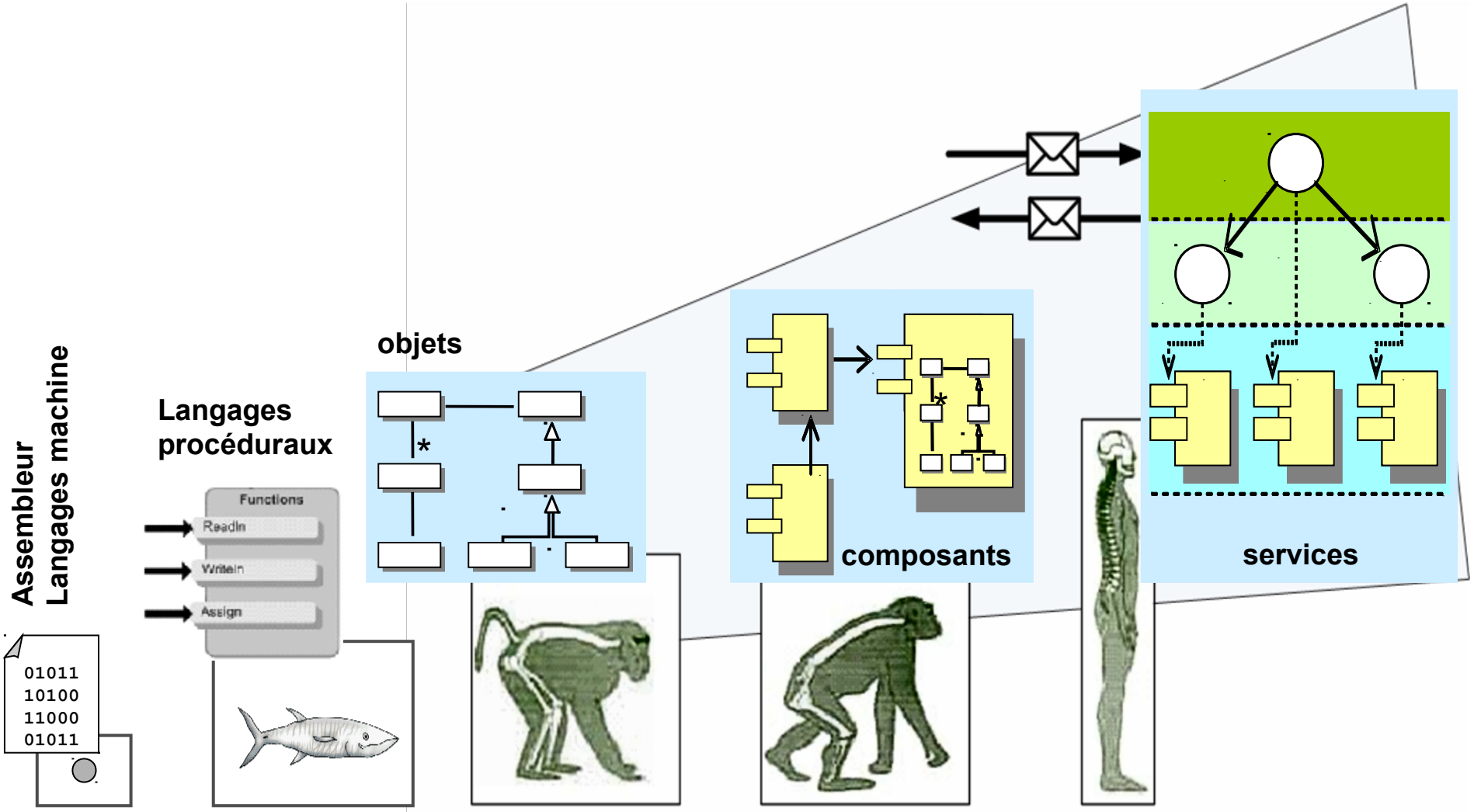
- Ainsi, on perçoit que, pour préserver son efficacité, une architecture distribuée doit minimiser l'interdépendance entre chacun de ses composants qui risque de provoquer des dysfonctionnements en cascade dont il est souvent complexe de détecter la cause et de déterminer précisément l'origine.

Approche Composant

- D'autre part, la préservation de la qualité de service du système d'information dans le cadre d'architectures distribuées est une lourde tâche, souvent complexe, en particulier lorsque les composants techniques de l'architecture sont hétérogènes et exploitent de multiples produits et standards.

- Ces difficultés font naître le besoin d'une architecture plus flexible où les composants sont réellement indépendants et autonomes, le tout permettant de déployer plus rapidement de nouvelles applications. D'où l'apparition de l'architecture orientée service.

Chronique d'une évolution

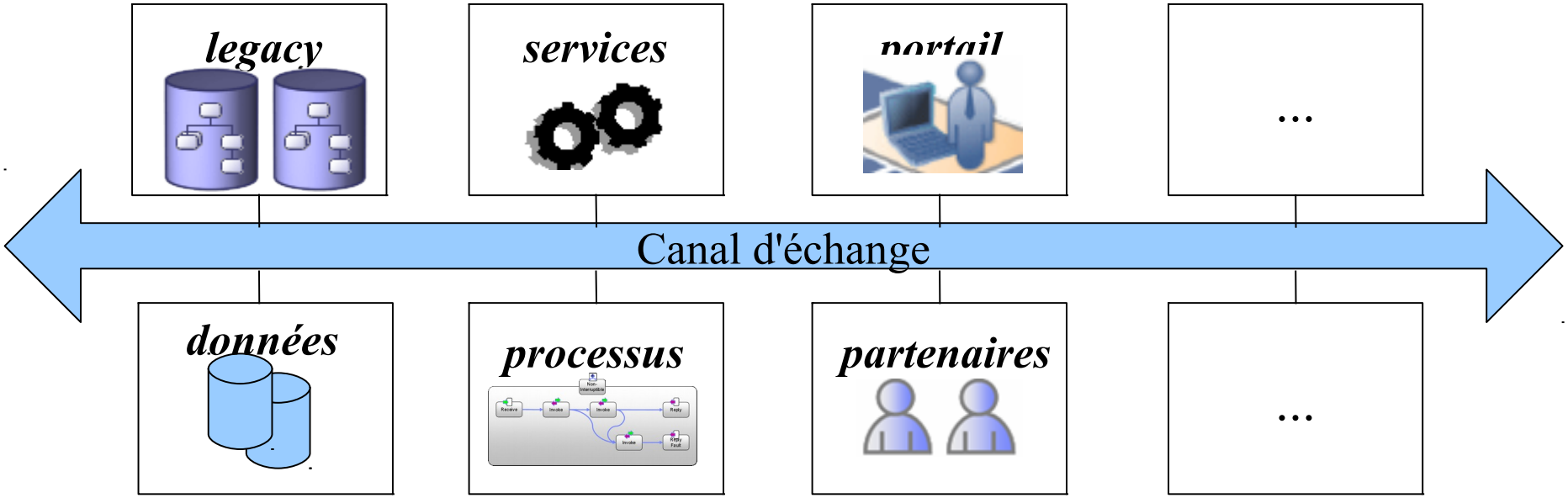


➢ ***Niveaux d'abstraction grandissant***

Demain : Architecture urbanisée

- L'**urbanisation** informatique définit l'organisation d'un SI à l'image d'une ville
  - découper le SI en modules autonomes (zone, quartier, îlot, bloc)
  - localiser les zones d'échange d'informations (routes, ponts, tunels) qui permettent de découpler les différents modules

- Objectif : faire évoluer le SI au même rythme que la stratégie et l'organisation des métiers de l'entreprise



Demain : Architecture urbanisée

Elle part du principe que l'agencement des fonctions informatiques les unes par rapport aux autres peut être défini à la manière des zones et quartiers d'une ville.

Alors que le plan de construction d'une agglomération est élaboré en fonction des besoins des habitants, de même le plan d'un système d'information sera dessiné au regard des exigences métier et/ou des priorités stratégiques de l'entreprise.

Demain : Architecture urbanisée

Rappelons qu'une démarche d'urbanisation consiste à concevoir le SI comme une ville composée de quartiers, chacun représentant un ensemble d'applications centré sur un domaine fonctionnel particulier.

Comme un système d'information, la ville dispose d'une infrastructure sur laquelle repose ses quartiers. Des interactions existent également entre eux, permettant à la ville de vivre.

Dans le cas du SI, il s'agit des processus métier qui, dans de nombreux cas, font également appel à divers systèmes pour fonctionner.

Les architectures

Niveaux d'abstraction d'une application

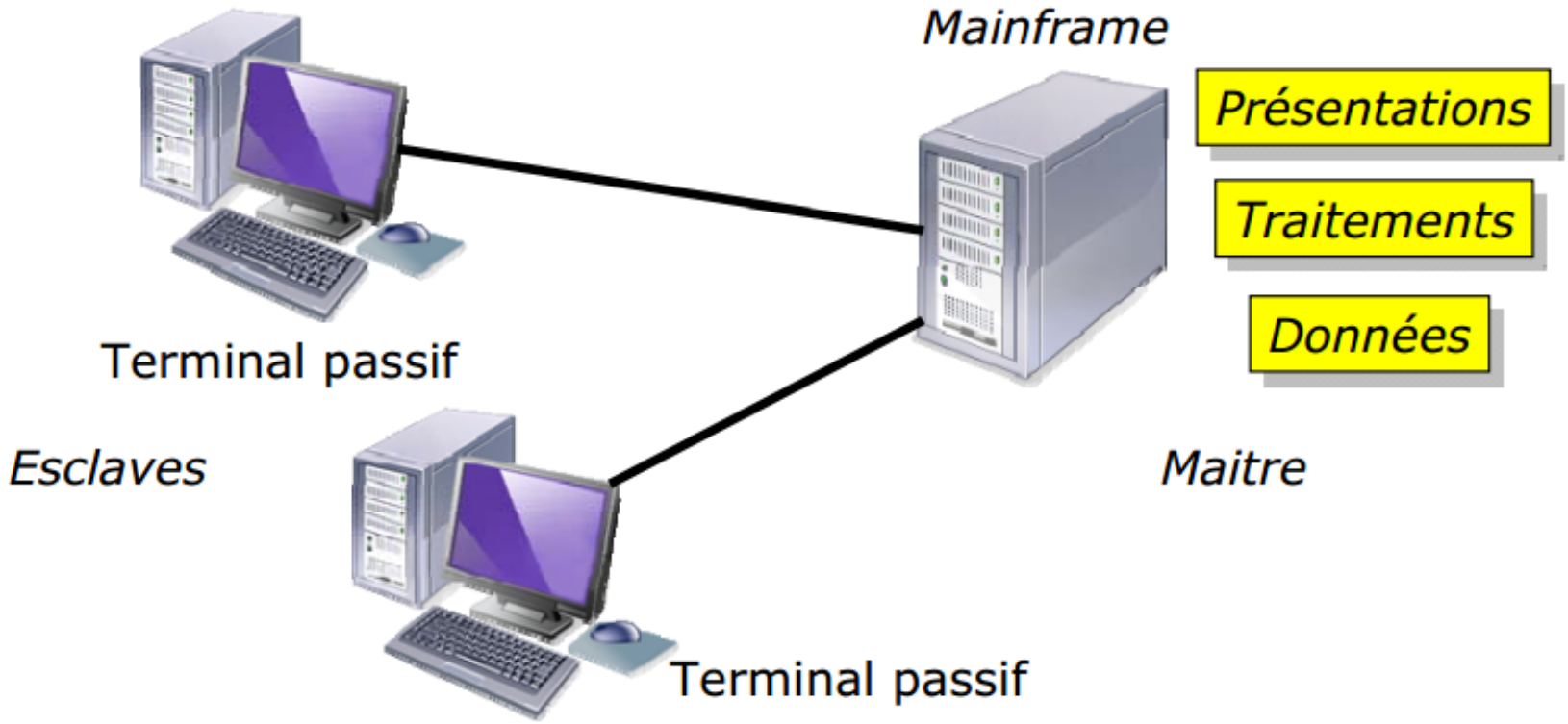
- En règle générale, une application est découpée en 3 niveaux d'abstraction :
- La couche présentation ou IHM (Interface Homme/Machine)
  - gère les interactions utilisateur/machine, la présentation
- La couche traitements :
  - Locaux : contrôles effectués au niveau du dialogue avec l'IHM
  - Globaux : L'application elle-même
- La couche données :
  - Gère le stockage des données et l'accès à ces dernières

Niveaux d'abstraction d'une application

- Ces 3 niveaux peuvent être imbriqués ou répartis de différentes manières entre plusieurs machines physiques ou logiques
- Suivant les contraintes d'utilisation ou contraintes techniques ➔ Orientation vers une architecture adéquate
  - Architecture 1-tiers
  - Architecture 2-tiers
  - Architecture 3-tiers
  - Architecture n-tiers

Architecture 1-tiers

- Les 3 couches applicatives s'exécutent sur la même machine
- On parle d'informatique centralisée :
- Contexte multi-utilisateurs dans le cadre d'un site central (mainframe)



Architecture 1-tiers

Avantages

- Mainframe : la fiabilité des solutions sur site central qui gèrent les données de façon centralisée
- Un tiers déployé : l'interface utilisateur moderne des applications.

Limites

- Mainframe : interface utilisateur en mode caractères
- Un tiers déployé : cohabitation d'applications exploitant des données communes peu fiable au delà d'un certain nombre d'utilisateurs.

