Architecture applicative et Cartographie

Mineure SOA

Cécile Hardebolle
cecile.hardebolle@supelec.fr

Programme

8 nov.	Introduction. Enjeux, rôle de l'architecte SI *Partie n°1 du cas d'étude*	
15 nov.	Architecture et cartographie	D1.13E
22 nov.	Modèle SOA	
29 nov.	Modélisation de processus *Partie n°2 du cas d'étude*	D1.13E
6 déc.	Web Services *Partie n°3 du cas d'étude*	D1.13E
13 déc.	Cloud *Partie n°4 du cas d'étude*	
20 déc.	Exécution de processus	D1.13E
10 jan.	Compléments et ouverture. Conclusion *Partie n°5 du cas d'étude*	

Deux intervenants :

- Olivier Besnard (Solucom)
- Cécile Hardebolle (Supélec)

27 jan. — Examen : présentation de vos travaux sur **l'étude de cas « Chaus'Star »**

Au programme aujourd'hui

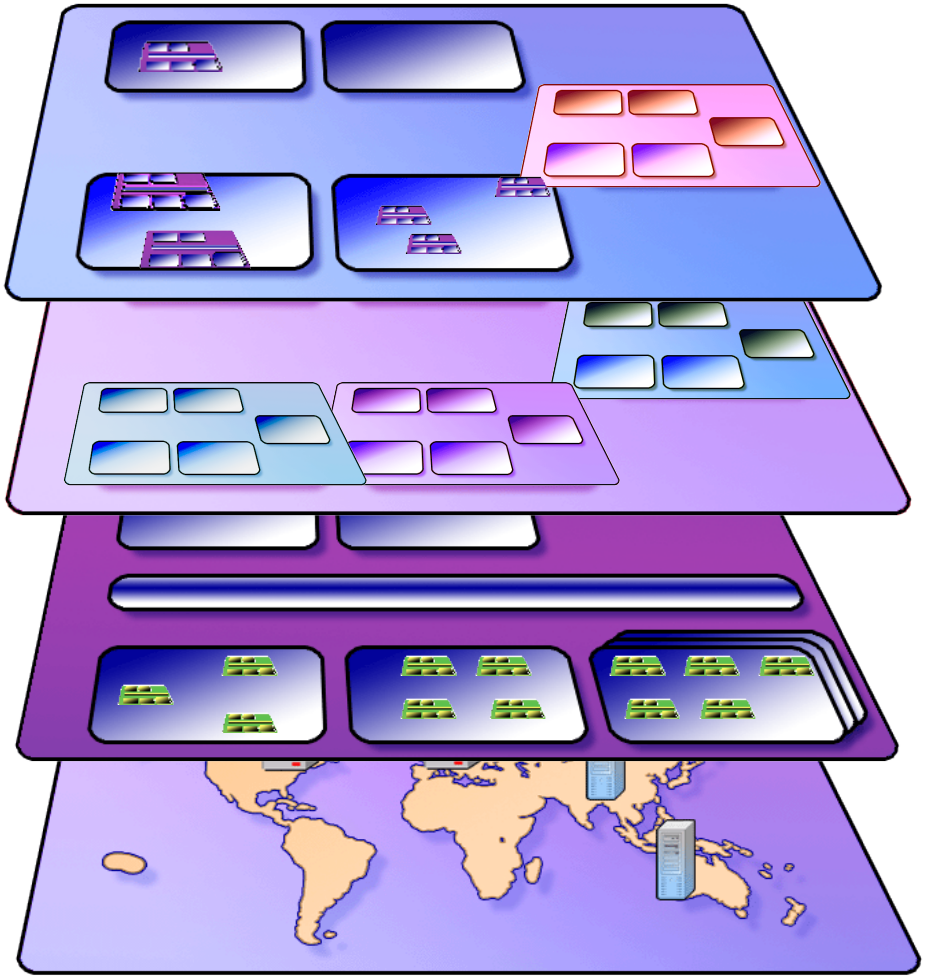
① Applications d'entreprise
- Typologie des applications
- Cartographie applicative

② Patrons d'architecture pour les applications
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux
- Typologie
- Qualification des flux

④ Architectures d'échange
- Typologie des architectures
- Solutions logicielles

Focus sur…



Vue métier
Les processus métier et leurs activités, l'organisation

Vue fonctionnelle
Les fonctions du SI supportant les processus métier

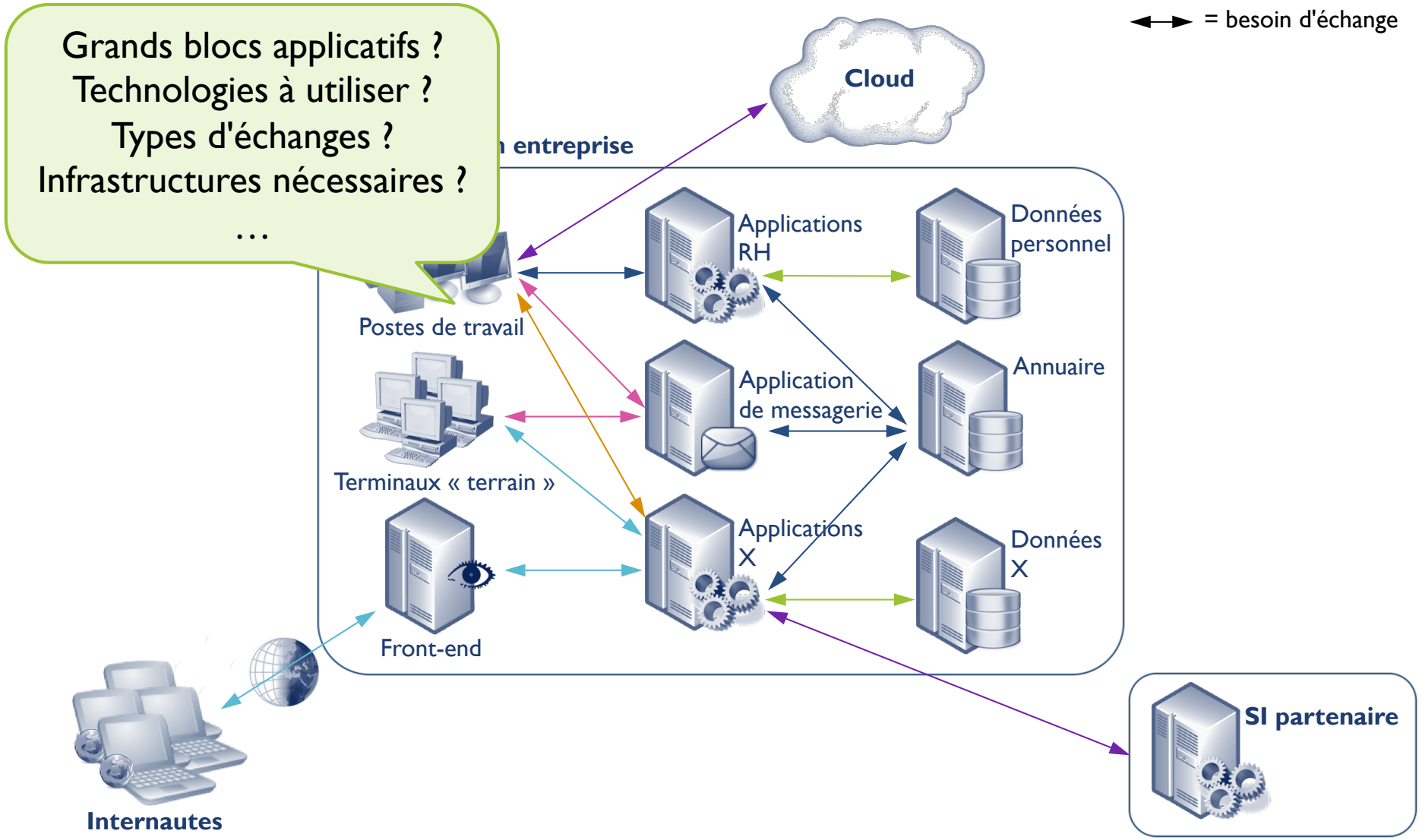
Vue applicative
Les blocs applicatifs, les messages, les données

Vue technique
Les matériels, les logiciels, les technologies

Source : Solucom

Rôle de l'architecte

Grands blocs applicatifs ?
Technologies à utiliser ?
Types d'échanges ?
Infrastructures nécessaires ?
…



Plan

① **Applications d'entreprise**
- Typologie des applications
- Cartographie applicative

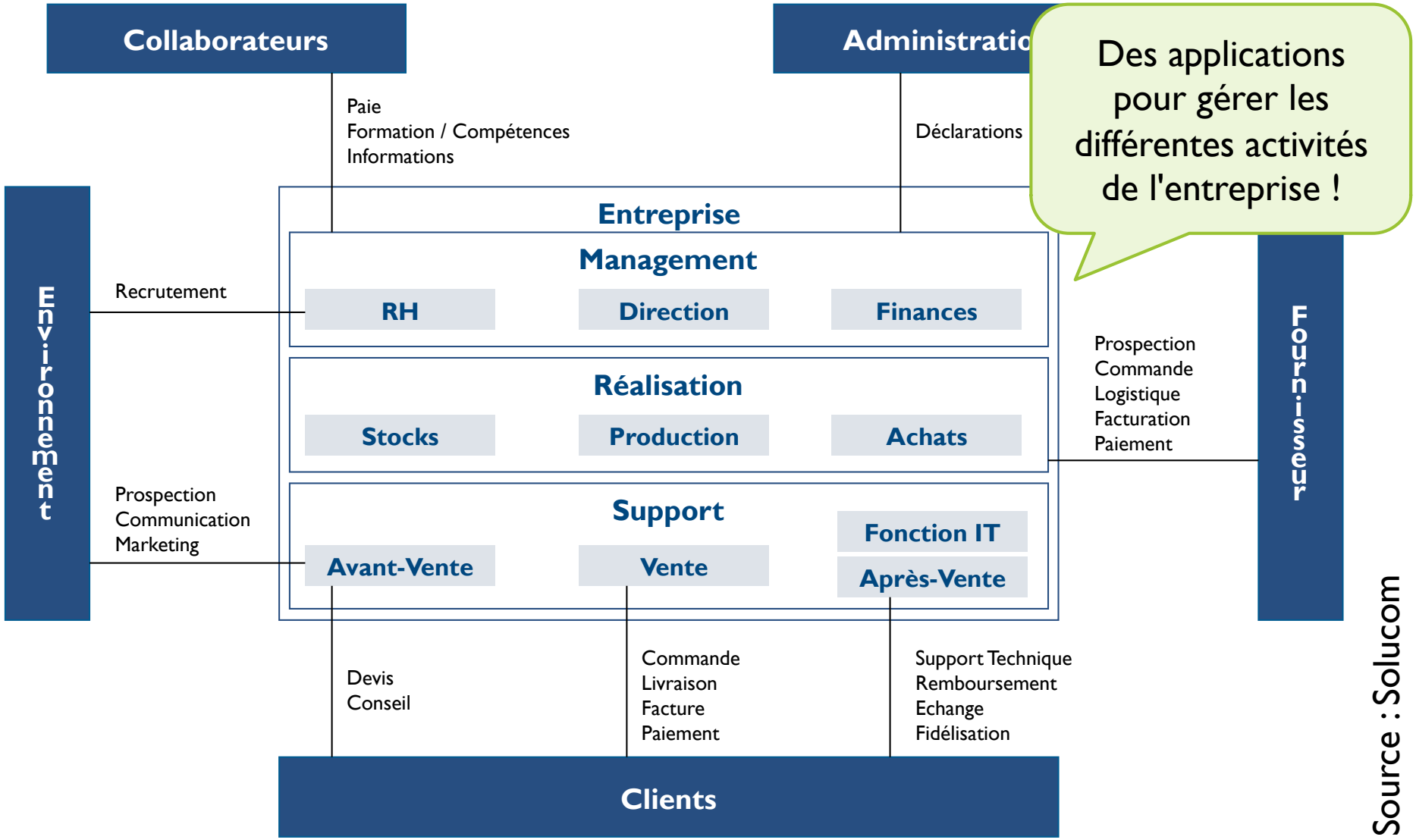
② Patrons d'architecture pour les applications
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux
- Typologie
- Qualification des flux

④ Architectures d'échange
- Typologie des architectures
- Solutions logicielles

Exemples d'applications

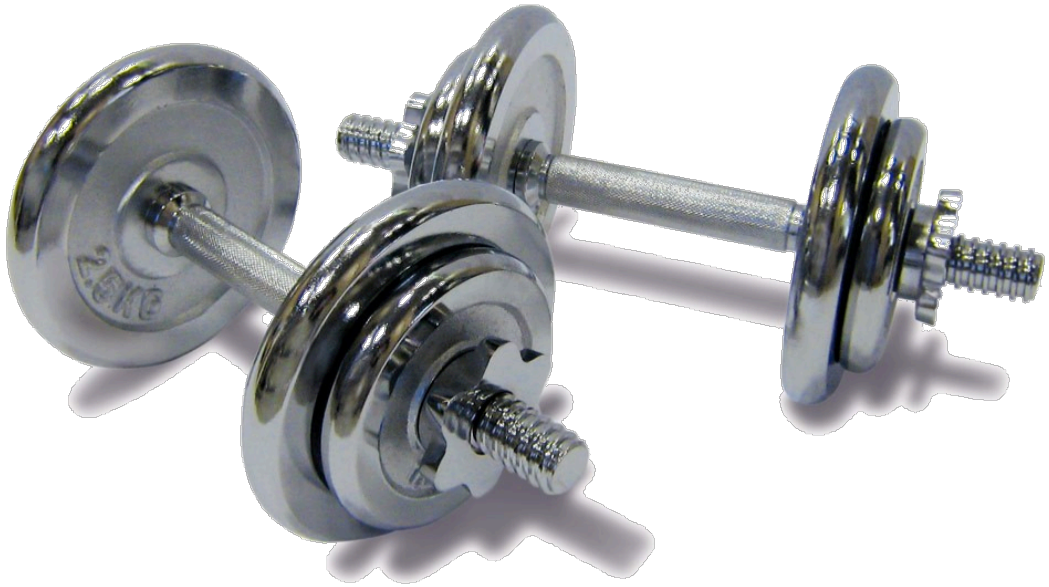
Rappel :
Organisation simplifiée de l'entreprise



Source : Solucom

Exercice 1 : cartographie applicative

☛ http://wwwdi.supelec.fr/hardebolle/SOA/ExercicesArchi



Implémentation : développer soi-même ou acheter ?

- Développements spécifiques
  - Sur serveur web/d'applications : Java EE, .NET, PHP…
  - Sur serveur de bases de données : Oracle, Access, SQL Server…
  - Sur client : Java, .NET, JavaScript, Applets…

- Progiciels et COTS (Commercial Off-The-Shelf)
  - CRM (Customer Relationship Management)
  - SCP (Supply Chain Management)
  - ERP (Enterprise Resource Planning)
  - Bureautique, messagerie
  - Applications de travail collaboratif
  - Workflows
  - …

- « Cloud »

Et pourquoi ne pas louer… ?

Plan

① Applications d'entreprise
- Typologie des applications
- Cartographie applicative

② **Patrons d'architecture pour les applications**
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux
- Typologie
- Qualification des flux

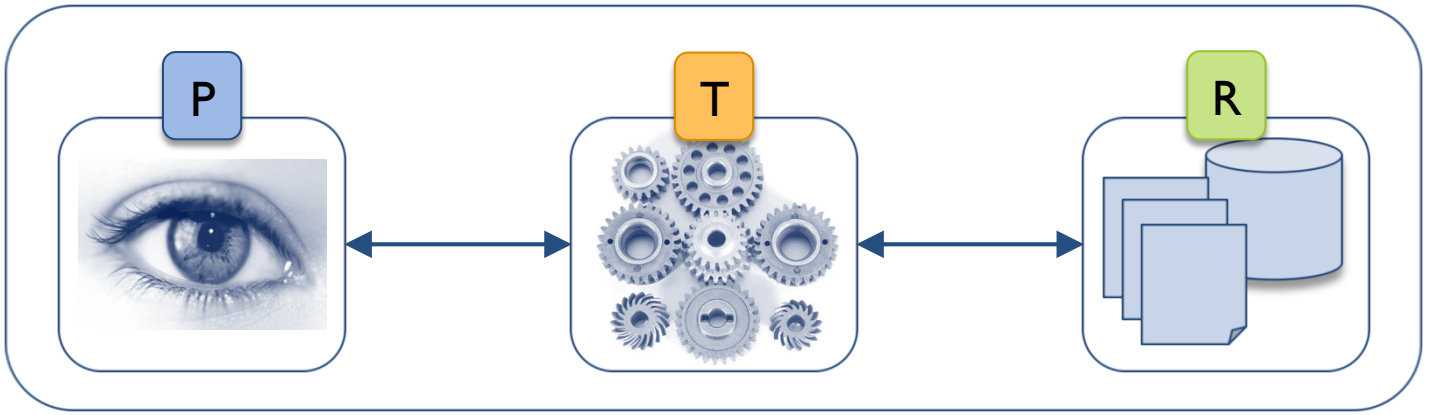
④ Architectures d'échange
- Typologie des architectures
- Solutions logicielles

Couches applicatives (rappel…)

- 3 types de responsabilités = 3 couches principales

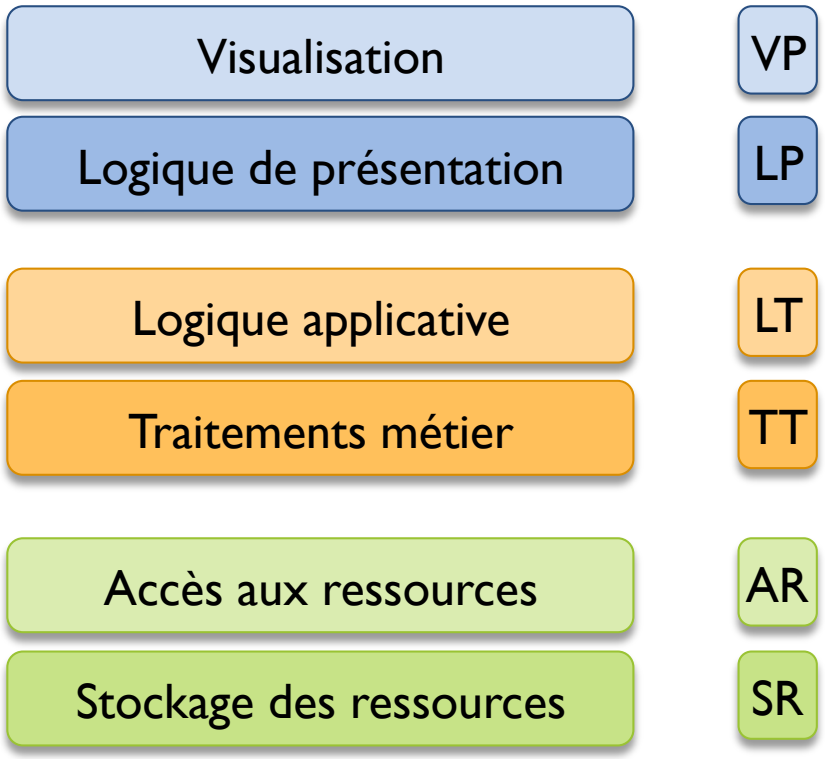
Présentation	Interaction avec l'utilisateur
Traitement	Traitements métiers, logique applicative
Ressources	Gestion des ressources, des données

- Principe de conception = séparation des responsabilités



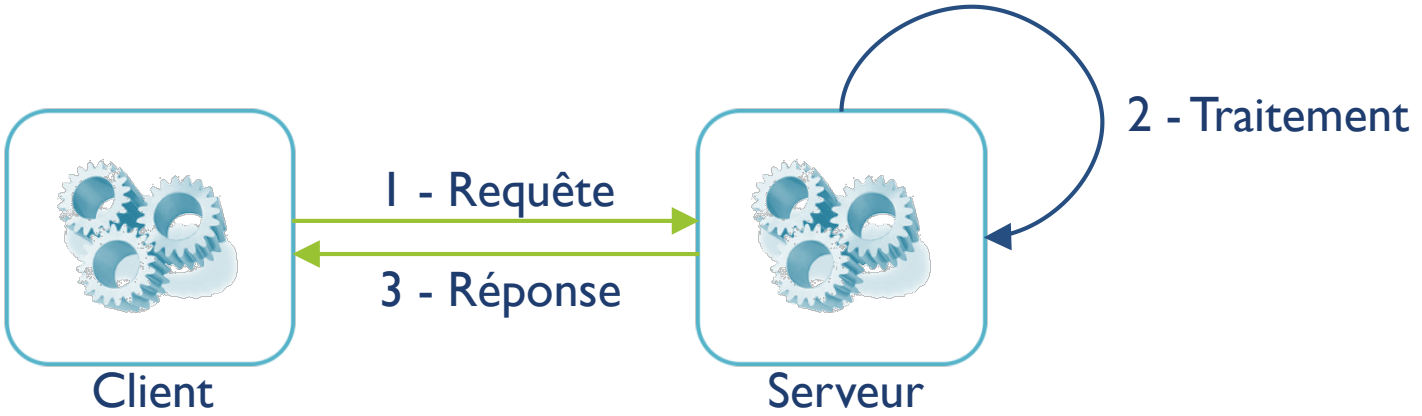
Couches applicatives détaillés

- Vue plus fine des responsabilités



Modèle Client-Serveur (rappel…)

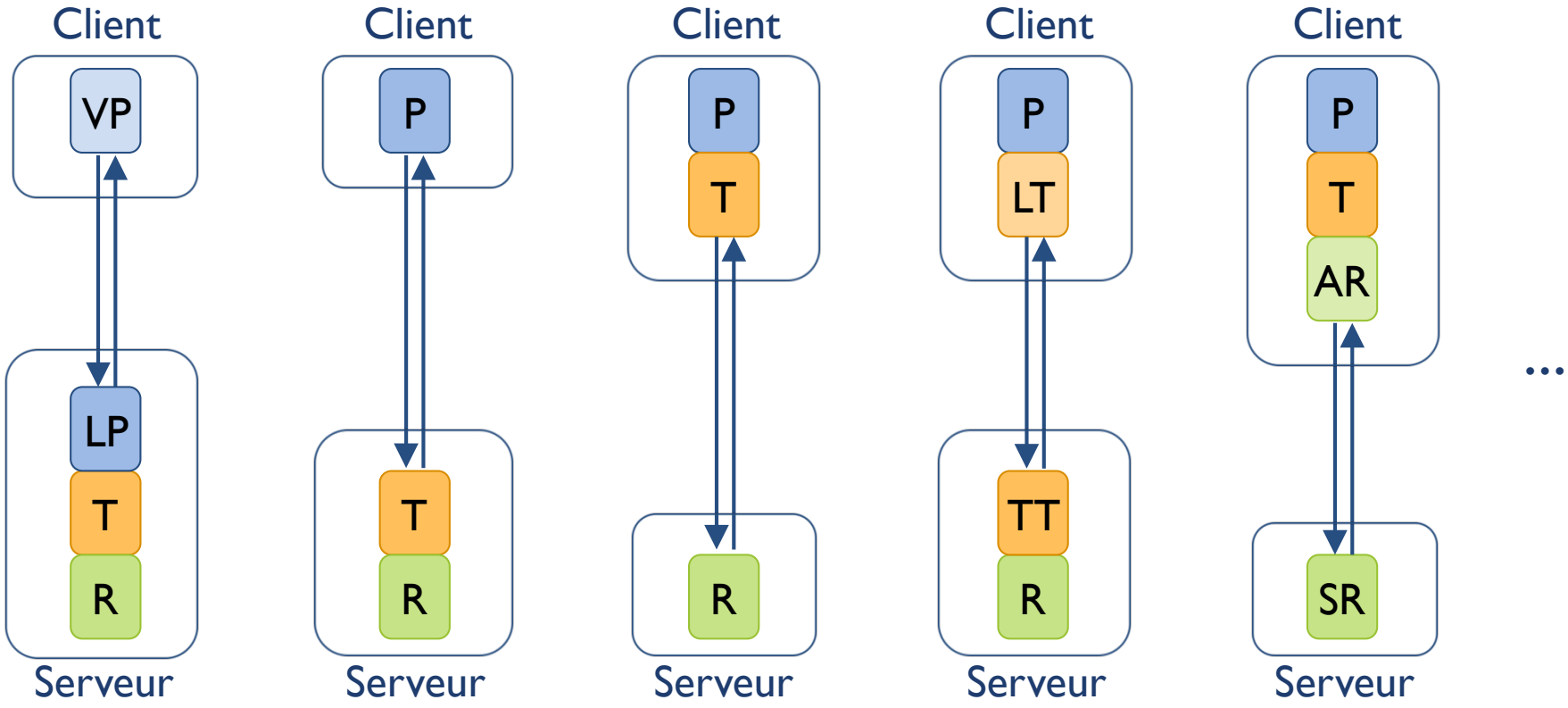
- Modèle Client-Serveur = 2 programmes + 1 protocole
  - Programme « serveur » = offre un service à des clients
  - Programme « client » = utilise un service fourni par un serveur
  - Protocole = moyen de communication



- Indépendant de la notion de « machine »
  - Client et serveur sur la même machine
  - Client et serveur sur des machines différentes

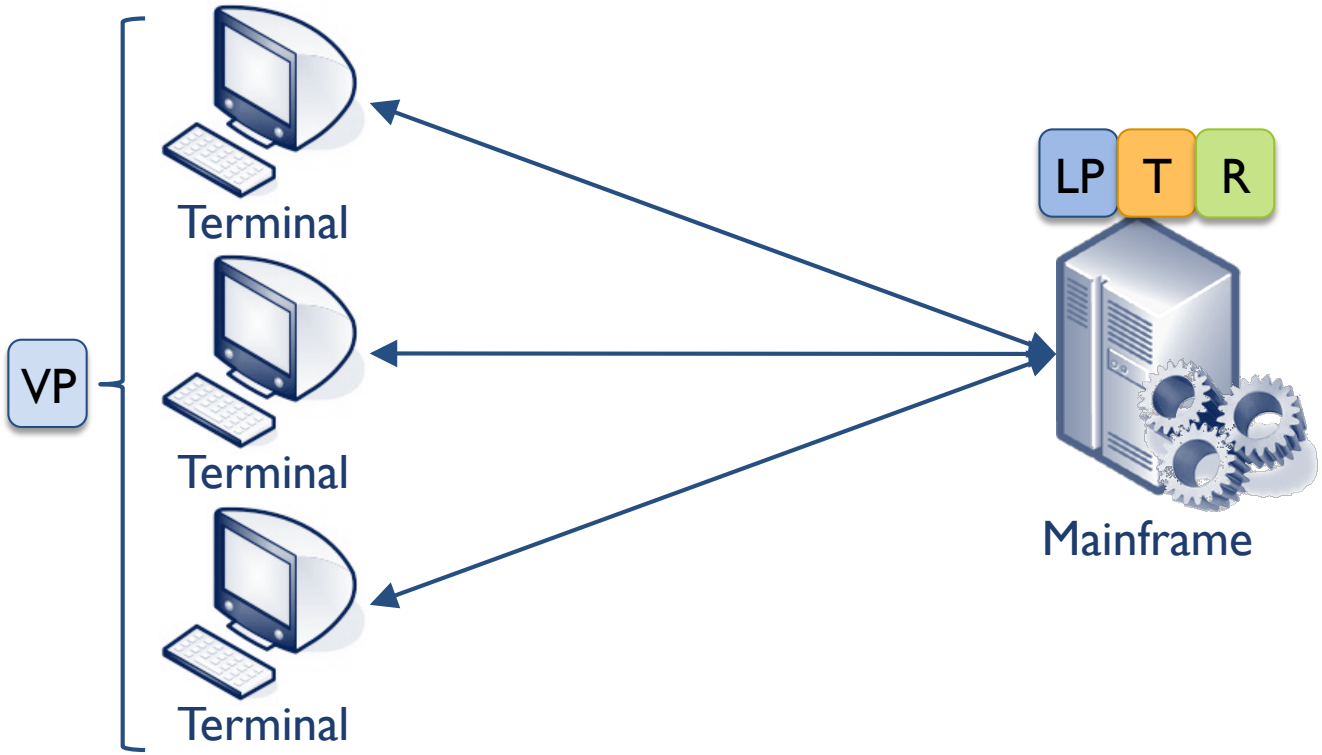
Modèle Client-Serveur
Où placer les couches applicatives ?