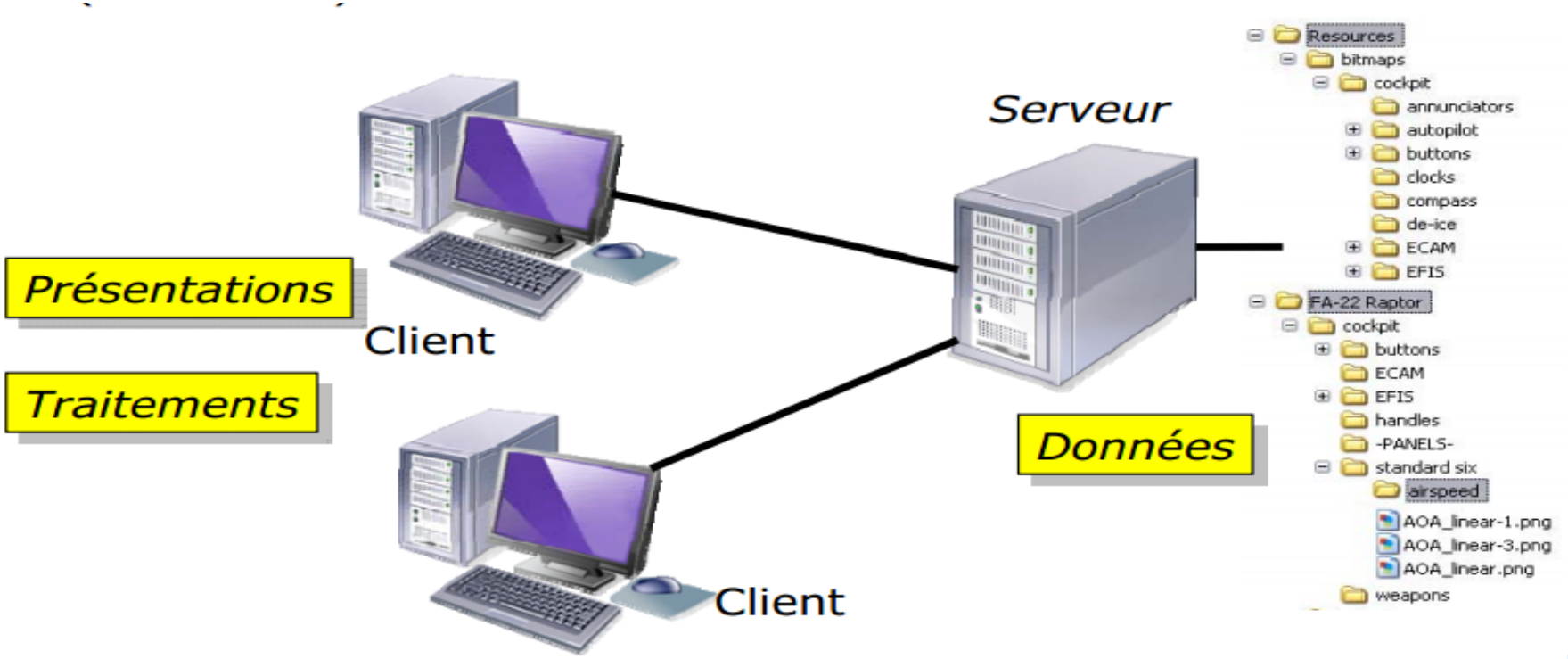
Conclusion

Il a donc fallu trouver une solution conciliant les avantages de cette architecture . Pour se faire, il a fallu scinder les applications en plusieurs parties distinctes et coopérantes : gestion centralisée des données, gestion locale de l'interface utilisateur.

Ainsi est né le concept du client-serveur

Architecture 2-tiers: Client-serveur

- Présentation et traitements sont sur le client
- Les données sur le serveur
- Contexte multi-utilisateurs avec accès aux données centralisées (middleware)



Architecture 2-tiers: Client-serveur

- **Avantages**

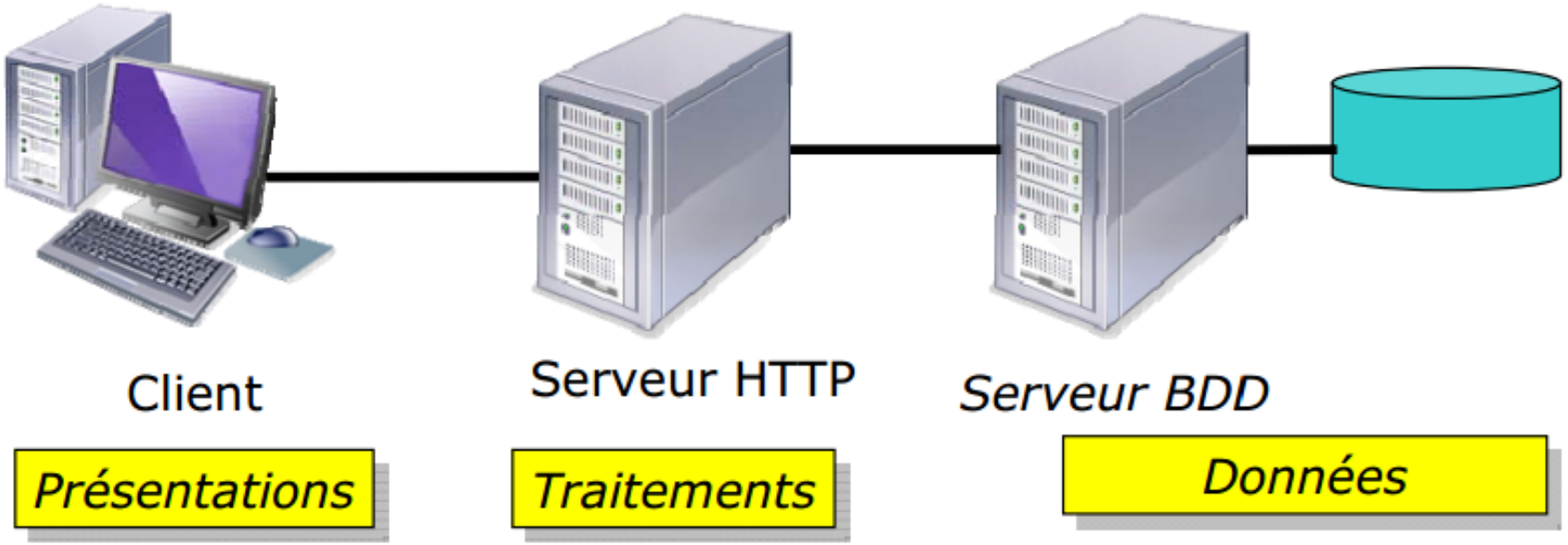
- Interface utilisateur riche
- Appropriation des applications par l'utilisateur

- **Limites**

- Importante charge du poste client, qui supporte la grande majorité des traitements applicatifs
- Maintenance et mises à jour difficiles à gérer
- Difficulté de modifier l'architecture initiale

Architecture 3-tiers

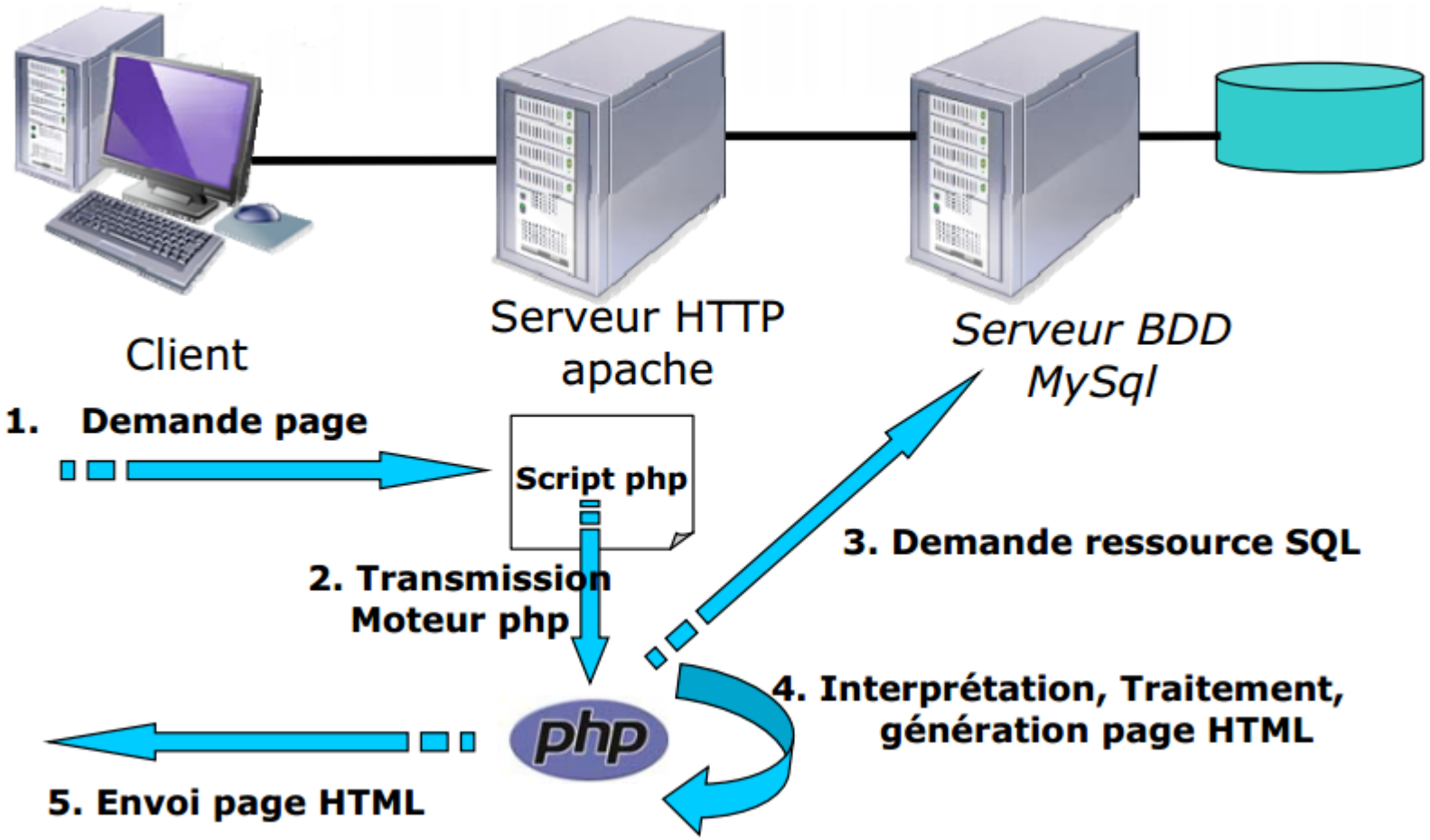
- La présentation est sur le client
- Les traitements sont pris par un serveur intermédiaire
- Les données sont sur un serveur de données
- Contexte multiutilisateur internet



Architecture 3-tiers

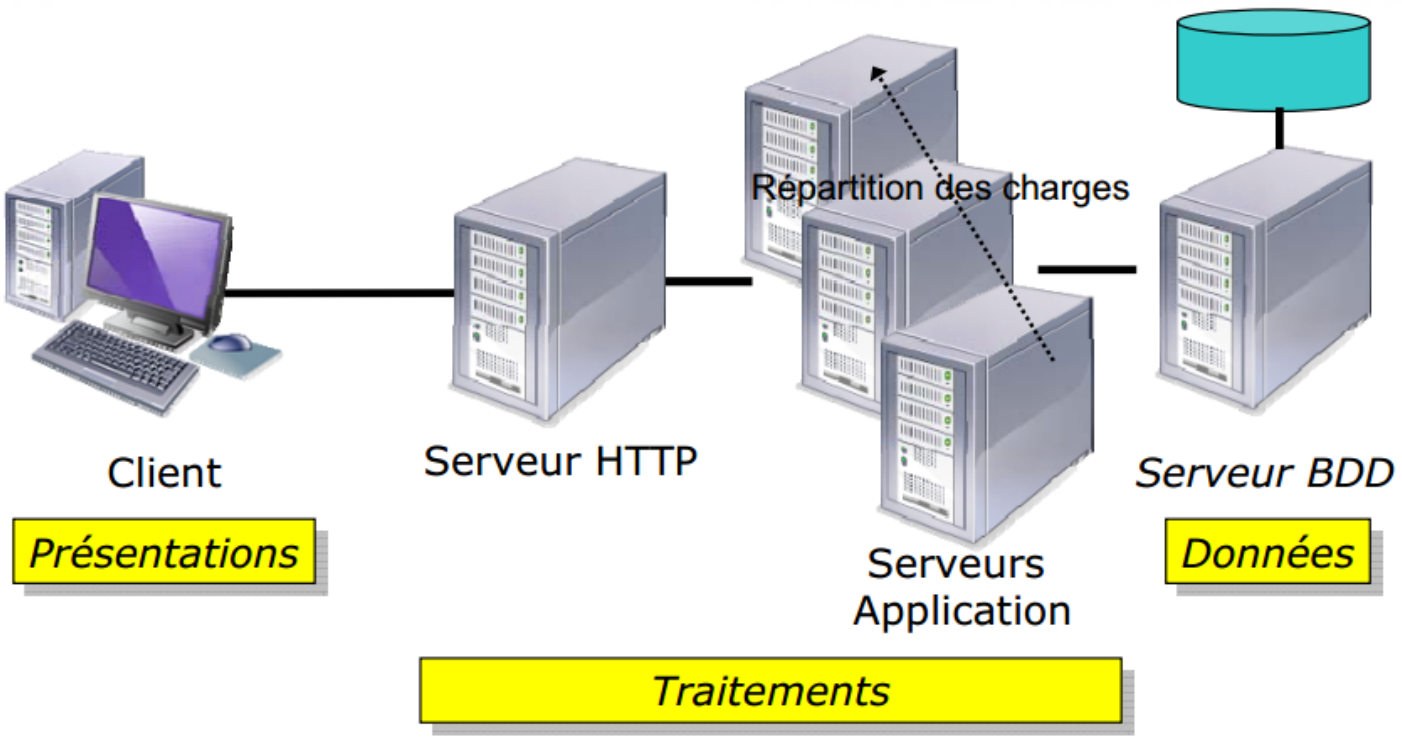
- Tous ces niveaux étant indépendants, ils peuvent être implantés sur des machines différentes, de ce fait :
- Le poste client ne supporte plus l'ensemble des traitements (client léger)
- Facilité de déploiement
- Sécurité : pas d ’exposition du schéma de la base de données
- La manipulation des données est indépendante du support physique de stockage
- Il est relativement simple de faire face à une forte montée en charge, en renforçant le service applicatif.

Architecture 3-tiers: Exemple php/MySQL



Architecture n-tiers

- La présentation est sur le client
- Les traitements sont pris par un serveur intermédiaire
- Les données sont sur un serveur de données
- Contexte multi-utilisateurs internet



Architecture n-tiers

- Le terme « client lourd », *en anglais « fat client » ou « heavy client » par opposition au client léger,*

  désigne une application cliente graphique exécutée sur le système d'exploitation de l'utilisateur.

- Un client lourd possède généralement des capacités de traitement évoluées et peut posséder une interface graphique sophistiquée.
- Néanmoins, ceci demande un effort de développement et tend à mêler la logique de présentation (l'interface graphique) avec la logique applicative (les traitements).