- Distribution des couches
- Possibilités multiples = typologies multiples (Gartner)



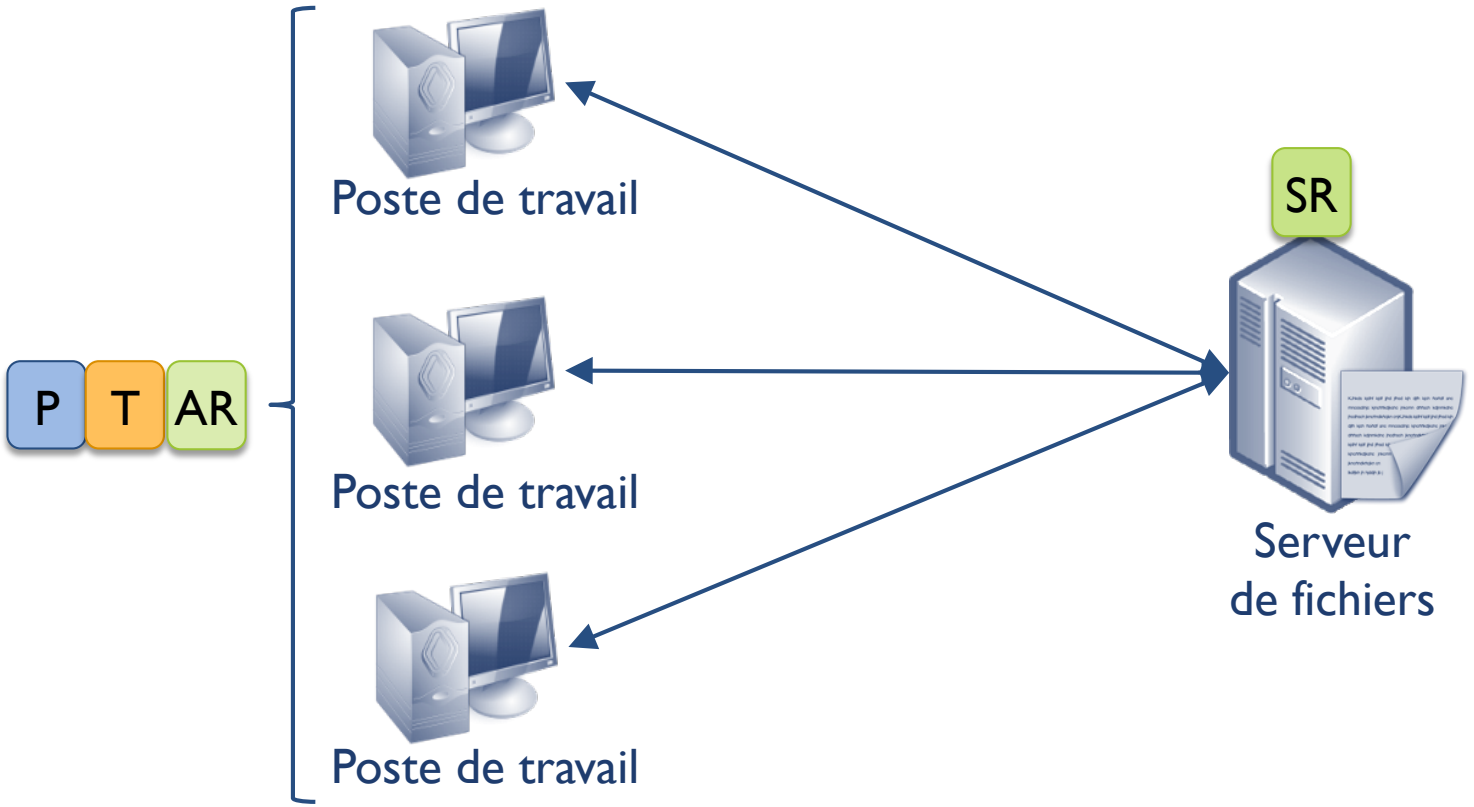
Architecture 1-tiers (mainframe)

- L'application est sur un serveur (éventuellement distant)
- Le client est une application « légère » de visualisation (client « passif »)
- Exemple :



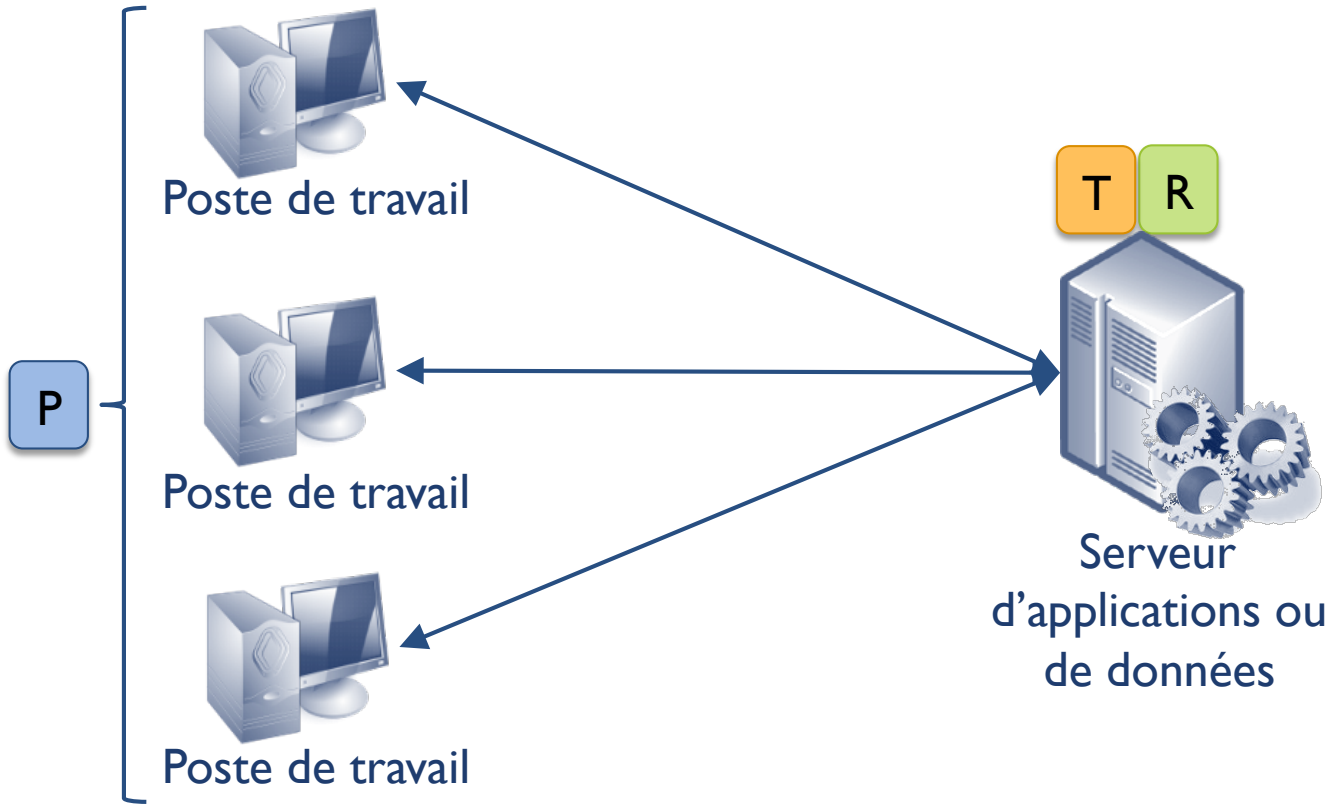
Architecture 1-tiers (client autonome)

- L'application est sur le client
- Les données sont sur un serveur (éventuellement distant)
- Exemple :



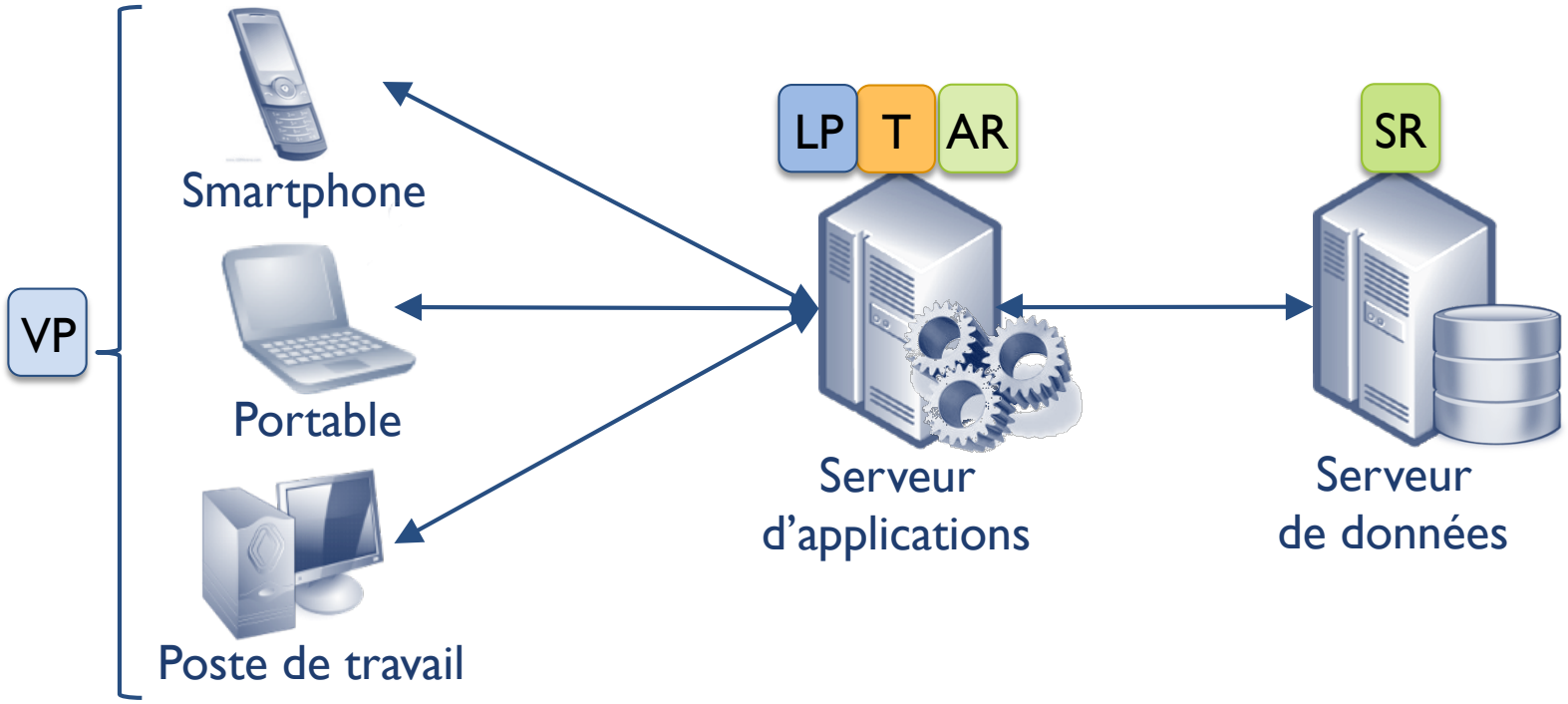
Architecture 2-tiers (client lourd)

- Le cœur de l'application est sur un serveur (éventuellement distant)
- La couche présentation (IHM) de l'application est sur le client
- Exemple :



Architecture 3-tiers (client léger)

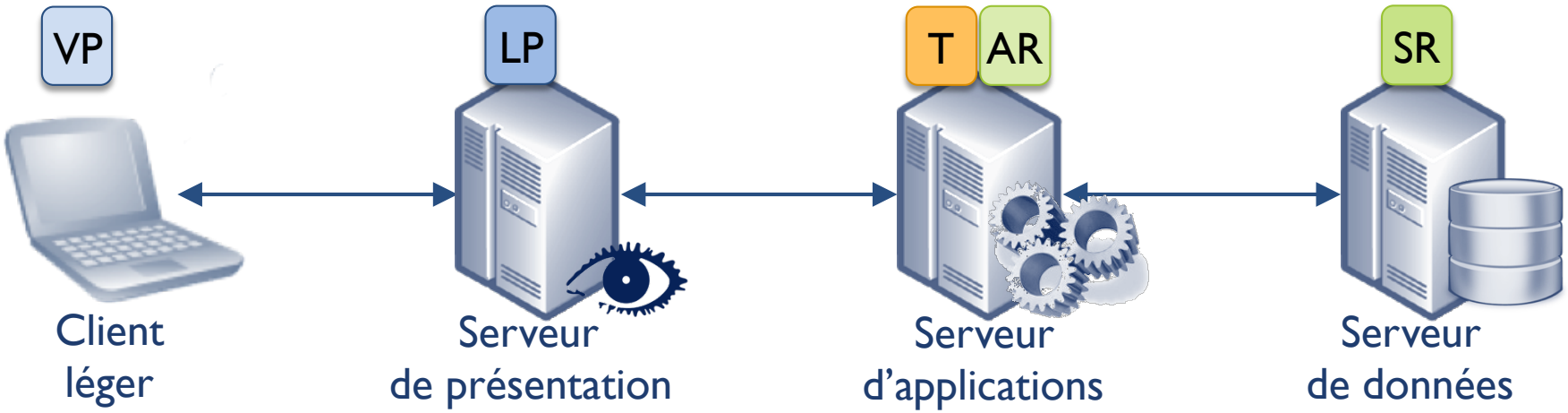
- Le cœur de l'application est sur un serveur
- Les données sont sur un autre serveur
- Le client est une application « légère » de visualisation (ex : navigateur web)
- Exemple :



Architecture n-tiers

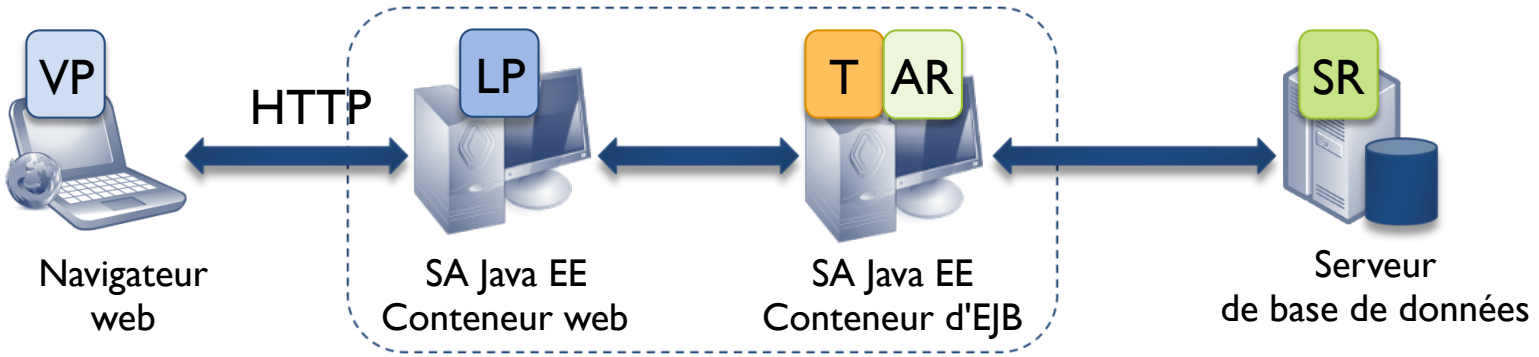
- Généralisation des modèles précédents
  (remarque : un serveur peut être un client pour un autre serveur !)
- Distribution des responsabilités en 4 ou + tiers

- Architecture type :