Architecture n-tiers

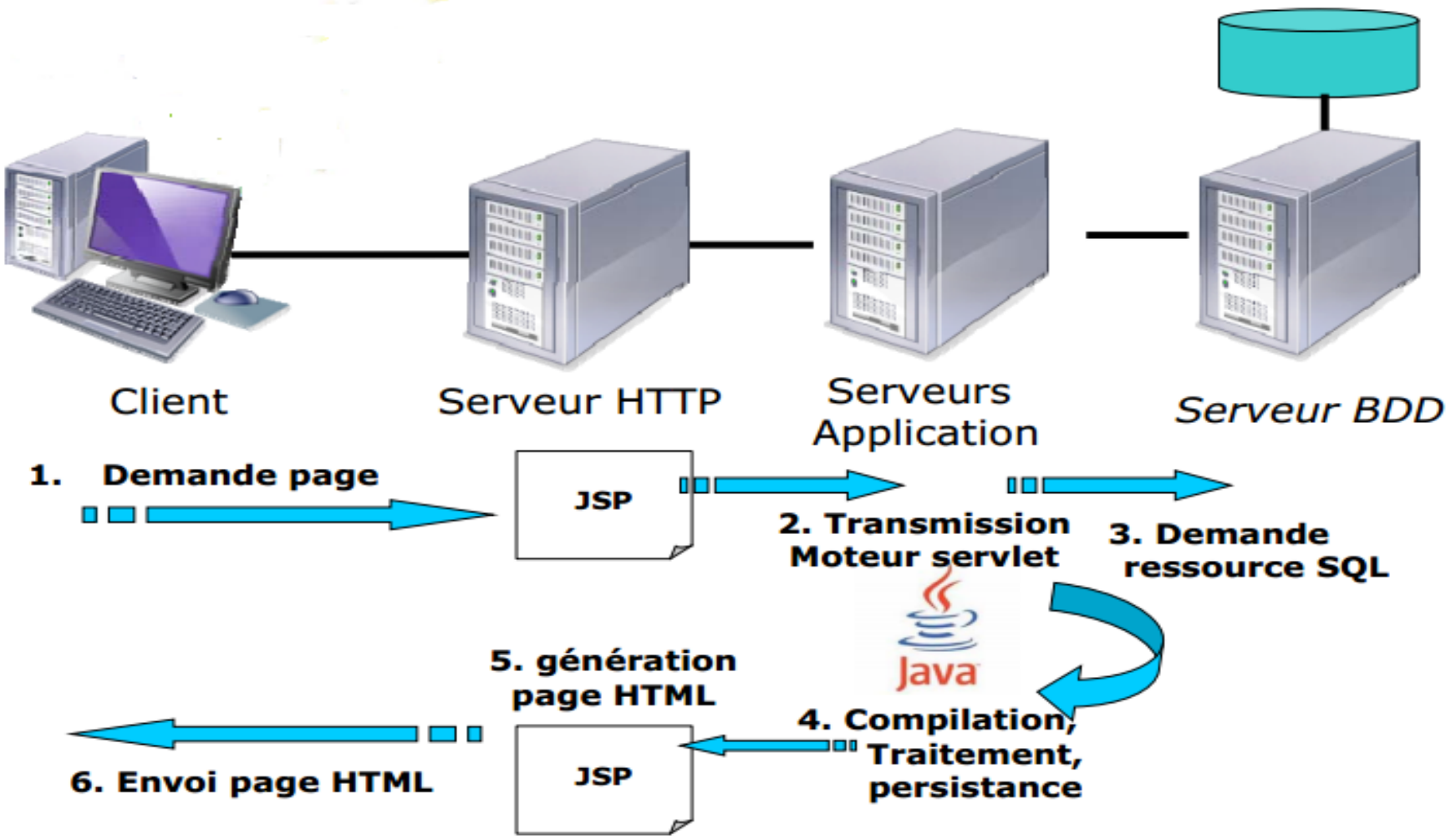
- Le terme « client léger », parfois « client pauvre », en anglais « thin client » par opposition au client lourd, désigne une application accessible via une interface web (en HTML) consultable à l'aide d'un navigateur web, où la totalité de la logique métier est traitée du côté du serveur.
- Pour ces raisons, le navigateur est parfois appelé client universel.

- L'origine du terme provient de la limitation du langage HTML, qui ne permet de faire des interfaces relativement pauvres en interactivité, si ce n'est pas le biais du langage javascript, DHTML, XHTML, etc..

Architecture n-tiers

- Un « **client riche** » est un compromis entre **le client léger** et **le client lourd**.
- L'objectif du client riche est donc de :
  - proposer une interface graphique, décrite avec une grammaire de description basée sur la syntaxe XML,
  - obtenir des fonctionnalités similaires à celles d'un client lourd (glisser déposer, onglets, multi fenêtrage, menus déroulants, …).
- Il existe des standards permettant de définir une application riche :
  - **XAML** (*eXtensible Application Markup Language*), prononcez « zammel », un standard XML proposé par Microsoft ;
  - **XUL**, prononcez « zoul », un standard XML proposé par la fondation Mozilla ;
  - **Flex**, un standard XML proposé par la société Macromedia

Architecture n-tiers: Exemple Java



Architecture Web Services

- Type d'architecture reposant sur les standards de l'Internet
- Alternative aux architectures classiques :
  - Client/serveur
  - n/tiers
- Orientée services permettant à des applications de communiquer sans préoccupation des technologies d'implantations utilisées de part et d'autre
- Priorité : Interopérabilité
- Née fin 90 (Microsoft, IBM, SAP)
- Basée sur les technologies XML

Architecture Web Services

- Web-services: logiciel qui interagit avec d'autres au moyen de protocoles & langages universels (http, xml …)
- Deux formes de services-web :
  - XML-RPC : Remote Procedure Call
  - SOAP : Simple Object Access Protocol
- Présentent 2 caractéristiques :
  - Enregistrement (facultatif) auprès d'un service de recherche (UDDI)
  - Interface publique avec laquelle le client invoque le service Web (WSDL)

Architecture Web Services

- **UDDI :** ***Universal Desciption, Discovery and Integration peut*** être vu comme les pages blanches (ou jaunes) des services-web.
  - C'est un annuaire permettant à des fournisseurs de présenter leurs services à des '*clients*'.
- **WSDL :** ***Web Service Description Language est un langage*** reposant sur XML dont on se sert pour décrire les servicesweb.
  - Il est indispensable à UDDI pour permettre aux clients de trouver les méthodes leur permettant d'invoquer les services web.

Architecture Web Services

- **SOAP : *Simple Object Access Protocol est un protocole basé*** sur XML et qui définit les mécanismes d'échanges d'information entre les clients et les fournisseurs de serviceweb.
  - Les messages SOAP sont susceptibles d'être transportés en HTTP, SMTP, FTP...
- **XML-RPC : protocole RPC (*Remote Procedure Call) basé sur*** XML. Permet donc l'invocation de procédure distante sur internet.

Architecture Web Services

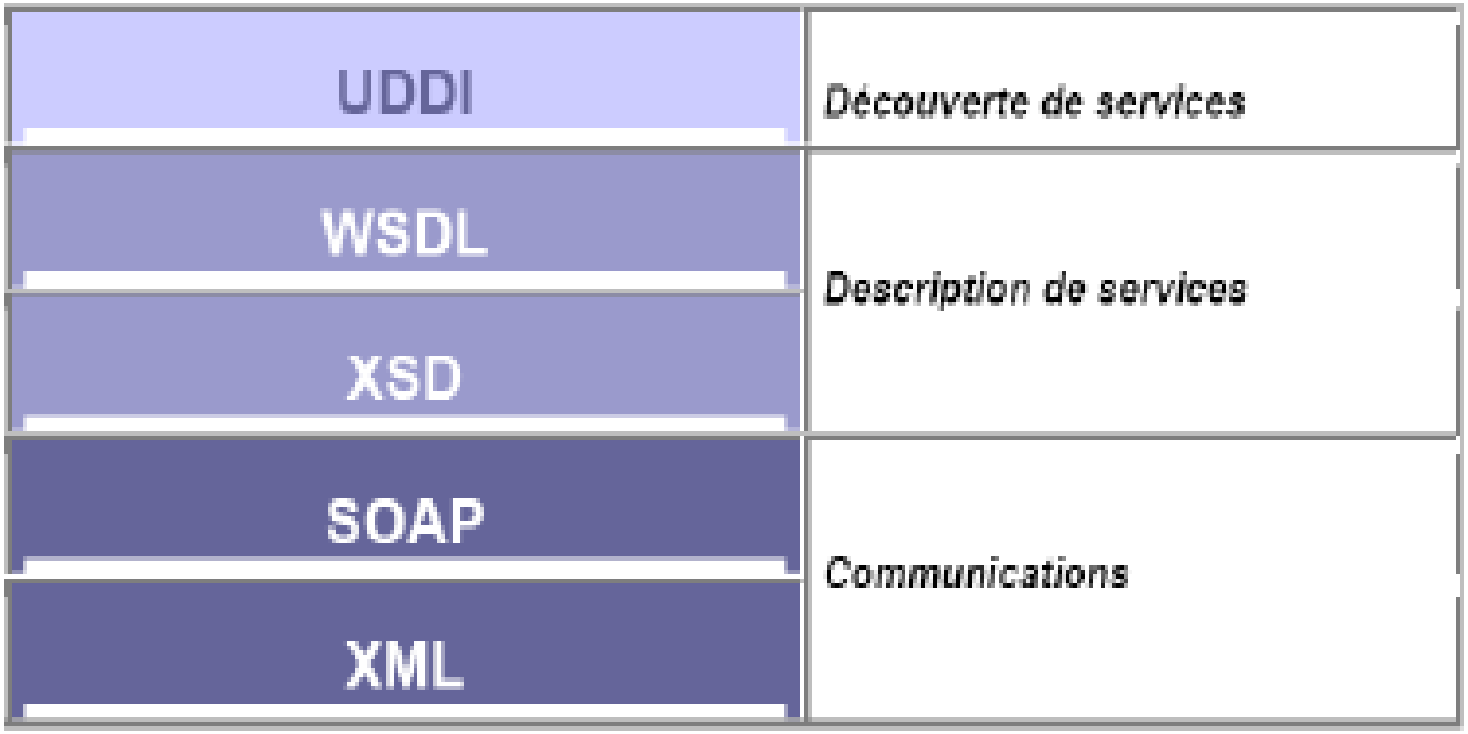
Interopérabilité

- Capacité des services Web à faire converser des applications & des composants hétérogènes
- Exemple :
  - Réalisation d'un service Web permettant de donner le cours d'une action en bourse, fonctionnant sous **Linux en Java**
  - Et l'interroger depuis une page Web **Asp.net en même** temps que depuis une application **PERL ou PHP**