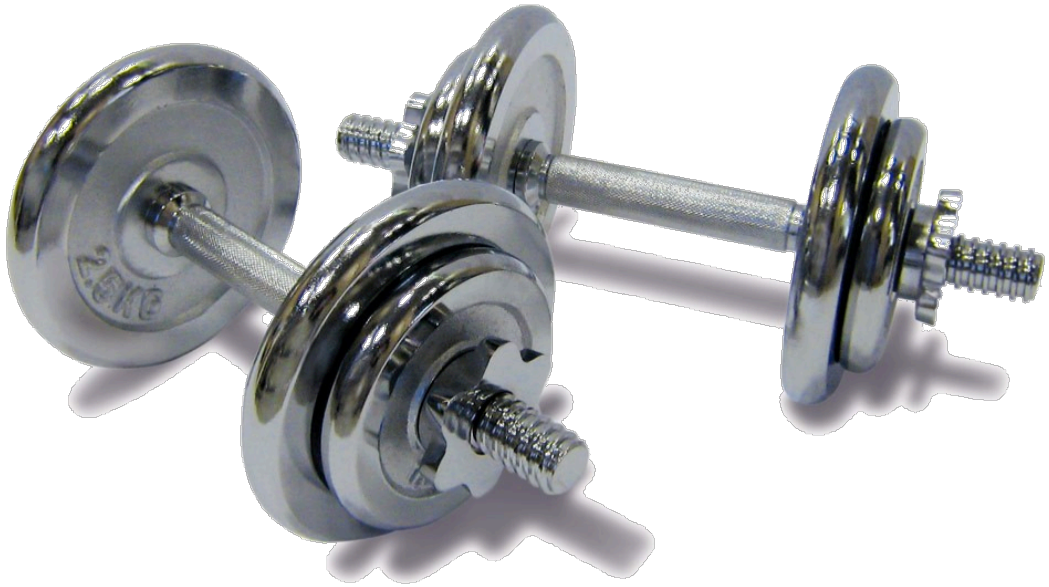
Exemple avec Java EE

- Application de gestion de catalogue de produits
- Architecture 3 ou 4 tiers :
  - Client léger
  - Serveur de présentation (pouvant être fusionné avec le serveur de traitement dans le cas de Java EE)
  - Serveur d'application
  - Serveur de données



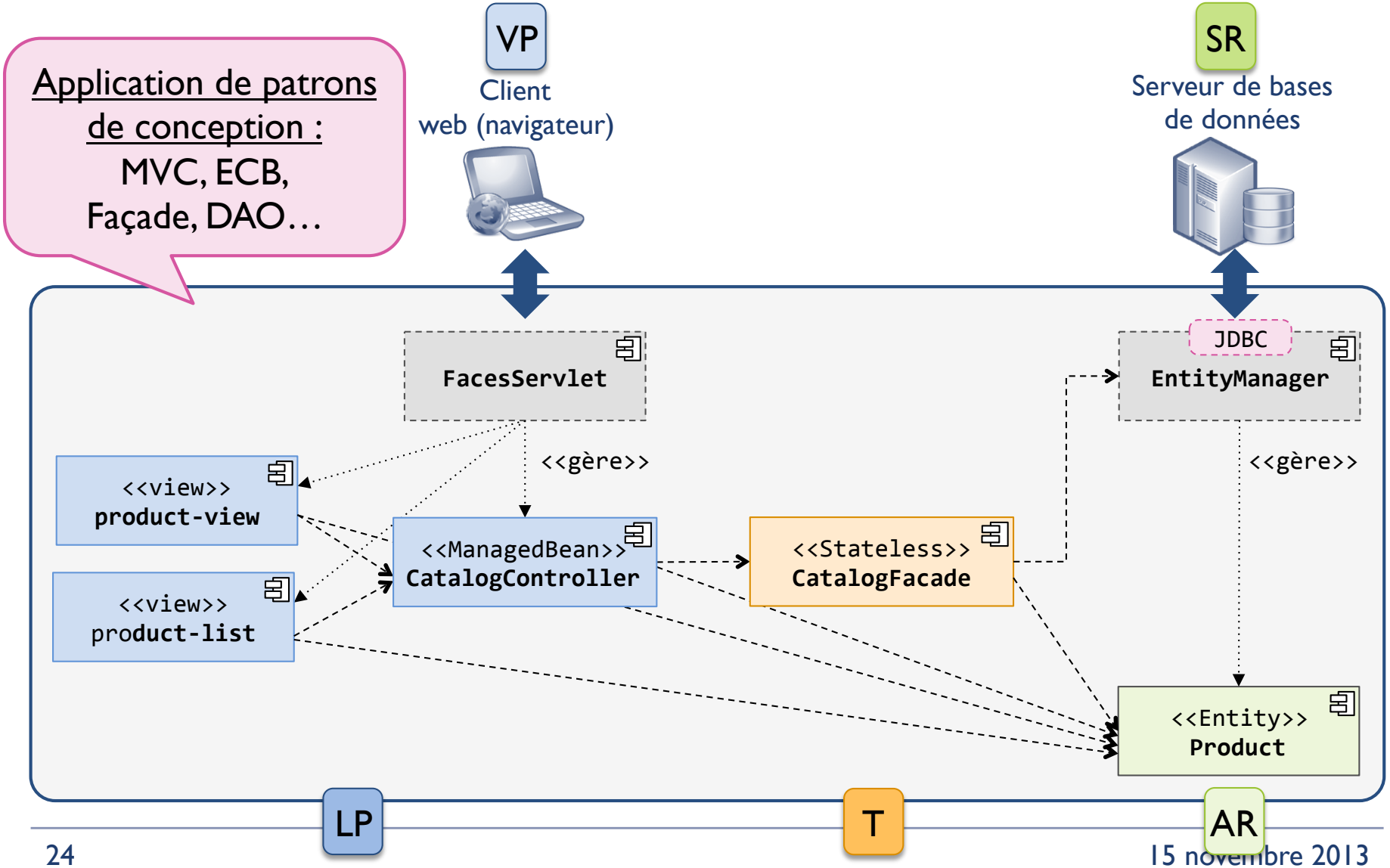
Exercice 2 : architectures n-tiers

☛ http://wwwdi.supelec.fr/hardebolle/SOA/ExercicesArchi



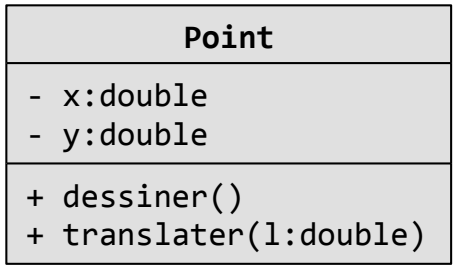
Comment « découper » une application en tiers ?

Gestion de catalogue avec Java EE

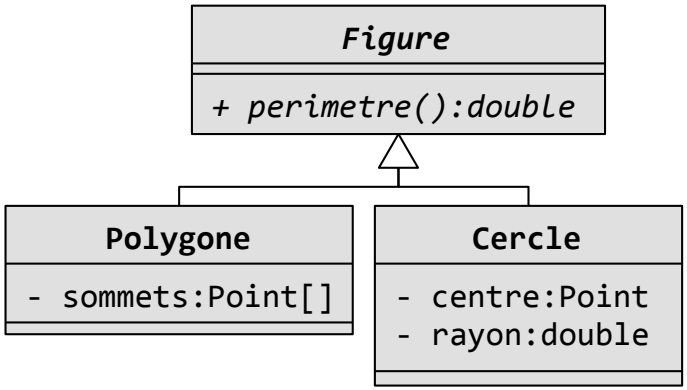


Rappel : Programmation orientée objet

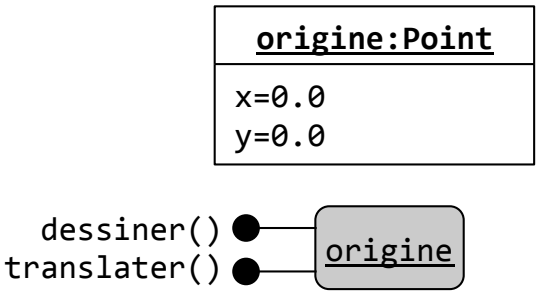
- Objet = entité possédant
  - Des caractéristiques = attributs
  - Des comportement = méthodes



- Classe / instance
  - Classe = modèle abstrait d'un objet
    - Héritage : permet d'étendre la définition d'une classe générique pour créer une classe spécifique



  - Instance = objet conforme à cette classe

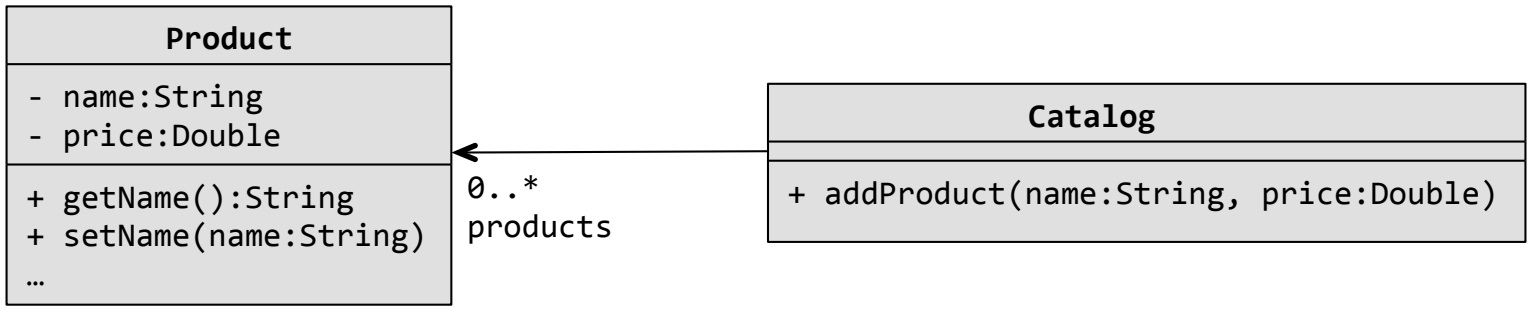


- Vue externe : boîte noire

Rappel : Classe Java
Et Plain Old Java Object (POJO)

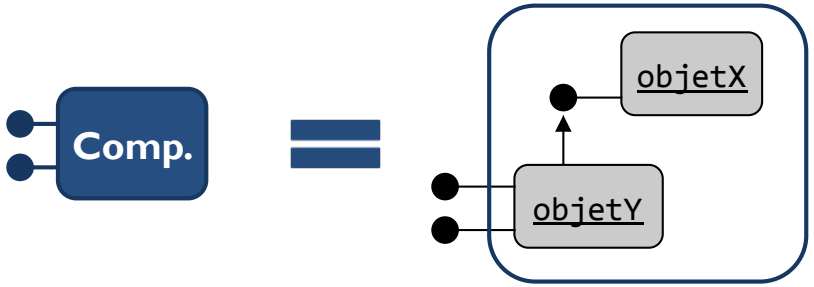
```
public class Product {

     private String name;
     private Double price;

     public Product(String n, Double p) {
          this.name = n;
          this.price = p;
     }

     public String getName() {
          return this.name;
     }
     public void setName(String name) {
          this.name = name;
     }
     …
}
```

```
public class Catalog {

     private ArrayList<Product> products;

     public Catalog () {
          this.products = new ArrayList<Product>();
     }

     …
     public void addProduct(String name,
                            Double price){
          Product p = new Product(name, price);
          this.products.add(p);
          return p;
     }
}
```



Composants

- Composant = unité logique de traitement
  - Assemblage d'objets interdépendants
  - Rend un service (fonction)
  - Vue boîte noire



- Propriétés
  - Identification : nom unique, référencé dans un annuaire
  - Indépendance : utilisable tout seul
  - Réutilisation : utilisable dans différents contextes
  - Intégration : combinable avec d'autres composants

- Technologies d'implémentation multiples

Exemple avec Java : JavaBeans

- Composant implémenté par une classe Java
  - ≈ Plain Old Java Object (POJO)

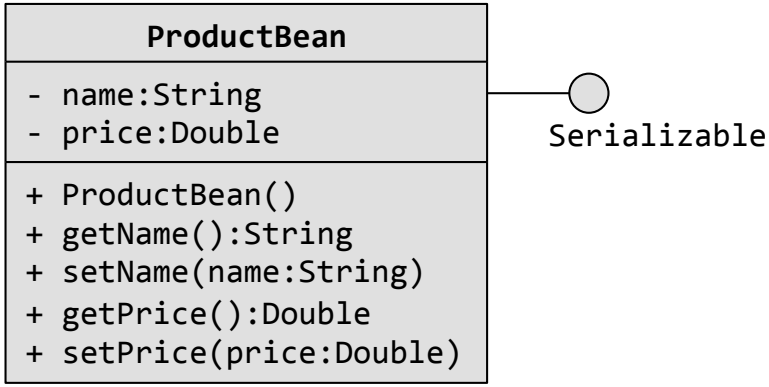
- Conventions à respecter
  - Sérialisation
  - Constructeur par défaut

  - Propriétés privées avec accesseurs
    (encapsulation et introspection)
    - `public <returntype> getPropertyname()`
    - `public void setPropertyname(parameter)`

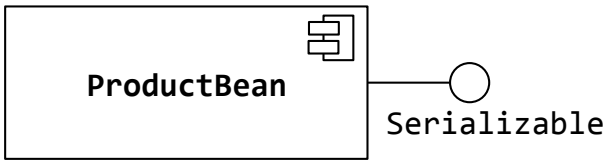
  - Méthodes d'interception d'événements
    - Utilisation d'écouteurs et génération d'événements
      - Ex : PropertyChangeListener

JavaBeans
Exemple

```
public class ProductBean implements Serializable {
     private static final long serialVersionUID = 1L;

     private String name;
     private Double price;

     public ProductBean() {
          this.name = "";
          this.price = 0.0;
     }

     public String getName() {
          return this.name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public Double getPrice() {
          return this.employed;
     }
     public void setPrice(Double price) {
          this.price = price;
     }
}
```