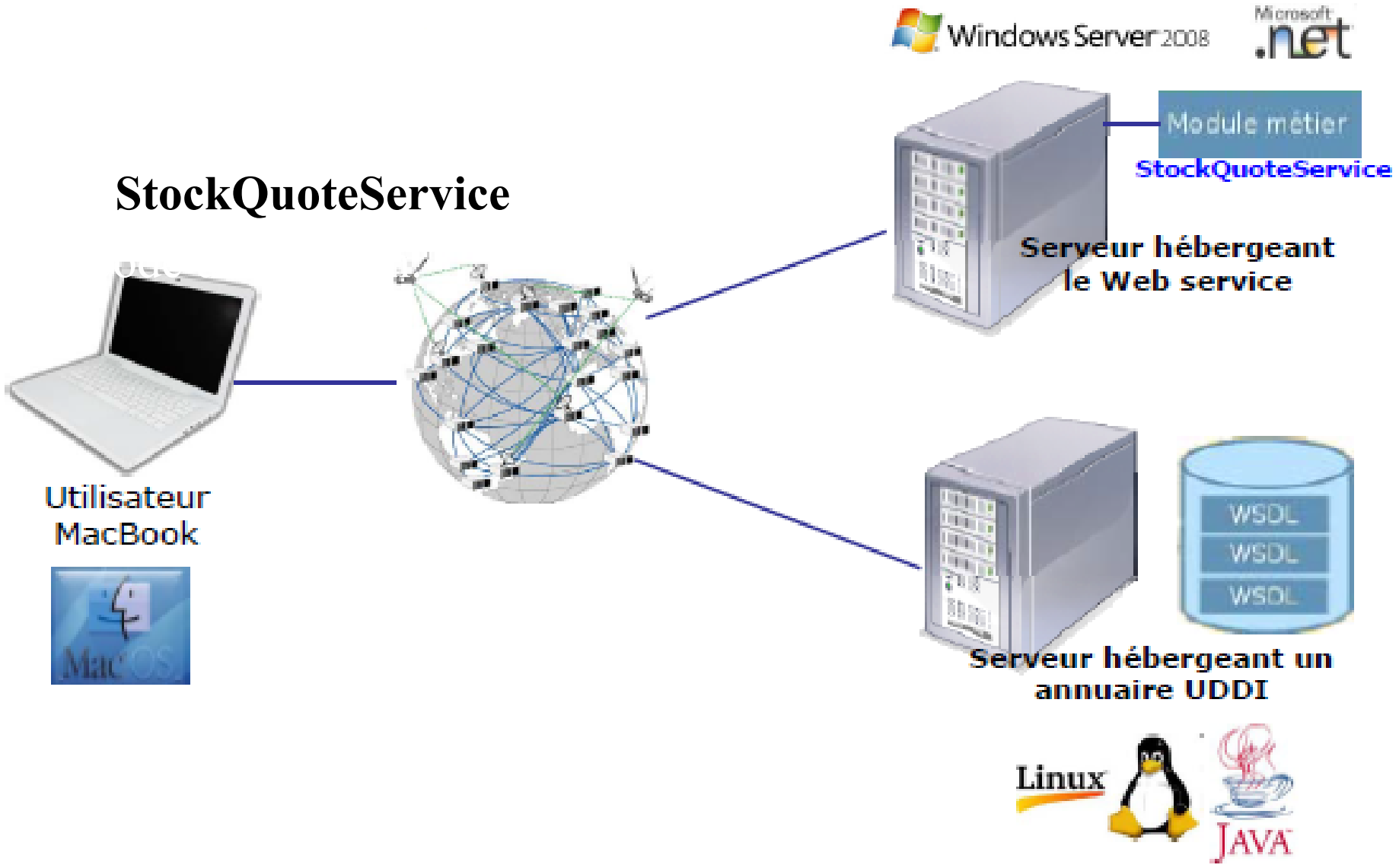
Architecture Web Services

- **Fonctionnement – Couches**

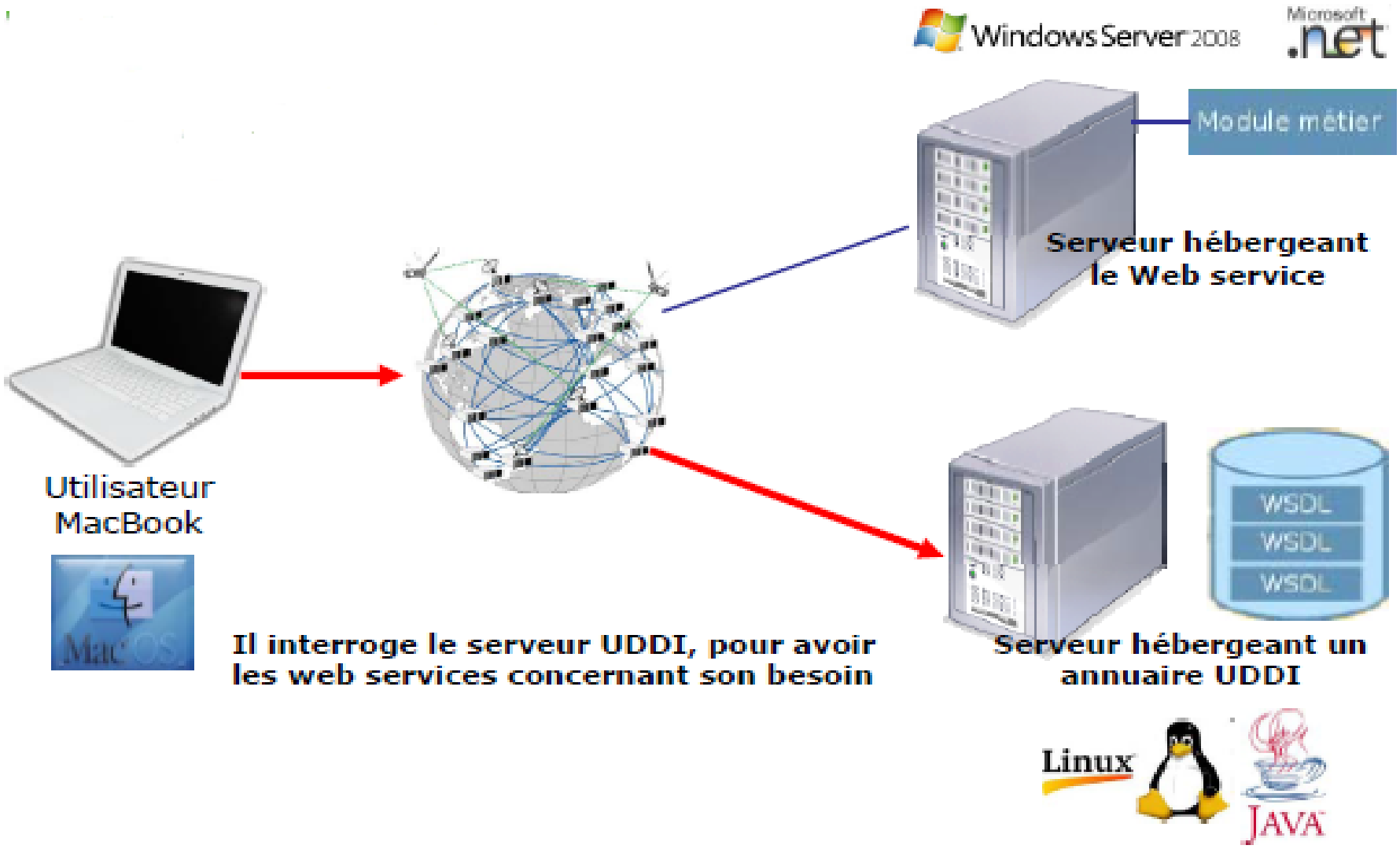
Toutes ces technologies sont disposées en couches et constitue l'architecture services Web



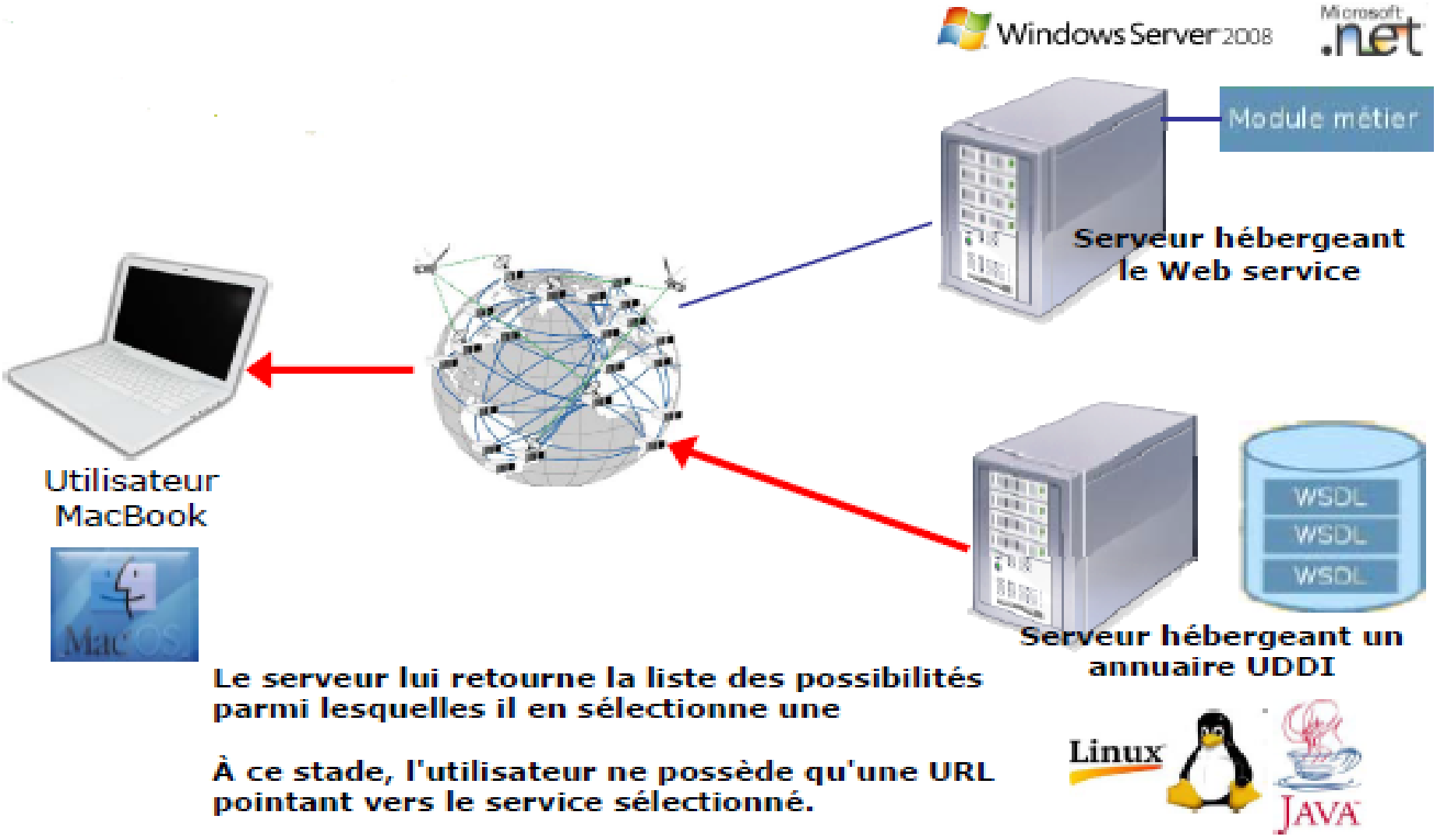
Architecture web service: Exemple (1/8)



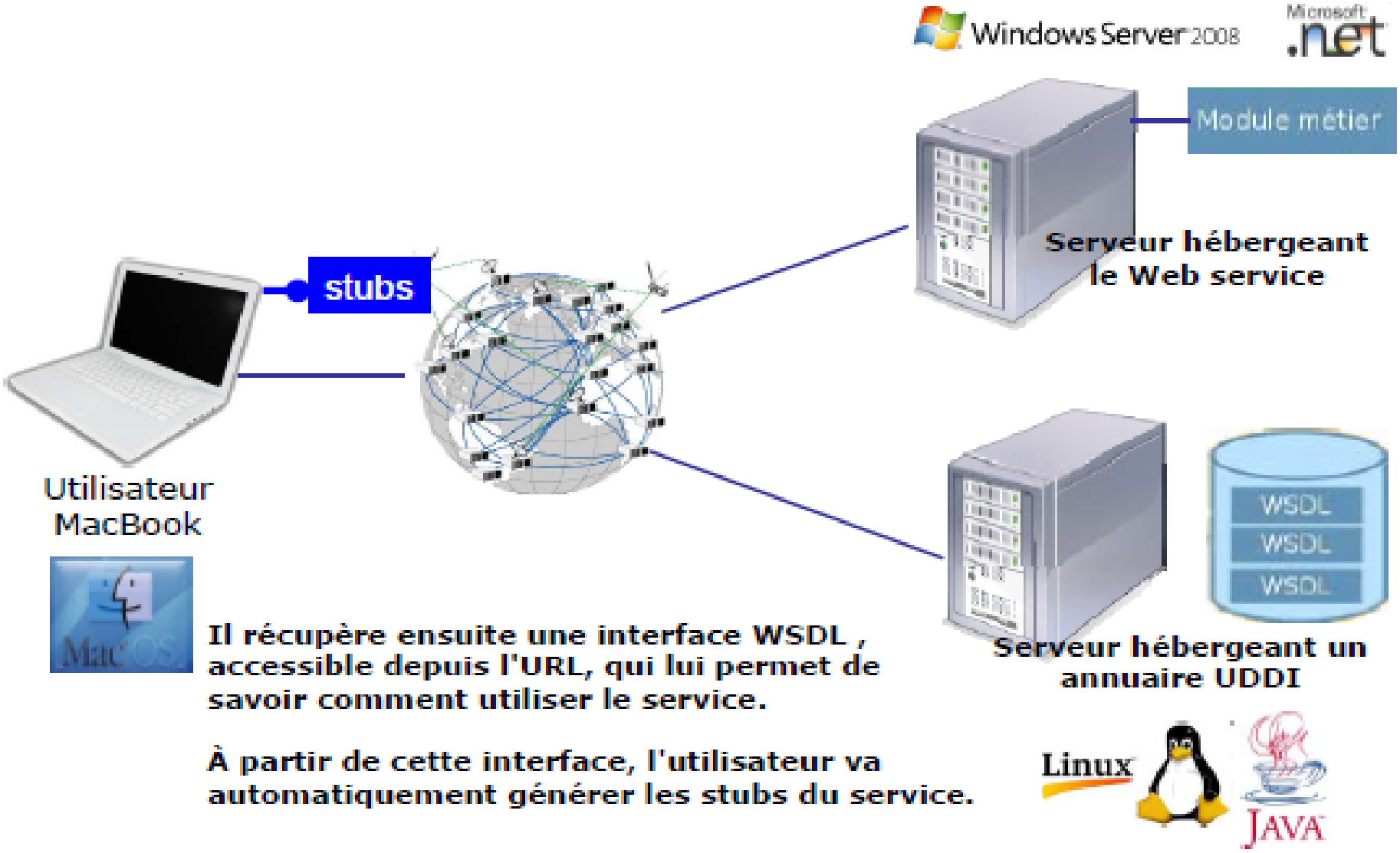
Architecture web service: Exemple (2/8)



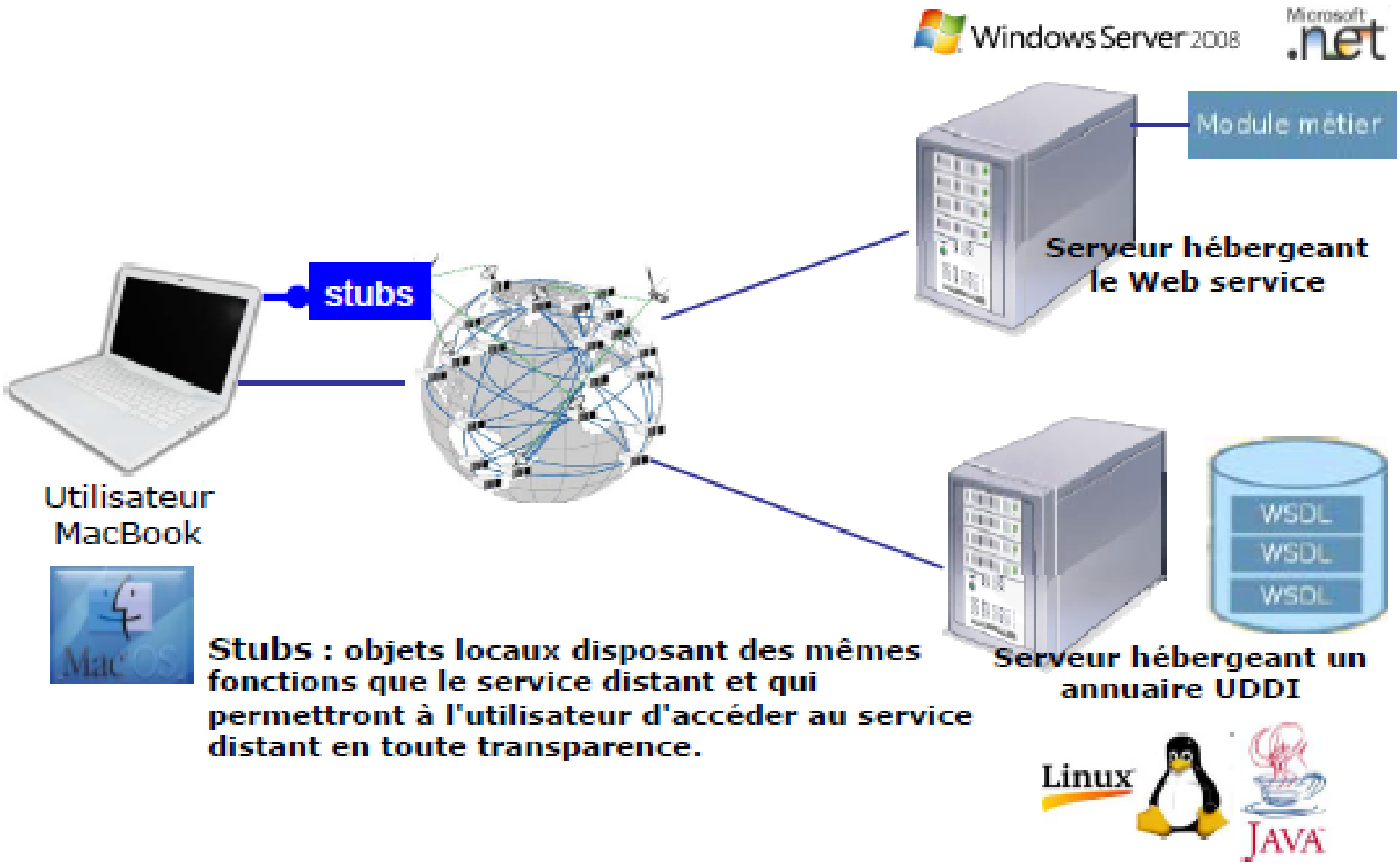
Architecture web service: Exemple (3/8)



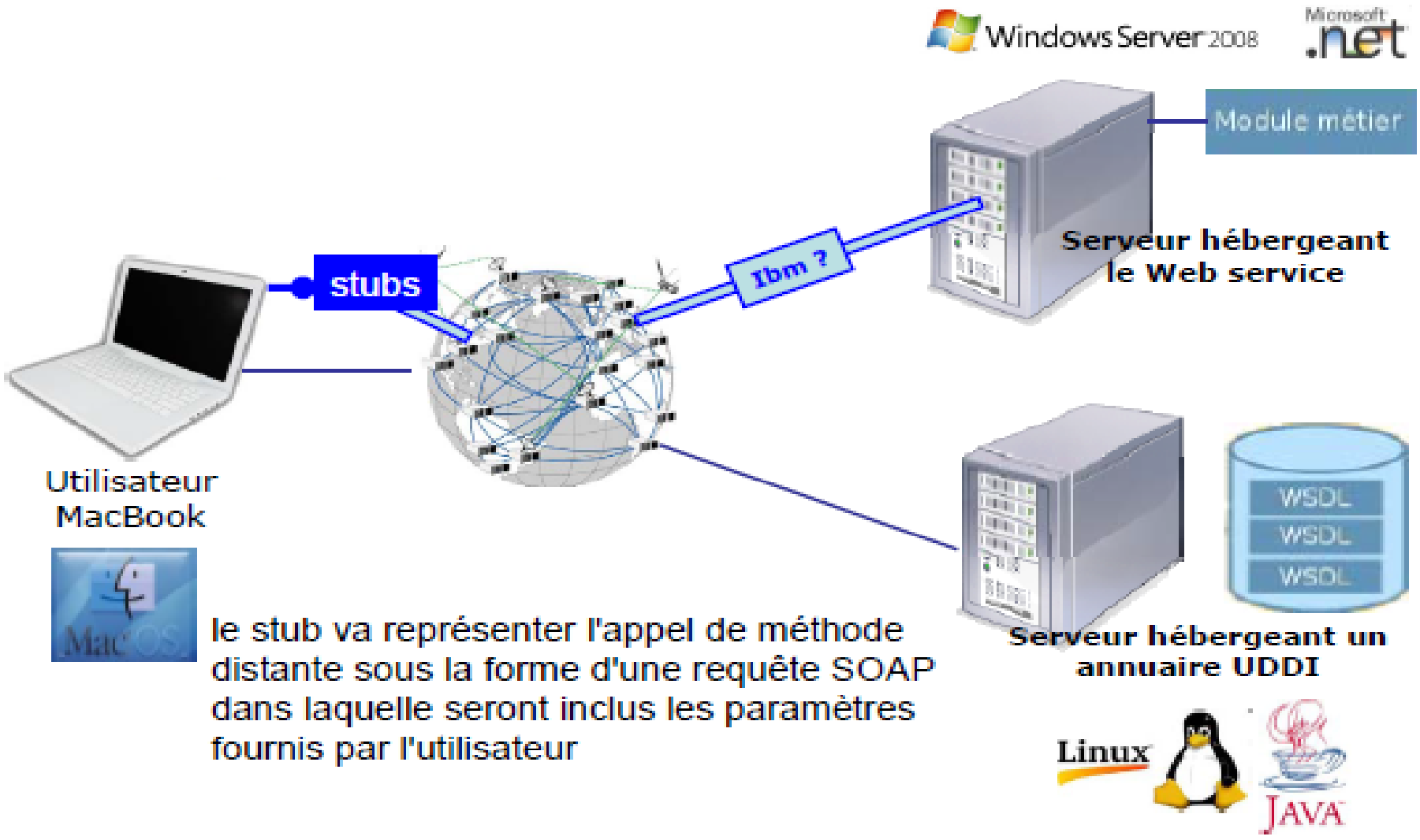
Architecture web service: Exemple (4/8)



Architecture web service: Exemple (5/8)

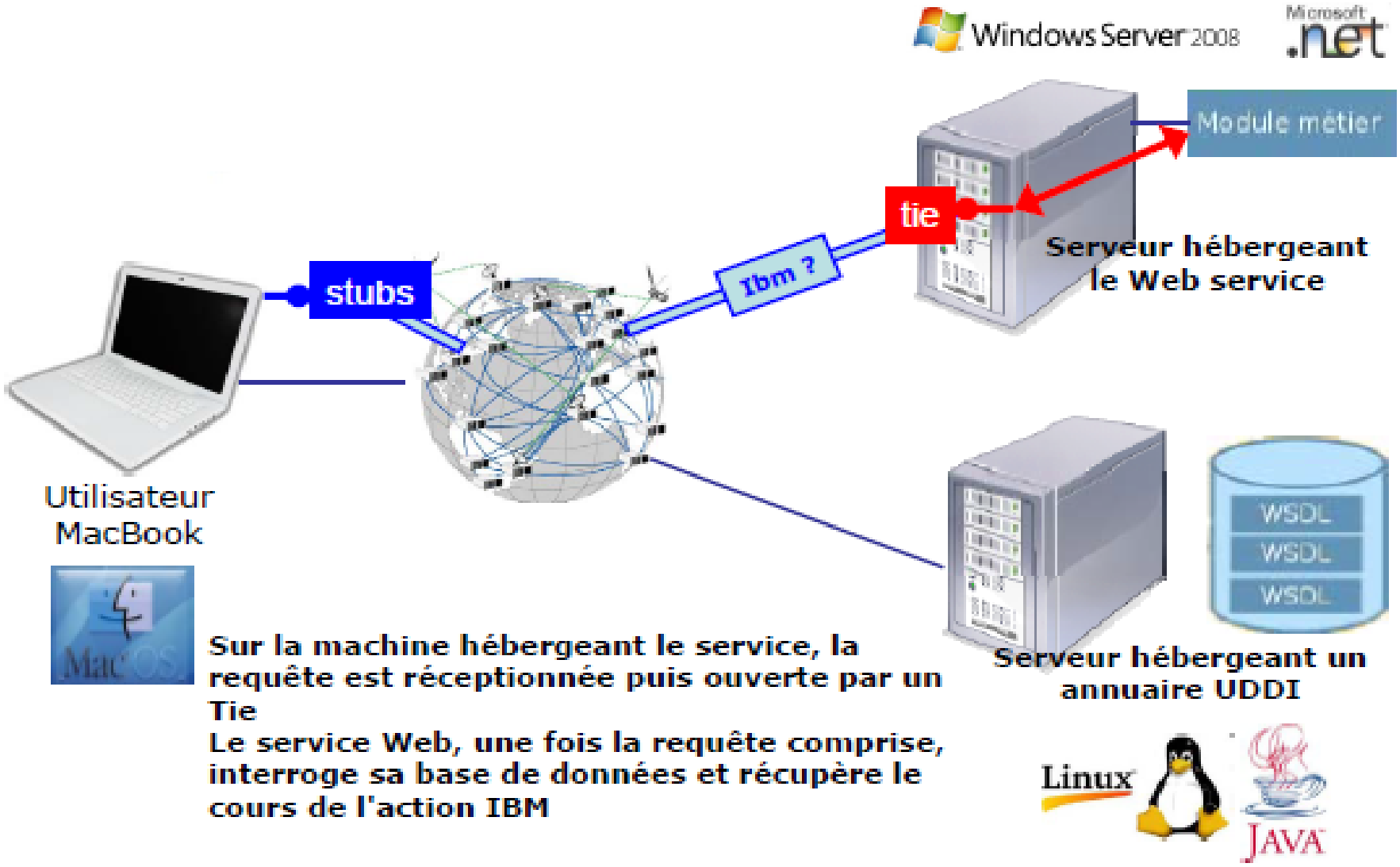


Architecture web service: Exemple (6/8)

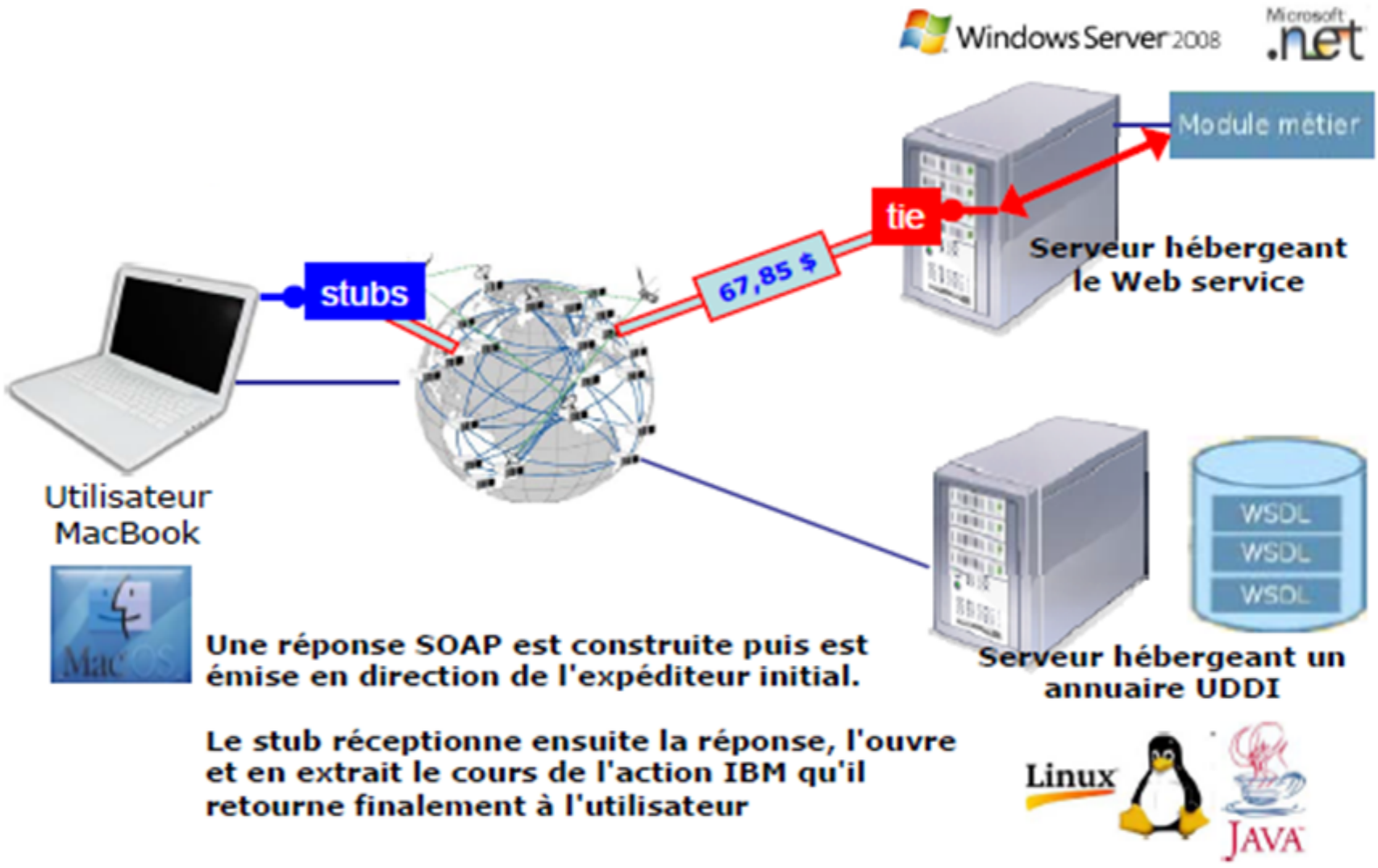


le stub va représenter l'appel de méthode distante sous la forme d'une requête SOAP dans laquelle seront inclus les paramètres fournis par l'utilisateur

Architecture web service: Exemple (7/8)



Architecture web service: Exemple (8/8)



Quels sont les principes de base du SOA ?

Principes fondamentaux de l'architecture SOA

Il n'existe pas une recette pour garantir le succès de la mise en place d'une SOA mais des principes à respecter :

- Discussion entre métier et IT
- Utilisation des use case métier

- Utilisation de standards
- Pas de remise en cause de l'existant lors d'évolutions technologiques

- Découplage entre fournisseur et consommateur de services
- Indépendance des ressources vis à vis de ceux qui les utilisent

Qu'est ce qu'un Service (au sens SOA) ?

- Partage les caractéristiques suivantes d'un objet
  - Modulaire (ensemble de fonctionnalités qui font sens)

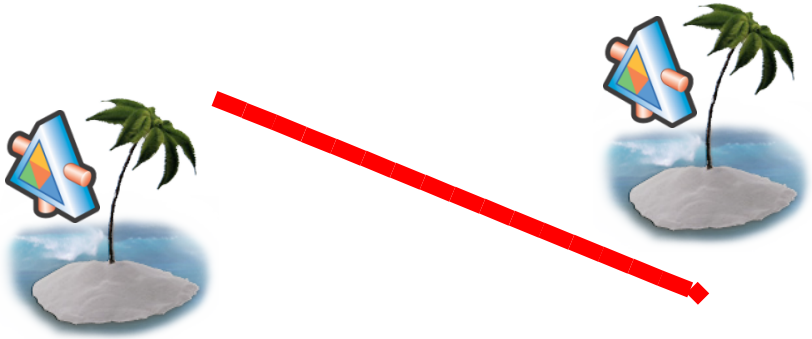
- Partage les caractéristiques suivantes d'un composant
  - Boite noire (séparation interface/implémentation)
  - Indépendant de la localisation
  - Neutralité vis-à-vis des protocoles de transport

- Correspond à un périmètre fonctionnel que l'on souhaite exposer à des consommateurs
- Est faiblement couplé (indépendant des autres services)
- Expose un petit nombre d'opérations offrant un traitement de bout en bout
- Sans état