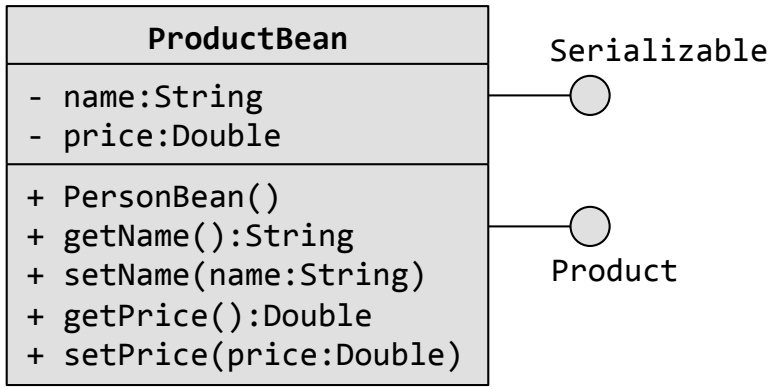


OU

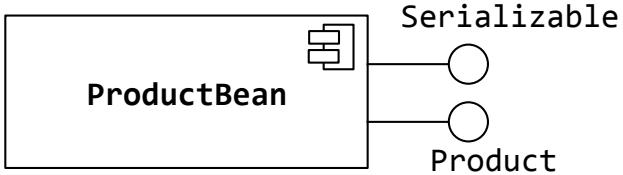


Interfaces d'un JavaBean

```
public interface Product {
  public String getName();
  public void setName(String name);
  public Double getPrice();
  public void setPrice(Double price);
}
```



ou

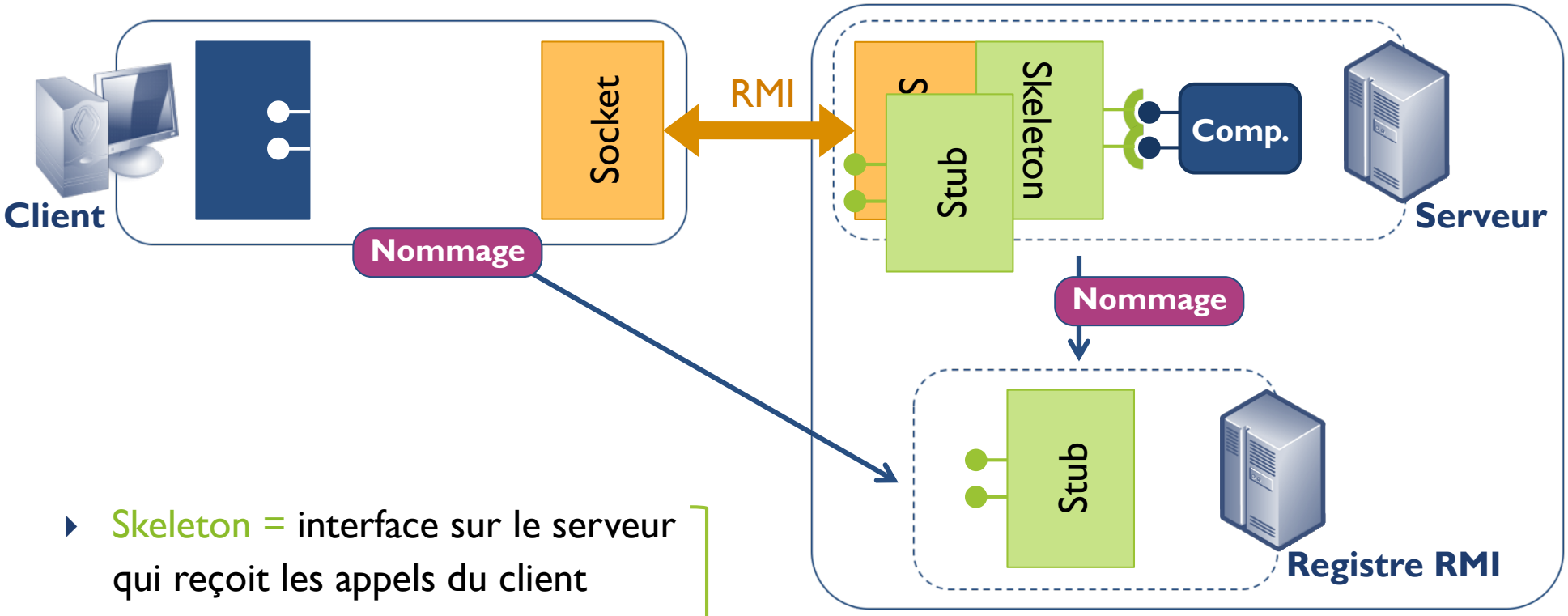



```
public class ProductBean implements Product,
                                    Serializable {
    private String name;
    private Double price;

    public ProductBean() {
        this.name = "";
        this.price = 0.0;
    }

    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Double getPrice() {
        return this.employed;
    }
    public void setPrice(Double price) {
        this.price = price;
    }
}
```

Composants distribués

- Un Client veut utiliser un composant
  qui se trouve sur un Serveur (distant)

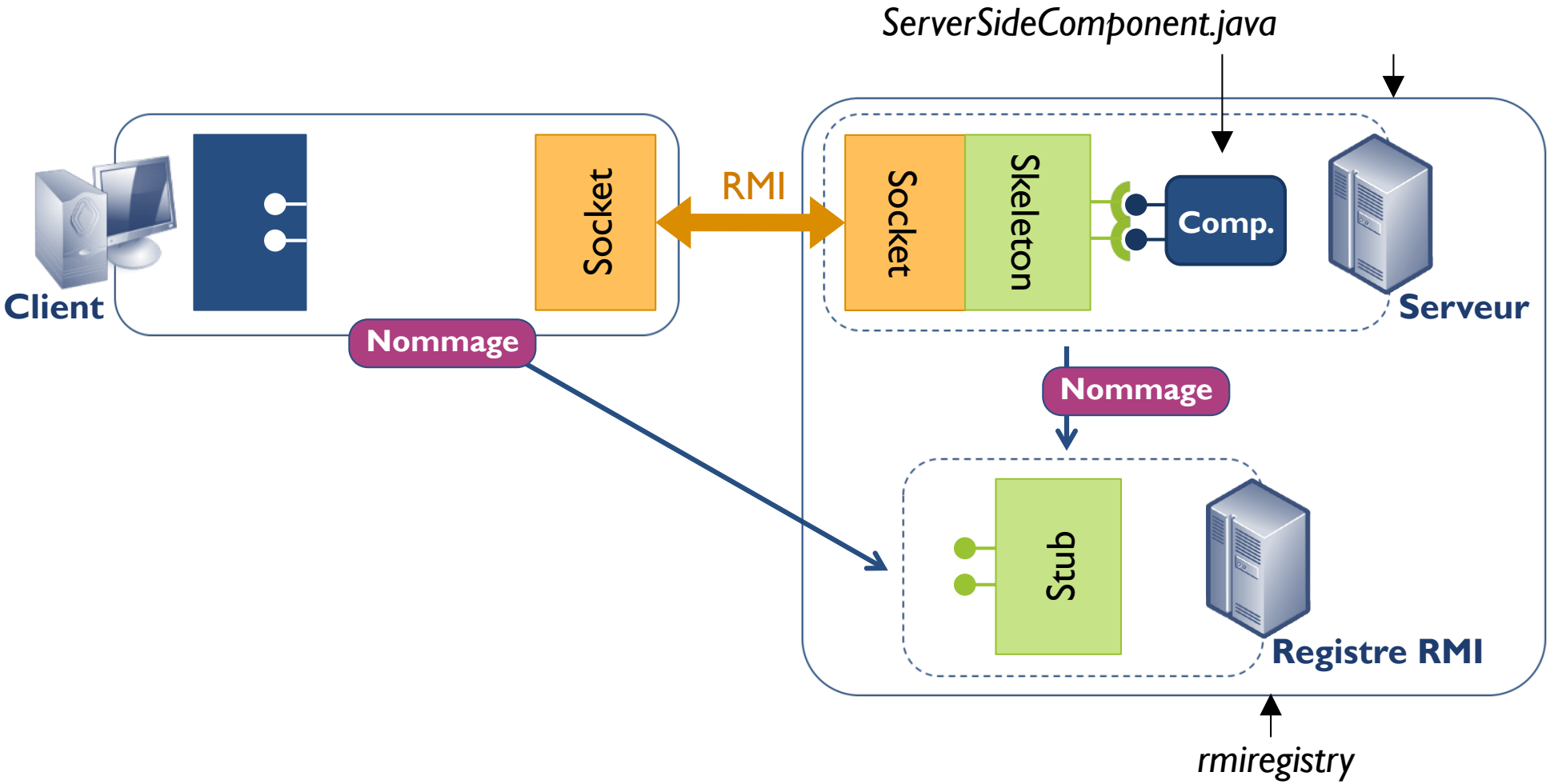
Solution Java :
RMI (Remote Method Invocation)



- Skeleton = interface sur le serveur qui reçoit les appels du client
- Stub = interface sur le client qui envoie les appels au serveur

dérivés du composant

Exemple avec RMI



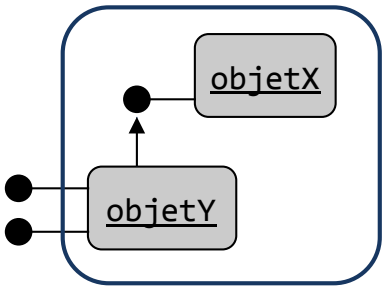
Exemple avec RMI

- Interface du composant (vue extérieure « boîte noire ») :

```
public interface ServerSideComponent extends Remote {
     public String getName() throws RemoteException;
}
```

- Implémentation du composant (« intérieur de la boîte ») :

```
public class ServerSideComponentImpl implements
ServerSideComponent,Serializable {
    private static final long serialVersionUID = 1L;

    public ServerSideComponentImpl() {
        super();
    }

    public String getName() throws RemoteException {
        return "Robert";
    }
}
```



Exemple avec RMI

- Application mettant à disposition le composant :

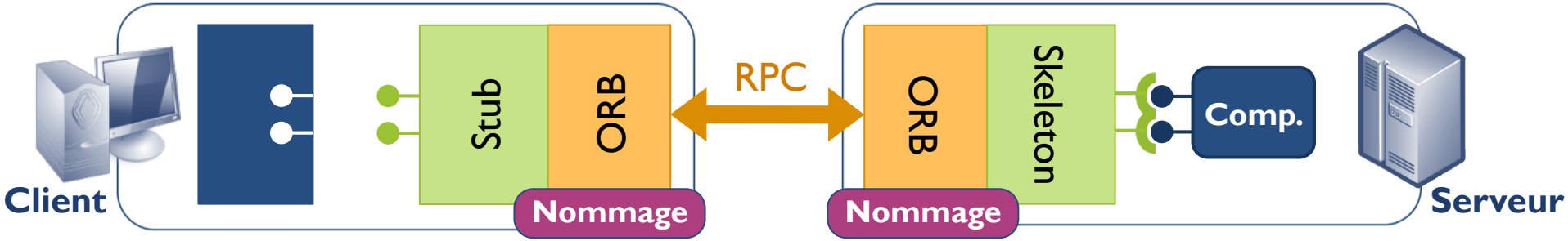
```
ServerSideComponent comp = new ServerSideComponentImpl();
ServerSideComponent stub =
     (ServerSideComponent) UnicastRemoteObject.exportObject(comp,0);
Registry registry = LocateRegistry.getRegistry("160.228.100.159");
registry.rebind("Component", stub);
```

- Application utilisant le composant :

```
Registry registry = LocateRegistry.getRegistry("160.228.100.159");
ServerSideComponent stub =
     (ServerSideComponent) registry.lookup("Component");
System.out.println("Stub obtained from registry : "
     +stub.toString());
System.out.println("Client result : "+stub.getName());
```

Solution CORBA

- Un Client veut utiliser un composant
  qui se trouve sur un Serveur (distant)



- Object Request Broker (ORB) = « bus logiciel » qui permet au client de rechercher le composant sur le serveur et de communiquer avec lui

- Skeleton = interface sur le serveur qui reçoit les appels du client
- Stub = interface sur le client qui envoie les appels au serveur

] dérivés du composant

- Remote Procedure Call (RPC) = appel de procédure distant

Problématiques

- Applications n-tiers à base de composants =
  composants distribués avec responsabilités distribuées

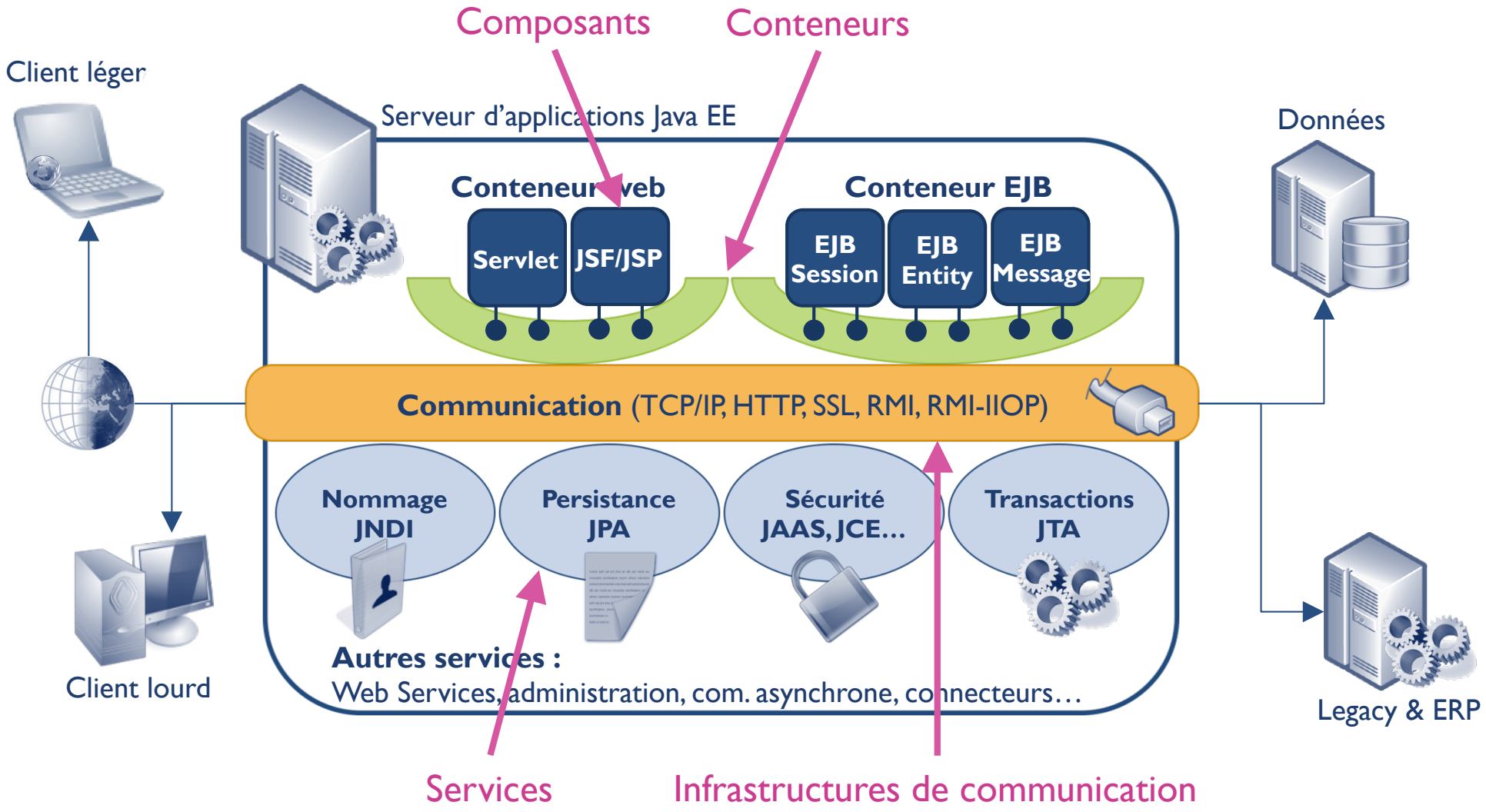
- Problématiques :
  - Complexité
    - Conception des composants et des applications
    - Développement des composants et des applications
    - Gestion des aspects transverses :
      sécurité, disponibilité, communication, persistance, transactions…
  - Interopérabilité des composants
  - Administration des composants et des applications
  - Sécurisation de bout en bout des applications
  - …