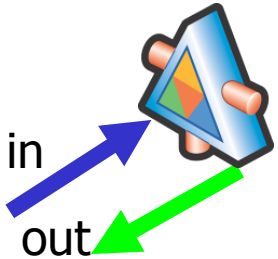
4 propriétés du service à retenir

- Un Service est <u>Autonome</u> et sans <u>état</u>

- Les Front[illegible]ntre services sont <u>Explicites</u>



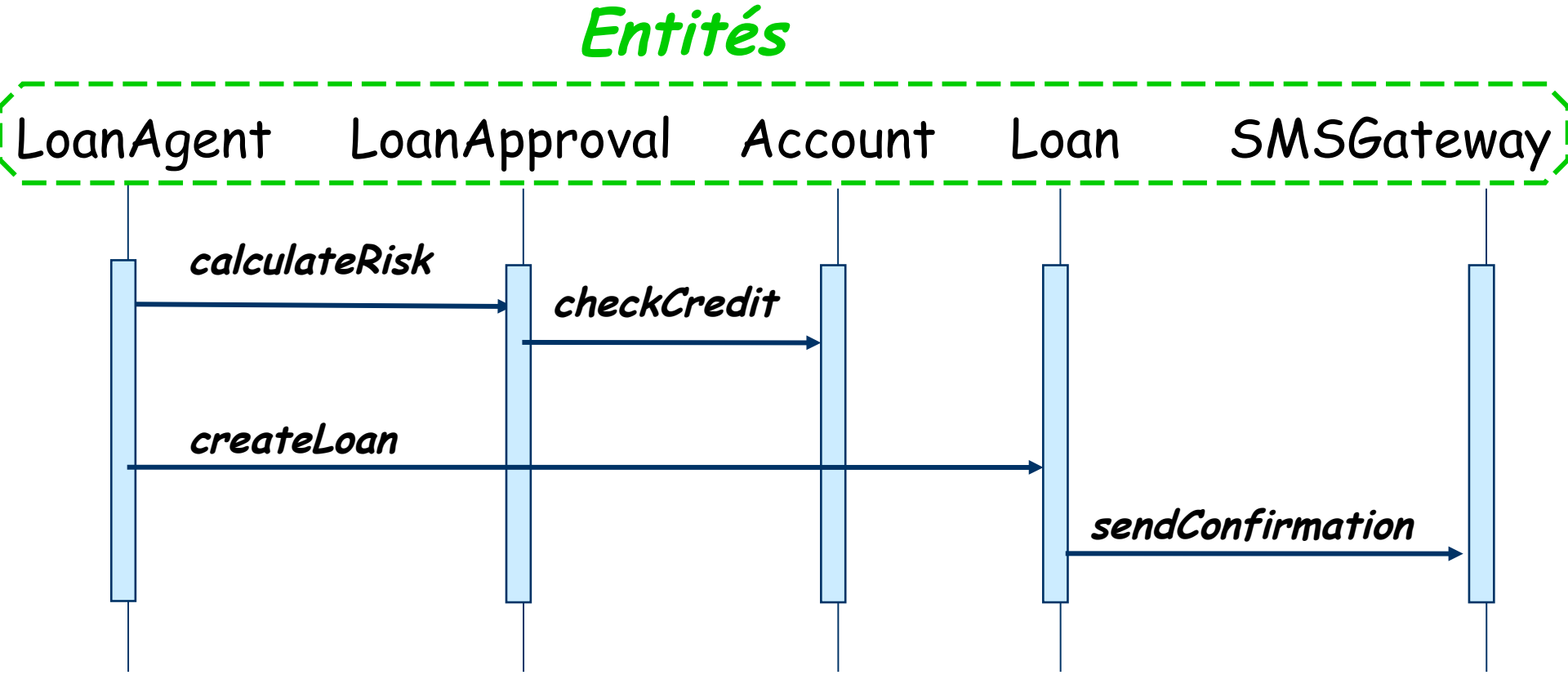
- Un Service expose un <u>Contrat</u>



Conditions Générales de Vente
Règlement Intérieur
Vos droits/Vos devoirs

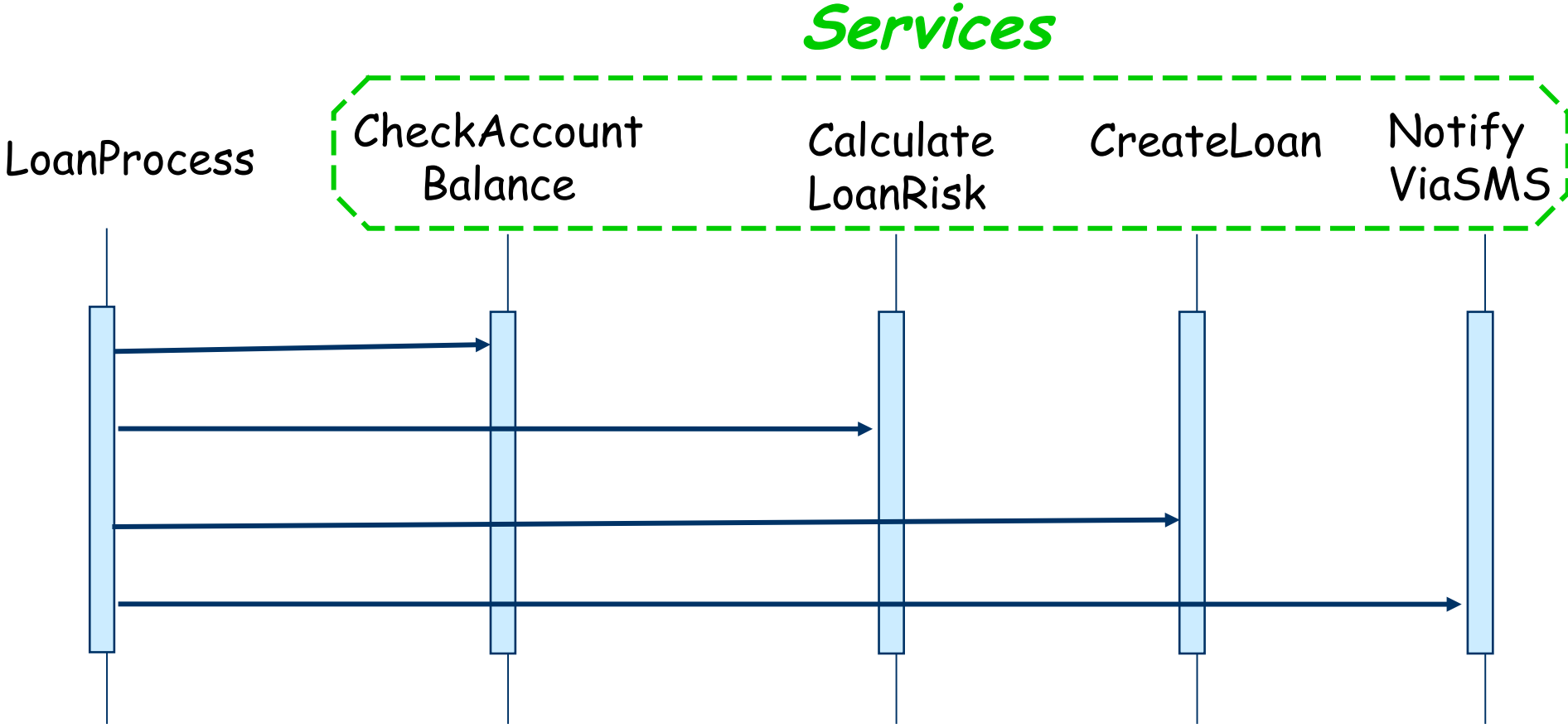
- Les services communiquent par messages

Exemple de couplage fort : Gestion de prêts



- LoanAgent est lié à LoanApproval et Loan
- LoanApproval est lié à Account
- Loan est lié à SMSGateway

Gestion de prêts en couplage faible



- Qu'est ce que LoanProcess ?
- Un **processus métier** !
  Il permet d'**orchestrer** les services => couplage lâche

Définition: Processus métier

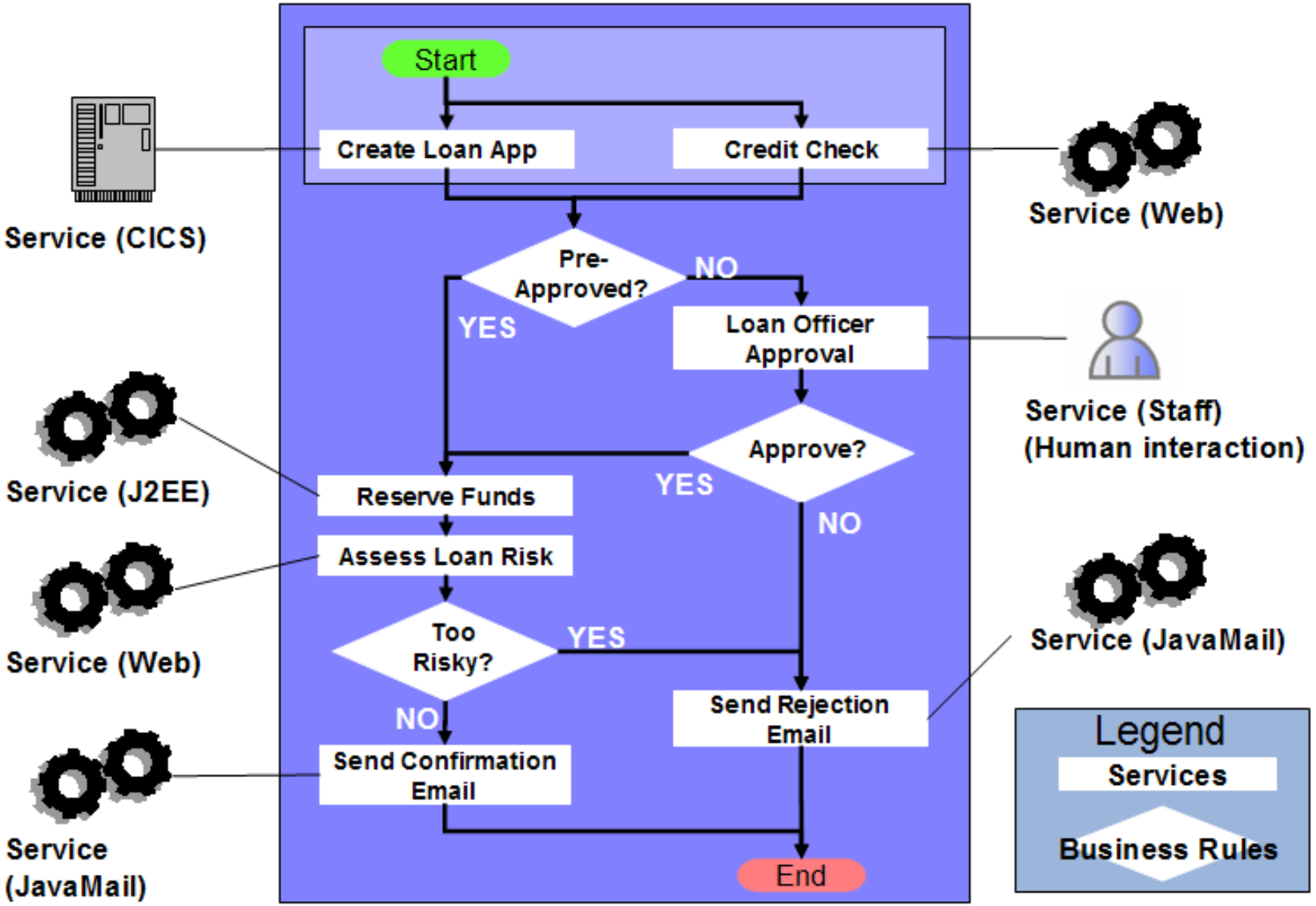
- un processus métier est une transformation qui produit une valeur ajoutée tangible à partir d'une sollicitation initiale.
- Il est divisé en activités (ou tâches), réalisées par des acteurs (humains ou automatiques) avec l'aide de moyens adaptés, qui contribuent chacune à l'obtention du résultat escompté.
- Le caractère métier du processus s'exprime par la nature du résultat, qui doit avoir un sens pour un client (client au sens large, interne ou externe) et mesurable.

Business Process Management (BPM)
Gestion du Processus métier

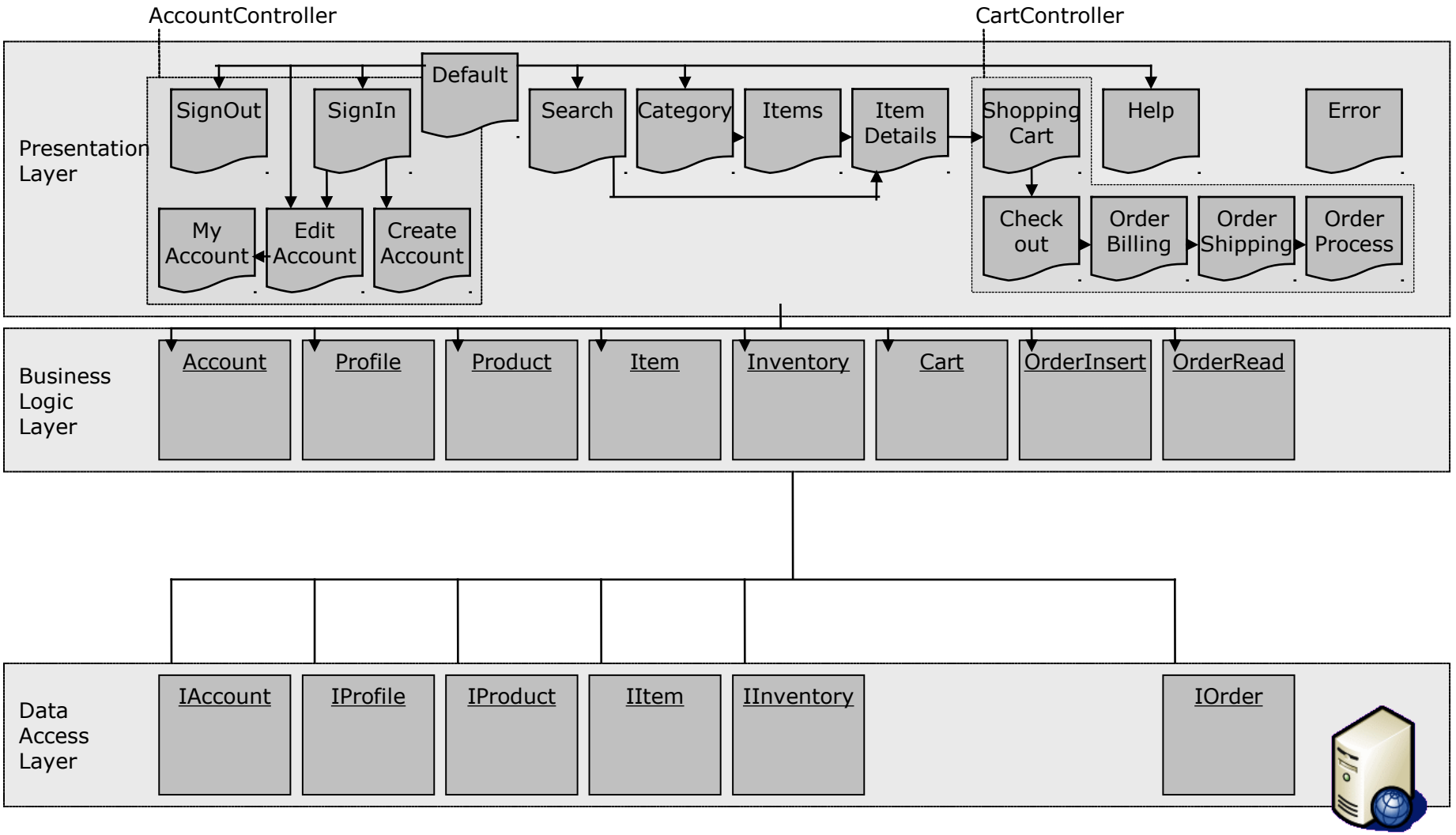
- But : Donner à l'Entreprise les moyens de gérer ses processus métiers de manière informatisée (modélisation, simulation, exécution et audit)
  - Optimisation, adaptation aux besoins en temps réel

- Un **processus** est composé de **sous processus**, de **décisions** (Business rules) et d'**activités**
- Un sous processus a son propre but, entrées et sorties
- Les activités
  - correspondent aux parties du processus métier qui n'incluent pas de décision et sont associées à des rôles
  - Sont réalisées par des systèmes ou des humains
- Des **mesures** (KPI pour Key Performance Indicators) permettent de capturer les performances du processus
- Un processus est le **résultat d'une orchestration** de service
- Le processus est lui-même accessible en tant que service

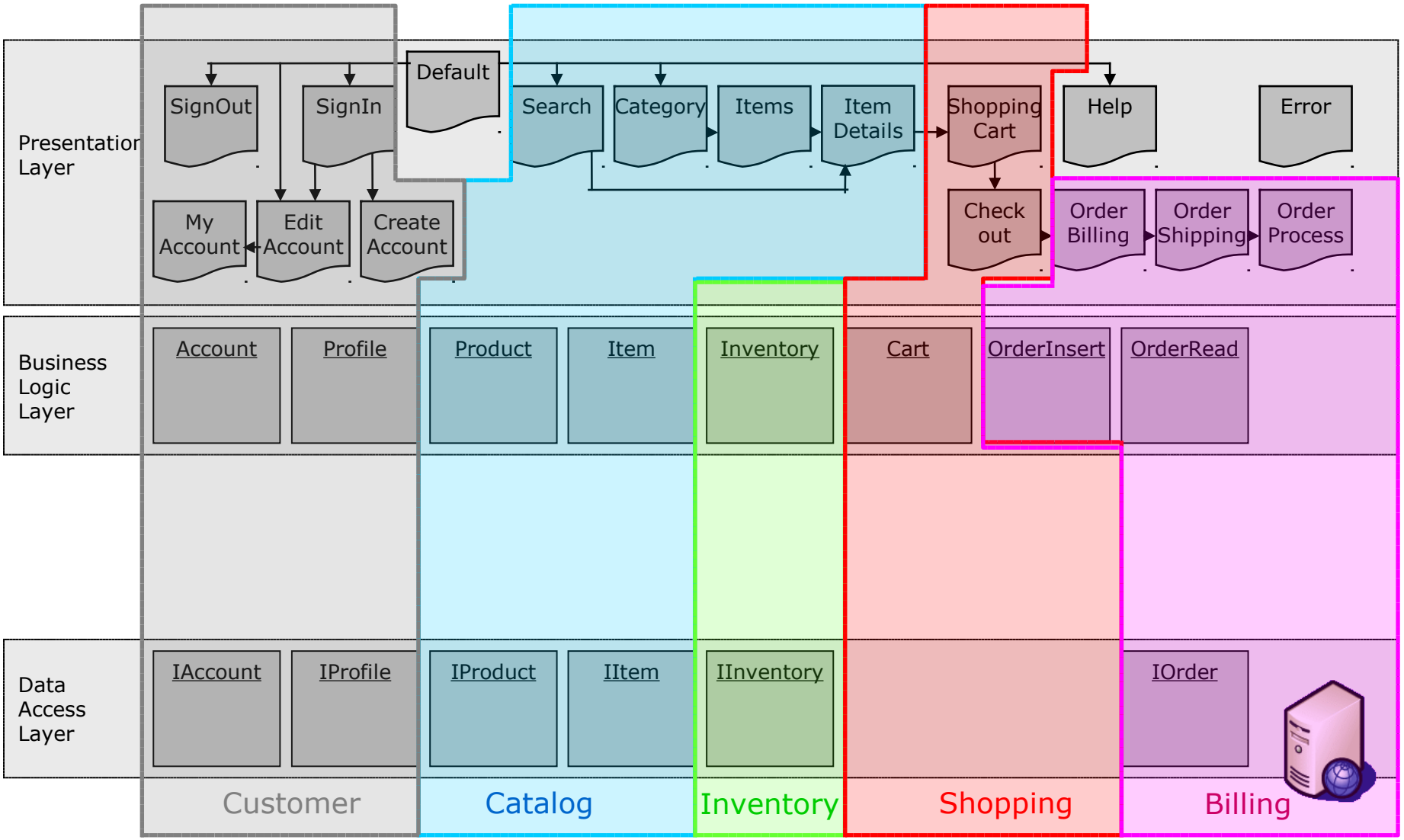
BPM par l'exemple



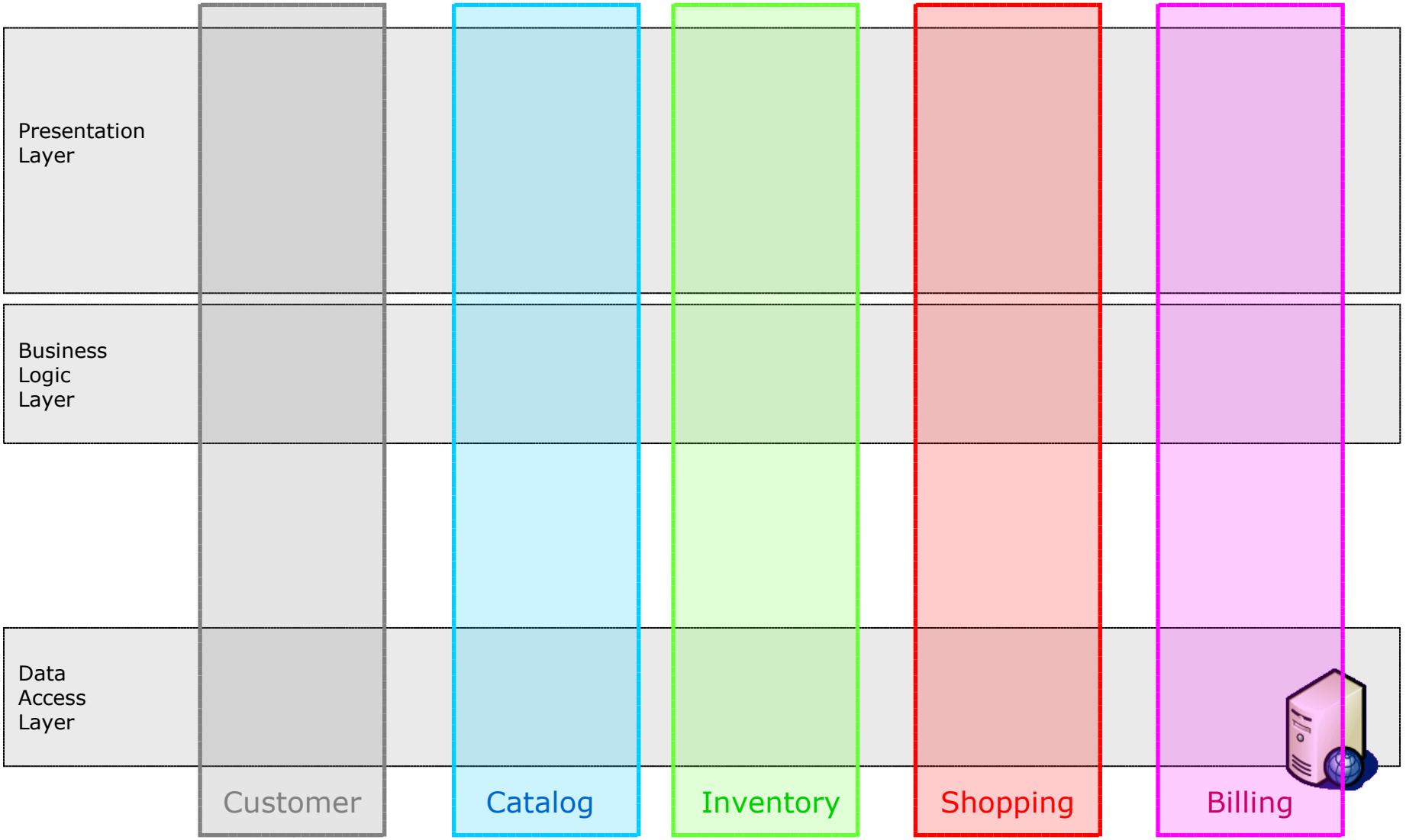
e-store : Couches



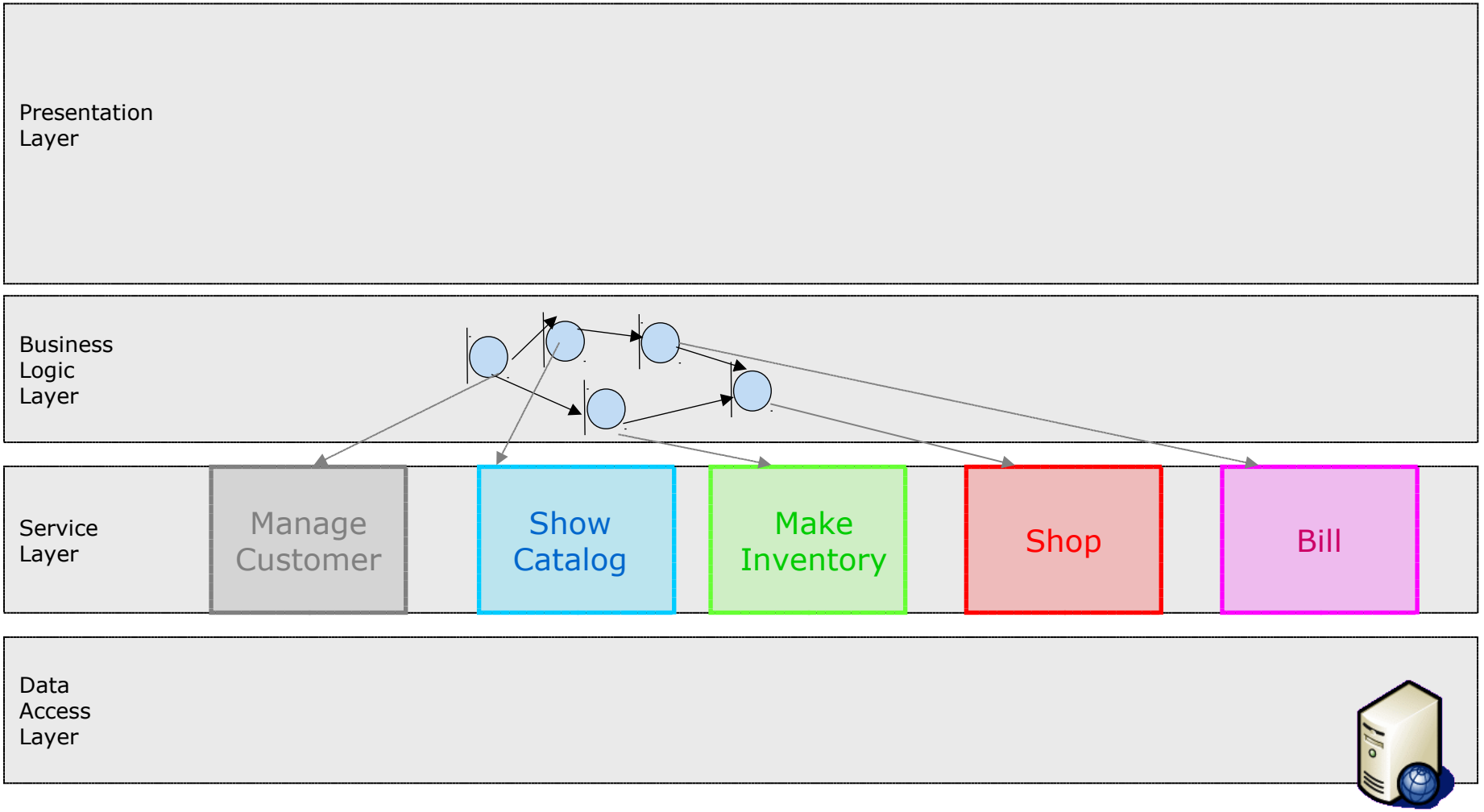
e-store : Domaines



e-store : Domaines



e-store : Services



Bénéfices métier

- Améliorer l'agilité et la flexibilité du métier
- Faciliter la gestion des processus métier

- Offrir la capacité à casser les barrières organisationnelles (silos)
- Réduire en temps le cycle de développement des produits

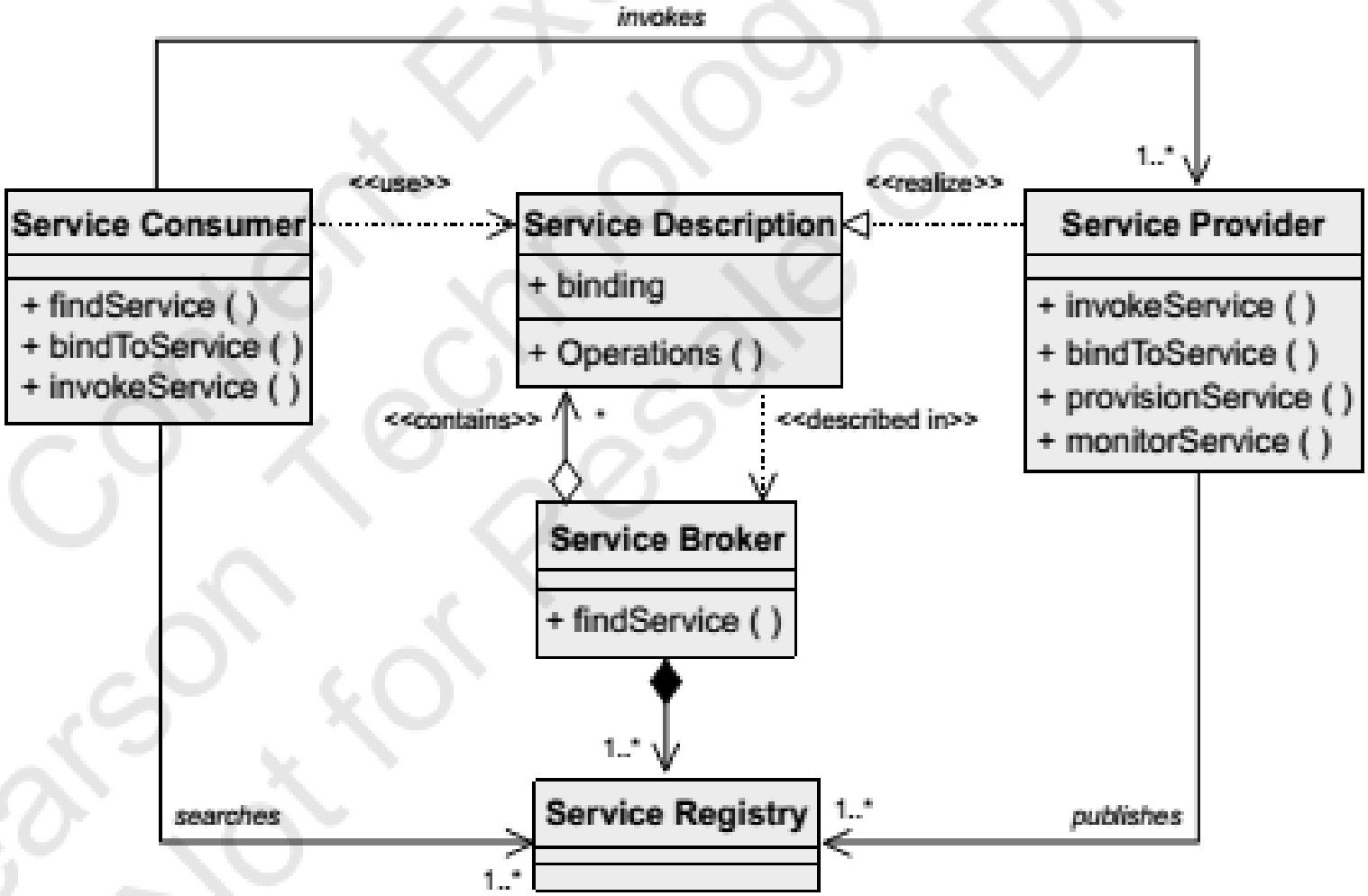
- Améliorer le retour sur investissement
- Accroître les opportunités de revenu

Bénéfices techniques

- Réduire la complexité de la solution
- Construire les services une seule fois et les utiliser fréquemment
- Garantir une intégration standardisée et le support de clients hétérogènes
- Faciliter la maintenabilité

Quels sont les éléments clé d'une architecture orientée services ?