Serveurs d'application & frameworks de développement

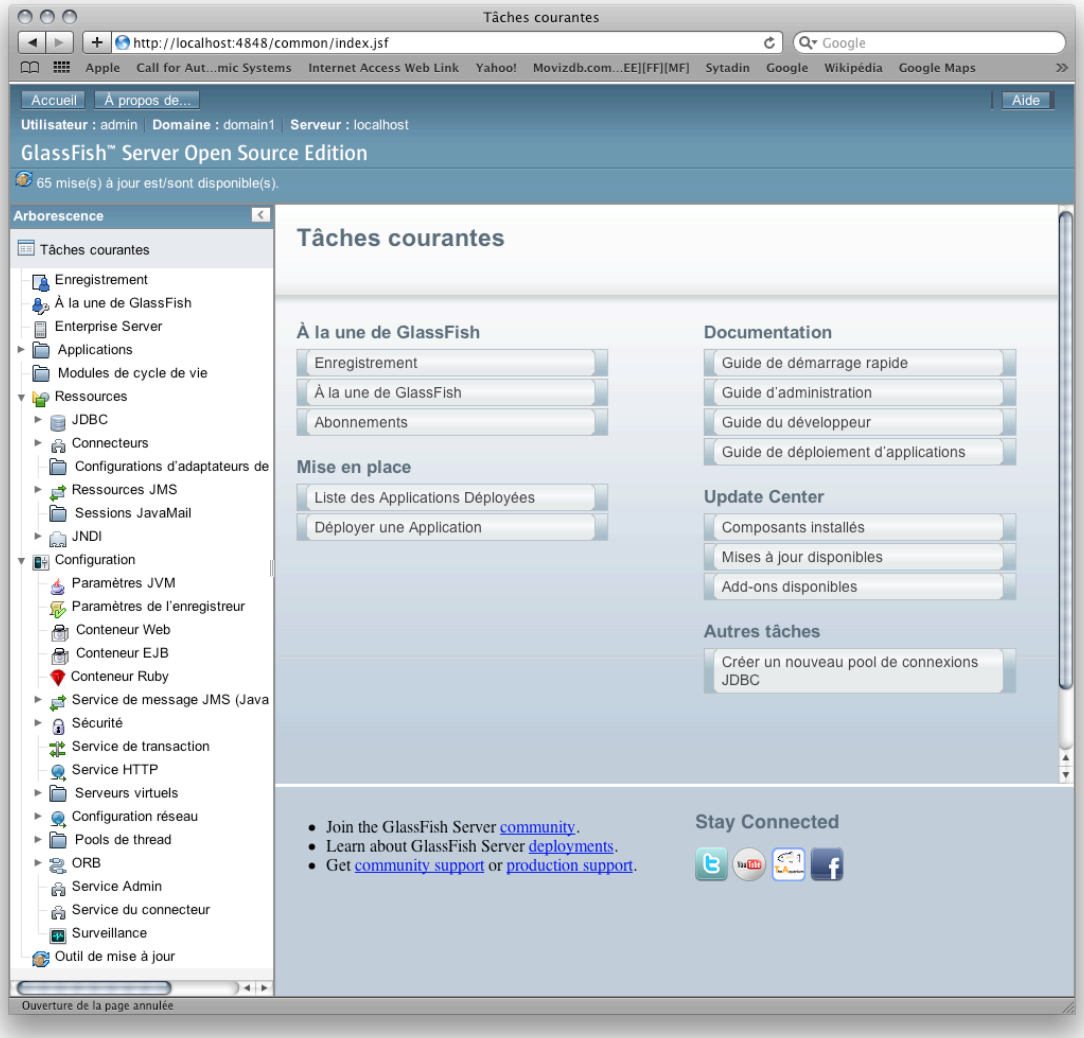
- Serveur d'Applications (SA) =
  conteneur et fournisseur de services pour des composants et des applications
  - Gestion du cycle de vie des applications et des composants
  - Administration des applications et des composants
  - Allocation de ressources
    - Processeur, mémoire, réseau, composants logiciels externes…
  - Support pour les aspects transverses
    - Sécurité, gestion des transaction, accès réseau…
  - Support pour l'interopérabilité

- Frameworks de développement =
  - Cadres pour la conception et le développement de composants (déployés sur SA) et d'applications à base de composants

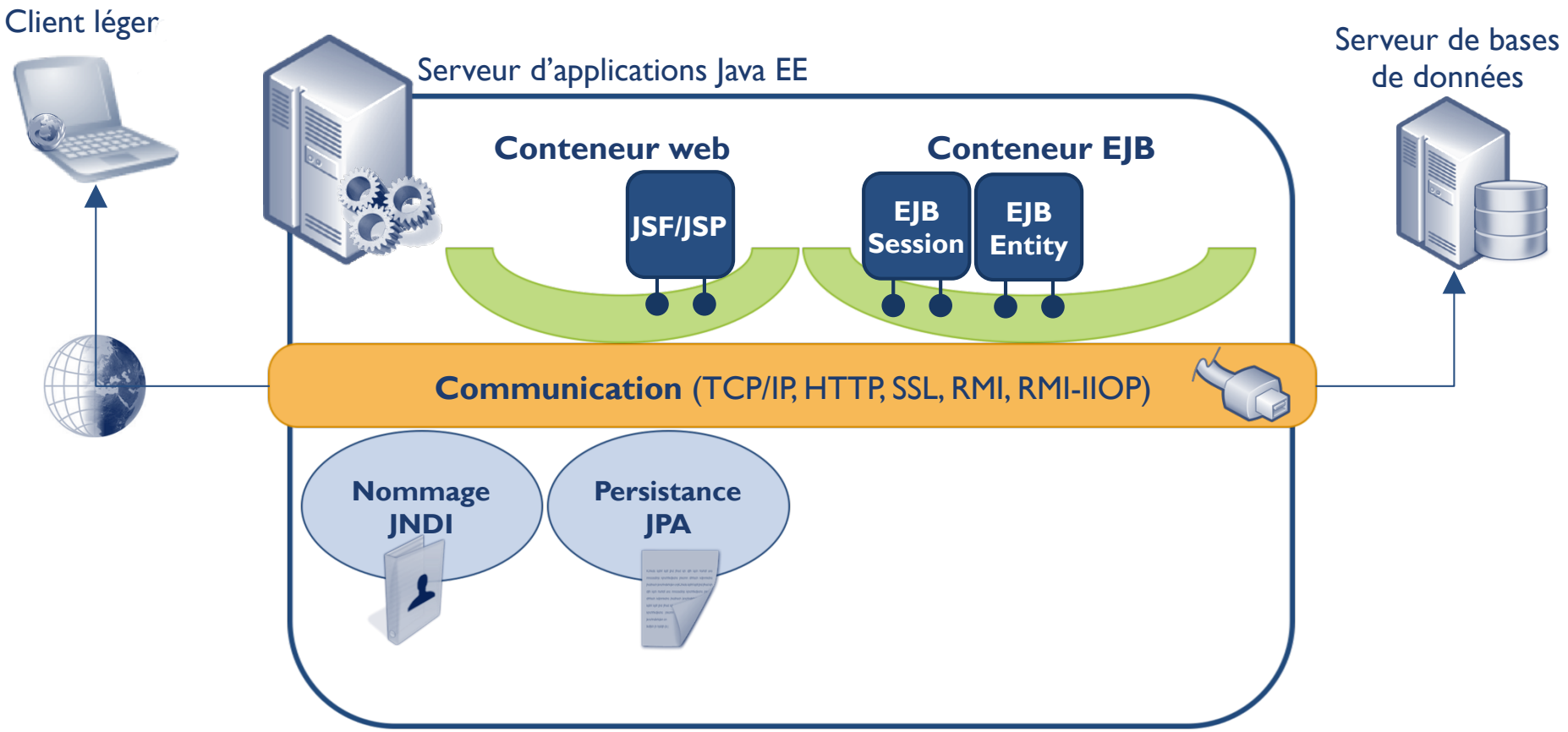
Exemple : SA + framework Java EE



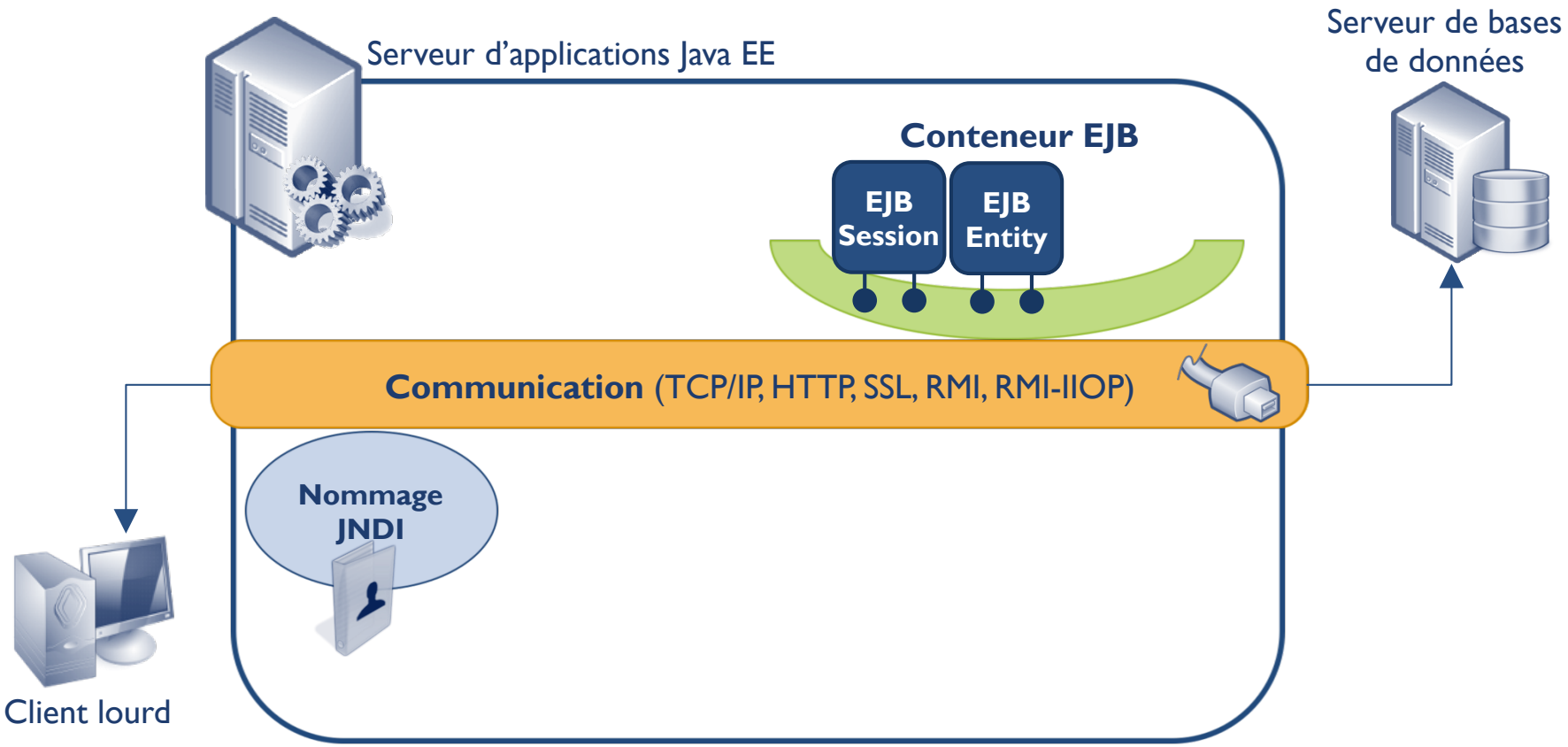
Un serveur d'applications Java EE : GlassFish



Rappel de l'exemple : Gestion de catalogue 3/4-tiers



Gestion de catalogue, une autre architecture possible



Périmètre des infrastructures logicielles

- Infrastructure logicielle = logiciel rendant des services aux applications
  - *Exemples de services : communication, administration, exécution, sécurité…*

- Interface entre les applications et l'architecture matérielle
  ☛ « en dessous » des applications

- Exemples :
  - Serveur web
  - Serveur de bases de données
  - Annuaire
  - Serveur de fichiers
  - ...
  - Serveur d'applications
  - Middleware (solution d'intégration)

Modélisation :
vue applicative
ou vue technique ?

Plan

① Applications d'entreprise
- Typologie des applications
- Cartographie applicative

② Patrons d'architecture pour les applications
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ **Flux**
- Typologie
- Qualification des flux

④ Architectures d'échange
- Typologie des architectures
- Solutions logicielles

Echanges d'informations ?

- Flux = données qui passent d'un point A à un point B
  - D'une application à une autre,
  - D'un module applicatif à un autre
  - D'un utilisateur à un autre
  - D'une base de données à une autre
  - D'une entreprise à une autre
  - …
- Exemples de flux
  - Transfert de fichier
  - Partage de fichier
  - Appel de procédure distant
  - Requête sur une base de données
  - …

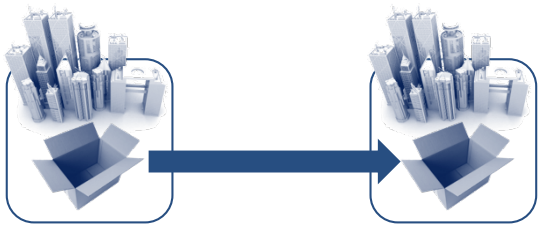
Périmètre

- Flux privé = intra-application (entre composants)

- Flux public = inter-applications
  - Bonne gestion des flux publics = flexibilité !