SOA Architectural Style



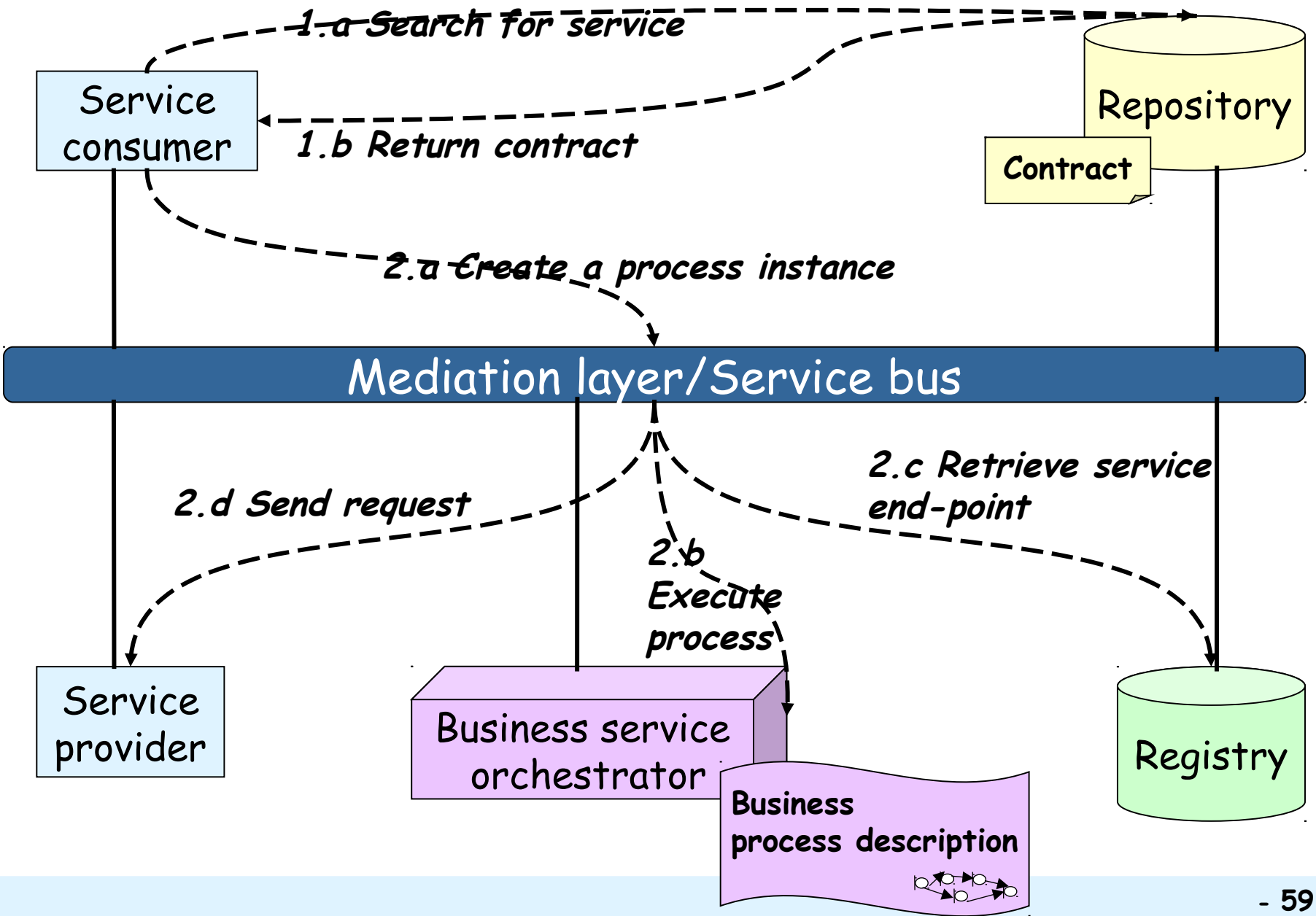
SOA Architectural Style

- Un consommateur de service appelle ou utilise un service.
- Le consommateur de service utilise la description du service:
  - Pour obtenir les informations nécessaires sur le service de fournisseur (par exemple, le service de compte) pour être consommé.

SOA Architectural Style

- La description du service fournit les informations de liaison, donc:
  - Le consommateur peut se connecter au service, et la description identifie les différentes opérations (par exemple, un compte ouvert ou fermé) disponibles auprès du service de fournisseur.
  - Un courtier peut être utilisé pour trouver le service en utilisant un registre qui contient les informations concernant le service et son emplacement.

Points clés de l'architecture



Standards de l'architecture

Les standards sont un élément clé d'une SOA, ils assurent l'interopérabilité

SOAP

W3C

Simple Object Access Protocol

Transporte

WSDL

W3C

Web Services Description Language

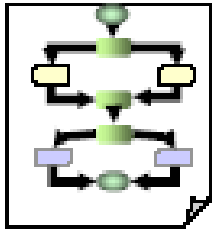
Décrit le contrat

UDDI

Microsoft, IBM, HP

Universal Description Discovery and Integration

Spec pour Repository/Registry



BPEL

Oasis

Business Process Execution Language

Décrit les processus métier

Les trois piliers des Services Web

XML, DTD, XML schéma

XML

- **Extensibilité:** pouvoir définir de nouvelles balises.
- **Structuration** : pouvoir modéliser des données d'une complexité quelconque.
- **Validation** : pouvoir vérifier la conformité d'une donnée avec un type (un modèle de structure).
- **Indépendance du média**: pouvoir formater un contenu selon des représentations diverses.
- **Interopérabilité** : Pouvoir échanger et traiter une donnée en utilisant de nombreux types de logiciels.

Caractéristiques de XML (1) : un langage de niveau méta

- XML n'est pas un langage de balises de plus (comme HTML).
- XML permet de définir de nouveaux langages de balises
- Notion de langage de niveau méta: Un langage qui permet de définir des langages.

Caractéristiques de XML (2) : un langage verbeux

Choix délibéré: XML définit tout en format caractère (lisible par un être humain).

- **Avantage :** pour le **développement** des codes, **la mise au point, l'entretien** et pour **l'interopérabilité**.
- **Inconvénient:** Les données codées en XML sont **toujours plus encombrantes** que les mêmes données codées en binaires.
  - L'espace mémoire, l'espace disque et la bande passante sont devenus **moins chers**.
  - Les **techniques de compression** sont accessibles **facilement et gratuitement**.
  - Elles sont applicables **automatiquement** dans les communications par modems ou en HTTP/1.1 et **fonctionnent bien** avec XML.

Structure d'un document XML

Un document XML est composé de trois parties:

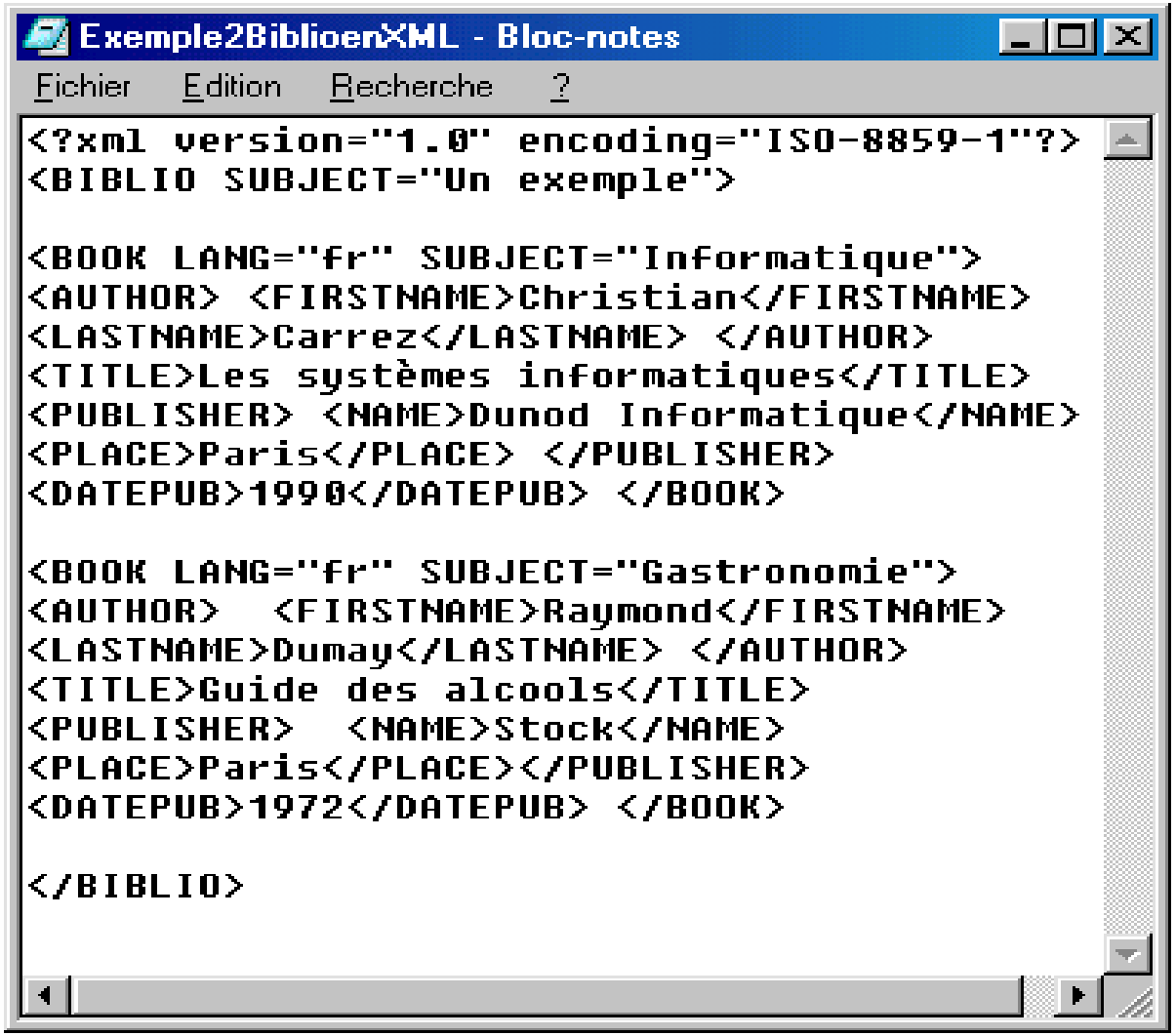
- **Un prologue** qui comporte:
  - » Des déclarations diverses
  - » Des instructions de traitement (optionnelles)
  - » Une déclaration de type du document (optionnelle)
- **Un ensemble d'éléments** organisés en arbre (les balises).
- **Des commentaires** <!-- Un commentaire -->

Corps d'un document XML

- **XML définit un cadre pour baliser n'importe quel document structuré en arbre (balises ou 'tags')**
- **Notion d'élément XML (associé à une balise)**: définit une donnée sous la forme d'une chaîne de caractères comme ayant un sens (message, de, pour, objet, texte)
- **Notion d'attribut XML (d'un élément)**: pour associer une donnée à une balise (localisation, codage).
- **Exemple simple:**

```
<message securite="secret défense">
<de>Jean</de>
<pour>Jacques</pour>
<objet>Rappel</objet>
<texte>On se voit demain à 10h</texte>
</message>
```

Exemple d'une bibliographie en XML



L'arbre associé à une bibliographie

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <BIBLIO SUBJECT="Un exemple">
  - <BOOK LANG="fr" SUBJECT="Informatique">
    - <AUTHOR>
        <FIRSTNAME>Christian</FIRSTNAME>
        <LASTNAME>Carrez</LASTNAME>
      </AUTHOR>
      <TITLE>Les systèmes informatiques</TITLE>
    - <PUBLISHER>
        <NAME>Dunod Informatique</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1990</DATEPUB>
    </BOOK>
  - <BOOK LANG="fr" SUBJECT="Gastronomie">
    - <AUTHOR>
        <FIRSTNAME>Raymond</FIRSTNAME>
        <LASTNAME>Dumay</LASTNAME>
      </AUTHOR>
      <TITLE>Guide des alcools</TITLE>
    - <PUBLISHER>
        <NAME>Stock</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1972</DATEPUB>
    </BOOK>
  </BIBLIO>
```

Documents bien formés

- **Si un document XML respecte les règles de la grammaire XML on dit qu'il est bien formé.**
- **Les règles d'un document bien formé**
  - Toute balise **ouverte** doit être **fermée**, Ex: <livre> </livre>
  - L'ensemble des balises est **correctement imbriqué**. Ex : Entrelacement mal formé   ….
  - Les valeurs d'attributs sont entre **guillemets**"
- Les **balises uniques correspondent à des documents vides** et sont notées: Ex : <IMG source="image.gif"/>
- Un document commence par une **déclaration XML**
  - <?xml version="1.0" encoding ="iso-8859-1" standalone="yes"?>

Conclusion XML

- **Multiples avantages**
  - **Un langage effectivement simple**.
  - XML rend possible l'arrivée d'une nouvelle génération **d'outils logiciels** pour des **plate-formes hétérogènes**.
    - de manipulation,
    - de transmission,
    - de visualisation de données distribuées.
- **Les inconvénients**
  - La simplicité de base conduit à **énormément de compléments de toutes natures** => apparition d'outils très nombreux qui ajoutent des détails.

Avenir de XML

- **XML devrait permettre** pour des applications (qui ne posent pas de problèmes cruciaux de performances) de supporter dans un cadre unifié:
- **La présentation des réseaux**
  - Format des données échangées par messages ou par invocations de méthodes.
- **La définition des documents.**
  - Outils de bureautique, de documentation ..
- **La définition des données.**
  - SGBD, logiciels de gestion, Échanges de Données Informatisé (EDI), …