- Flux AtoA = flux inter-applications sur un périmètre intra-entreprise

- Flux BtoB = flux inter-applications sur un périmètre inter-entreprises
  - *Exemple : envoi d'une commande de pièce à un fournisseur*



Granularité / Fréquence

« Evénementiels »

- Flux unitaire = données transmises une à une

- Flux au fil de l'eau = données transmises dès qu'elles sont disponibles

Exemple : transmission des commandes à la plate-forme logistique au fur et à mesure de leur validation

« Batch »

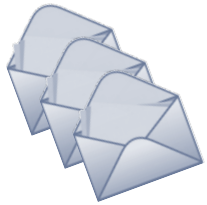
- Flux de masse = données regroupées en lots

- Flux cadencés = données transmises à des moments prédéterminés

Exemple : transmission chaque soir des données concernant l'ensemble des ventes de la journée pour stockage dans l'entrepôt de données

Exemples de flux

- Souvent pour les flux unitaires au fil de l'eau (« événementiels ») :
  - Appels distants entre composants (CORBA, RMI…)
  - Transferts de fichiers
  - Partage de base de données
  - Electronic Data Interchange (EDI) = norme définissant le(s) protocole(s) + le format d'échange de données pour le B2B
  - Web Services



- Souvent pour les flux de masse cadencés (« batch ») :
  - Transferts de fichiers
  - Partage de base de données
  - Batch = script qui ordonne et cadence un déplacement de données en volume
    - Solution « historique » et sans doute encore l'une des plus utilisées
  - Extract-Transform-Load (ETL) = progiciel de batch « à grande échelle » permettant de connecter des entrepôts de données

Modalité

- Flux synchrone = **bloquant** pour l'émetteur et le récepteur
  - Suppose la disponibilité de l'émetteur et du récepteur au même moment
  - *Exemple : appel de méthode RMI*

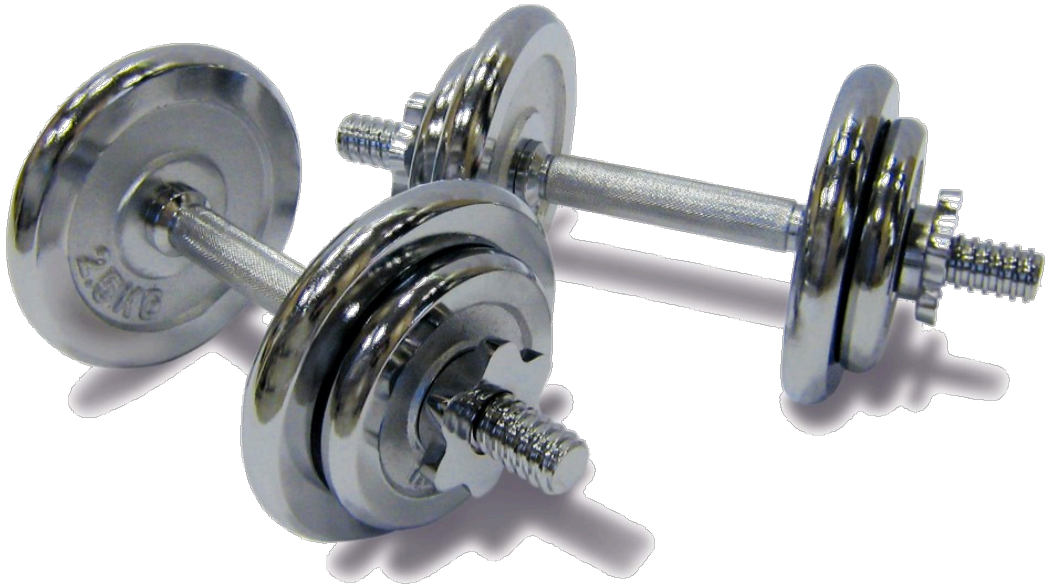
- Flux asynchrone = **non bloquant** (émission / réception différées)
  - Suppose l'existence d'une zone de stockage intermédiaire
  - *Exemple : emails*

- Flux requête-réponse = l'émetteur et le récepteur se connaissent
  - Contact direct
  - *Exemple : récupération d'une donnée dans un référentiel*

- Flux publication-abonnement = les récepteurs s'abonnent aux flux sans connaître les émetteurs
  - Contact indirect
  - *Exemple : abonnement aux mises à jour d'un référentiel de données*

Exercice 3 : qualification de flux

☛ http://wwwdi.supelec.fr/hardebolle/SOA/ExercicesArchi



Plan

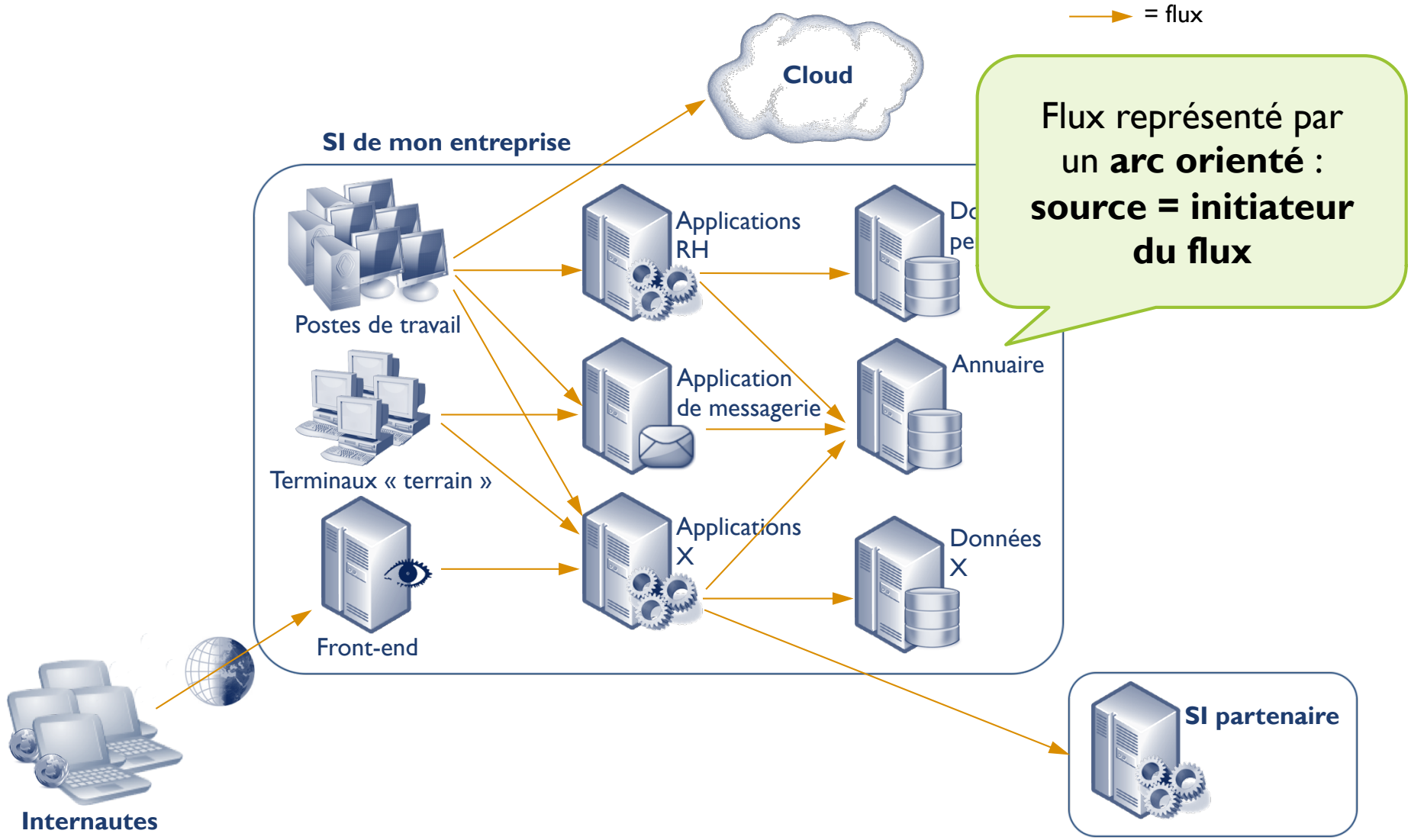
① Applications d'entreprise
- Typologie des applications
- Cartographie applicative

② Patrons d'architecture pour les applications
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux
- Typologie
- Qualification des flux

④ **Architectures d'échange**
- Typologie des architectures
- Solutions logicielles

Architecture point-à-point

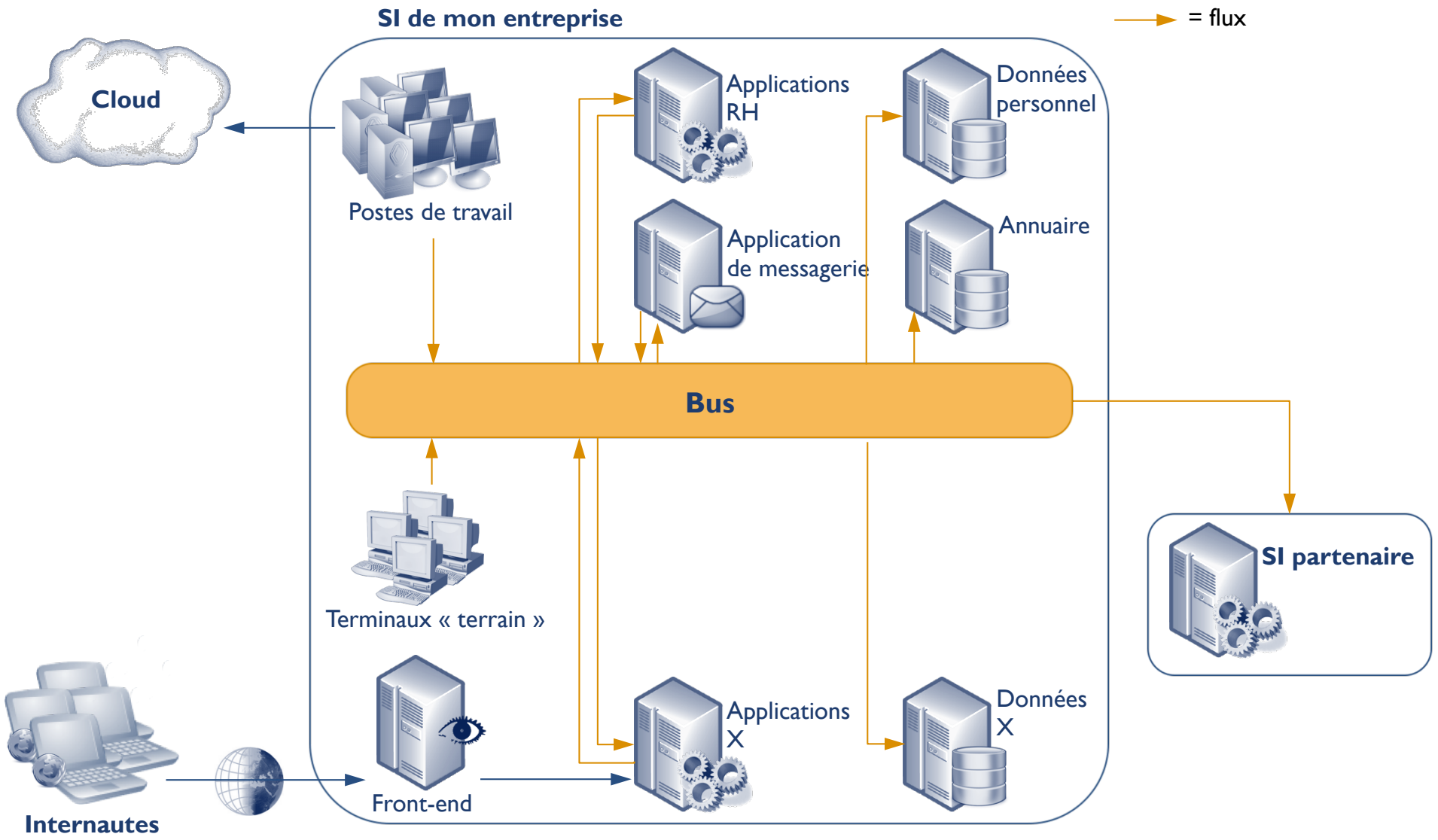


Analyse des solutions point-à-point

- Architecture « intuitive »
- Simplicité de mise en œuvre dans le cas où le nombre d'applications à intégrer est faible
- Efficacité des échanges directs
- Problème de passage à l'échelle
  - Si N applications, N(N-1)/2 liens…
  - Effet « plat de spaghettis »
- Evolutivité très réduite
  - Intégration d'une nouvelle application = ajout de nombreux nouveaux liens
  - Couplage fort entre les applications
    ➜ fort impact des évolutions (notamment interfaces)
- Exploitation et administration complexes
  - Manque de visibilité sur les échanges

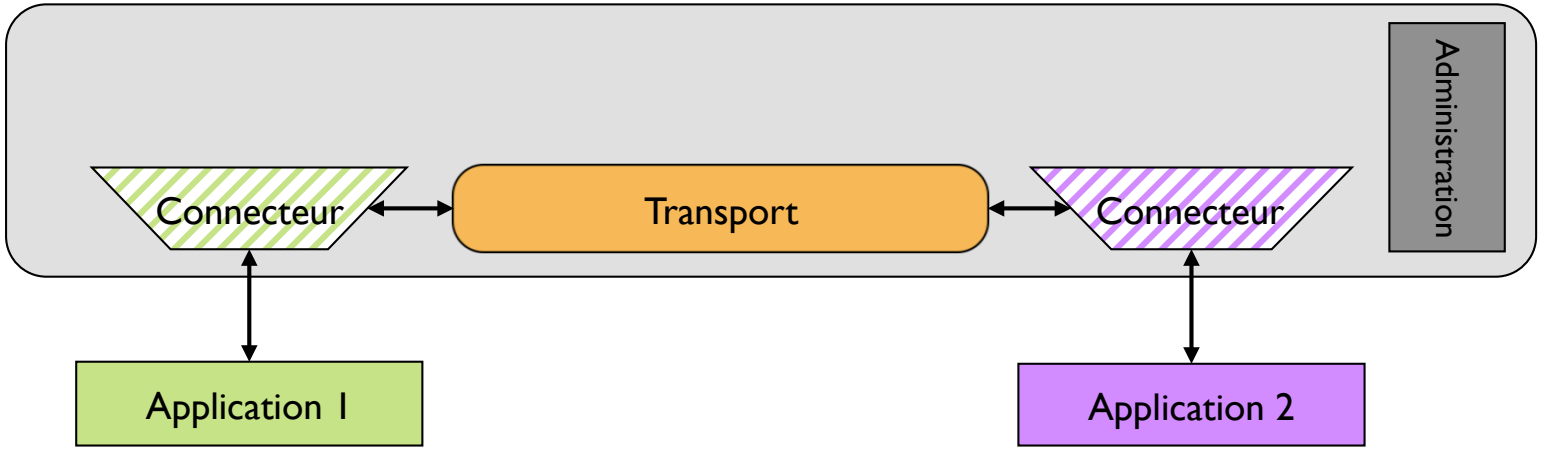
Source : Solucom

Architecture bus



Exemple de solution de type bus : le MOM

- Middleware Orienté Message (MOM) = bus logiciel de transport qui permet à des applications de recevoir des messages émis par d'autres
  - Connectivité : supporte différents protocoles de communication
  - Transport :
    - Garantit l'acheminement (intégrité, gestion des erreurs)
    - Gère différentes modalités : synchrone/asynchrone, publication/abonnement…
    - Gère les transactions
  - Existe aujourd'hui sur la plupart des serveurs d'applications



Source : Solucom

Exemple avec Java EE : JMS et EJB Message

- JMS (Java Message Service) = interface Java standard pour les MOM
  - Files d'attentes (queues) *pour le mode requête / réponse*
  - Sujets (topics) *pour le mode publication / abonnement*

- EJB Message = composant invoqué par messages
  - Traite les messages postés dans une file / sujet
  - Poste des messages réponse dans la file / sujet

```
@MessageDriven(mappedName="jms/Queue")
public class MyMessageBean implements MessageListener {

     public void onMessage(Message inMessage) {
          TextMessage msg = null;
          try {
               if (inMessage instanceof TextMessage) {
                    msg = (TextMessage) inMessage;
                    System.out.println("Message received: " + msg.getText());
               }
          } catch (JMSException e) { …   }
     }
}
```

Source : Les Entreprise JavaBeans 3.0 (EJB 3.0) , Jean-Marc Farinone, CNAM Paris

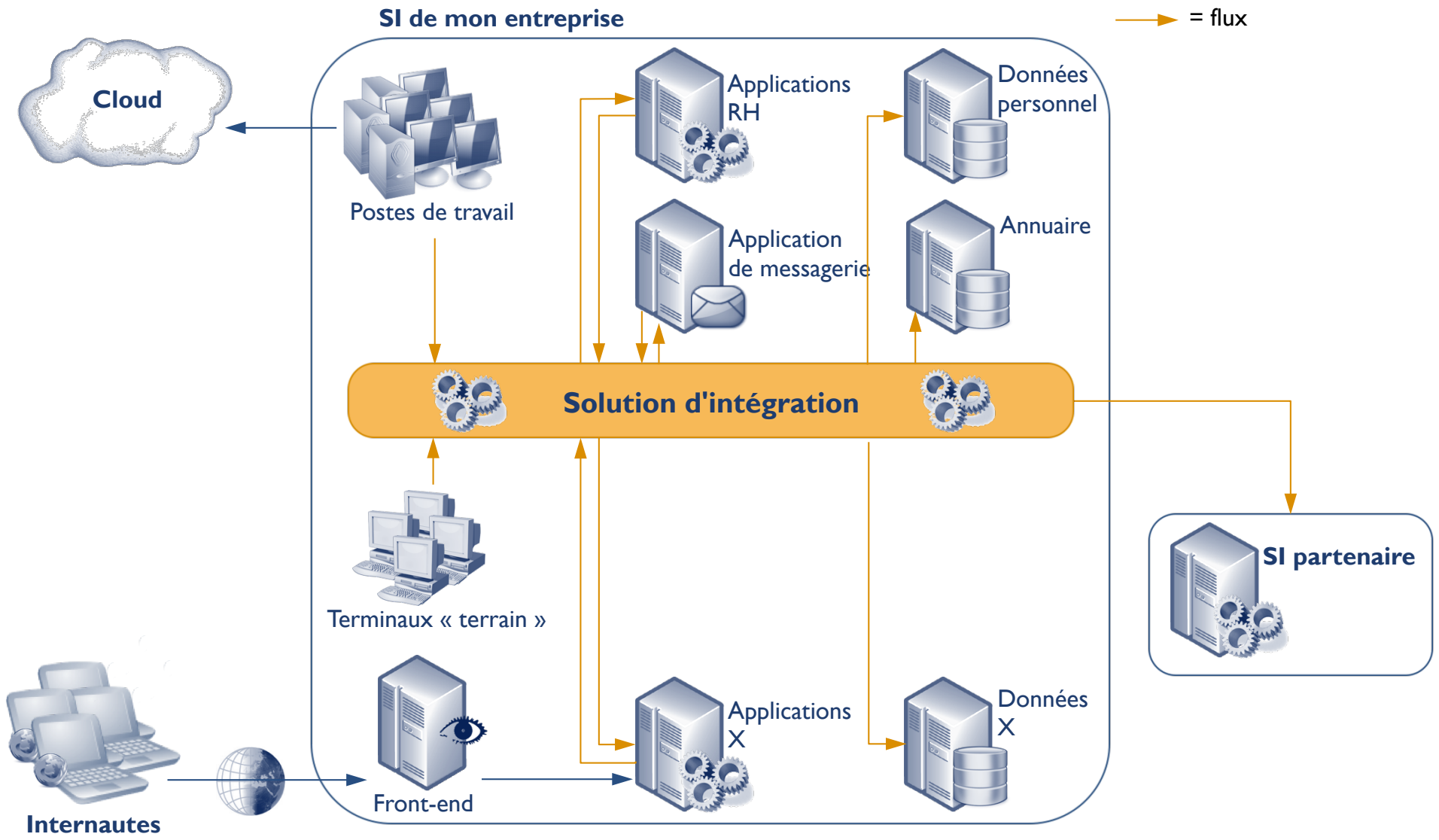
Analyse des solutions de type bus

- Passage à l'échelle facilité
  - Si N applications, au plus N liens bidirectionnels
- Meilleure évolutivité
  - Intégration d'une nouvelle application = un seul connecteur
- Couplage faible entre les applications
- Services de transport
  - Acheminement garanti des données (reprise sur erreur, gestion des doublons)
  - Intégrité des données

- Adhérence forte entre les applications et le bus
- Couplage fort entre les formats des données
  ➜ fort impact des évolutions du format d'échange
- Plate-forme centralisée
  ➜ hautement critique
  ➜ goulot d'étranglement

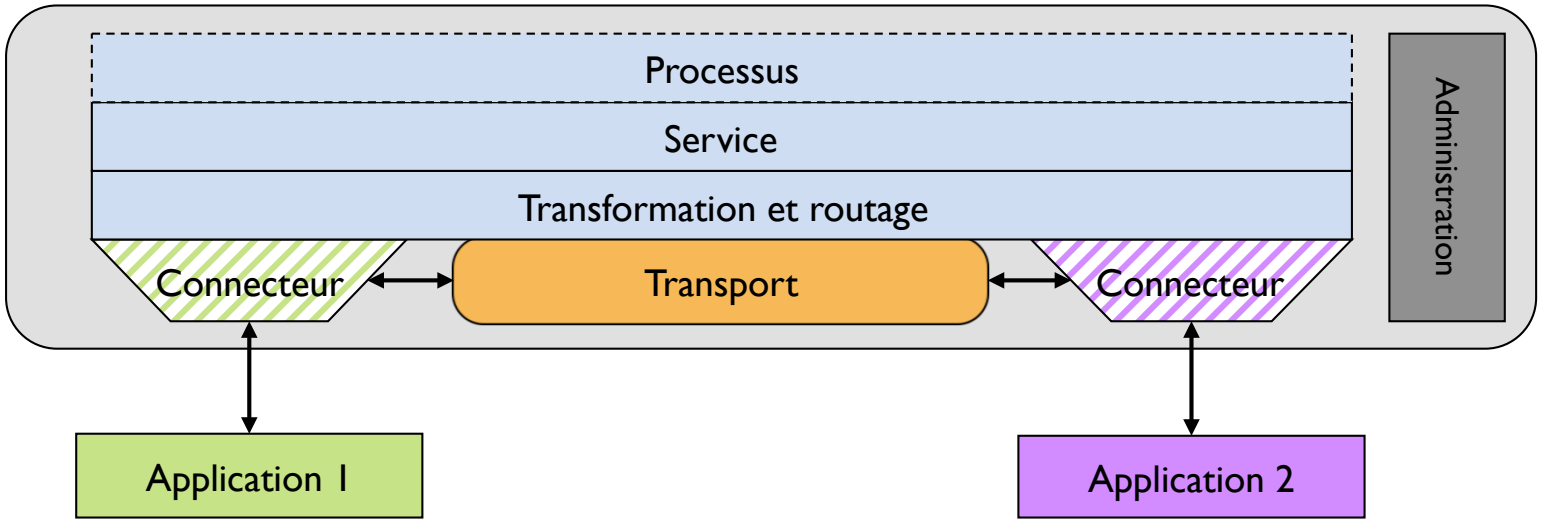
Source : Solucom

Architecture intégrée



Exemple de solution intégrée : l'EAI

- Enterprise Application Integration (EAI) = progiciel d'intégration inter-applicative
  - Connectivité & Transport
  - Transformation : gère l'hétérogénéité des formats et des valeurs
  - Routage : adresse intelligemment les données aux différents destinataires
  - Service : encapsule la logique d'intégration
  - + Eventuellement, processus : orchestre les services d'intégration



Source : Solucom

Analyse des solutions EAI

- Relations entre processus métiers et échanges inter-applicatifs plus lisibles

  Urbanisation fonctionnelle
  + Urbanisation technique
- Services applicatifs riches
- Coûts d'administration moins importants
- Coûts de développement réduits

La quasi-totalité des projets d'EAI se soldent par un échec…
70 % des projets d'intégration échouent…
(2003)

- Important travail d'urbanisation et /ou de réorganisation fonctionnelle indispensable
  - *Sinon « plat de spaghetti » dans l'outil…*
- Experts indispensables (et malheureusement très rares)
- Projets transverses par essence
  - Difficulté à établir les responsabilités
  - Problématiques organisationnelles (nombreux acteurs, besoin de processus)
- Technologies d'interconnexion propriétaires

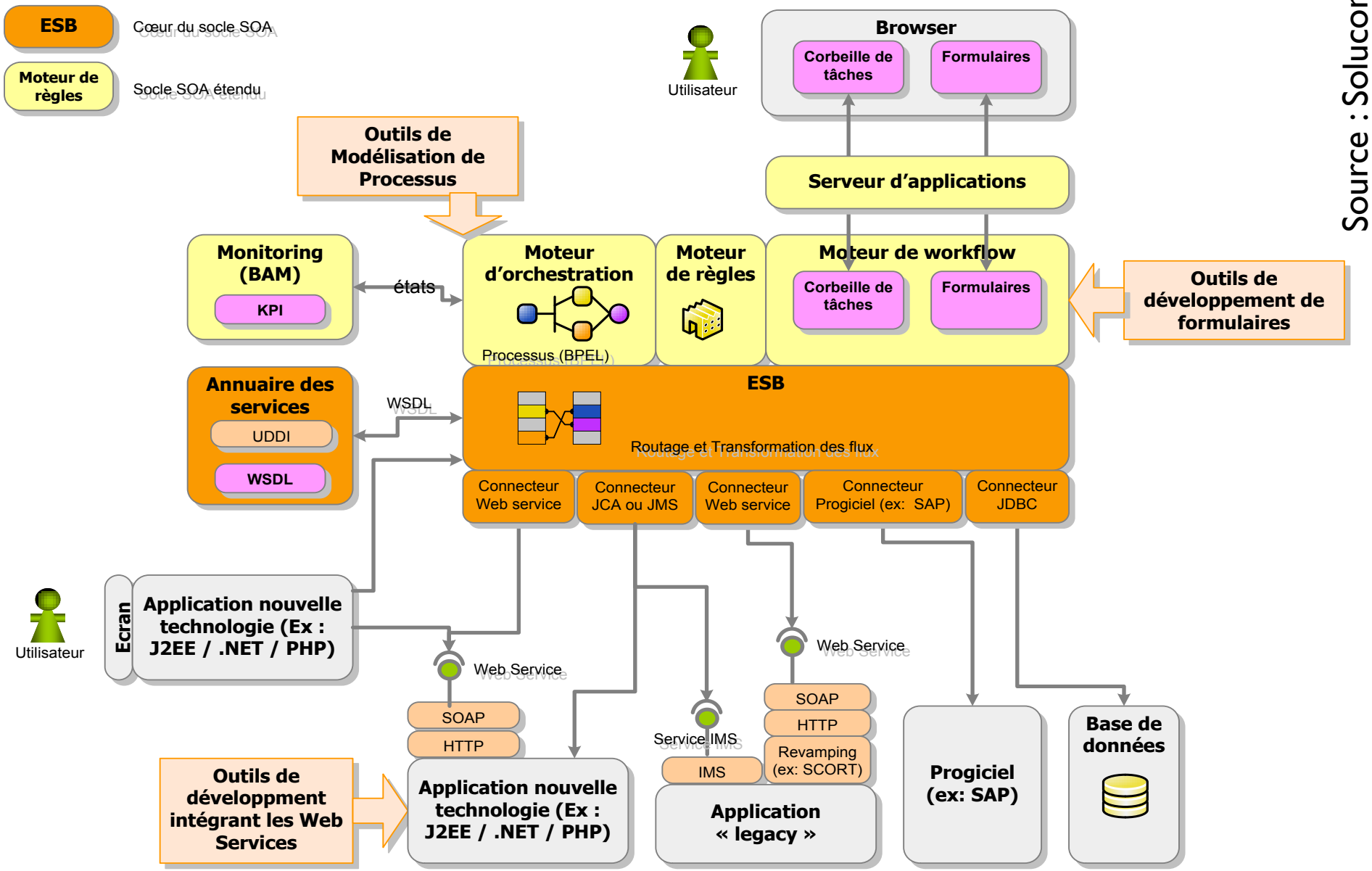
Source : Solucom

Autre solution intégrée : ESB

- Enterprise Service Bus (ESB) ≈ EAI basé sur les standards
  - Connectivité & Transport
  - Transformation : gère l'hétérogénéité des formats et des valeurs
  - Routage : adresse intelligemment les données aux différents destinataires
  - Service : encapsule la logique d'intégration
  - Processus : orchestre les services d'intégration
  - Monitoring : supervise le déroulement des processus

- L'ESB est considéré comme le socle technique de l'approche SOA

Enterprise Service Bus (ESB)



Source : Solucom

Plan

① Applications d'entreprise
- Typologie des applications
- Cartographie applicative

② Patrons d'architecture pour les applications
- Architecture en couches
- Modèle n-tiers
- Composants
- Infrastructures logicielles

③ Flux
- Typologie
- Qualification des flux

④ Architectures d'échange
- Typologie des architectures
- Solutions logicielles

Synthèse

① Quels sont les différents types d'applications dans le SI d'une entreprise ?

- Applications « classiques » pour les fonctions support, sinon dépend du métier
- Différents choix d'implémentation :
  - Développements spécifiques
  - Progiciels et COTS
  - Cloud

② Quels sont les grands patrons d'architecture pour les applications ?

- Répartition en couches applicatives
  - P Présentation
  - T Traitement
  - R Ressources
- Décomposition sur le modèle clients / serveurs (éventuellement distants)
- Encapsulation dans des composants (= boîtes noires avec interfaces publiques)
- Déploiement sur des infrastructures logicielles

Synthèse (suite)

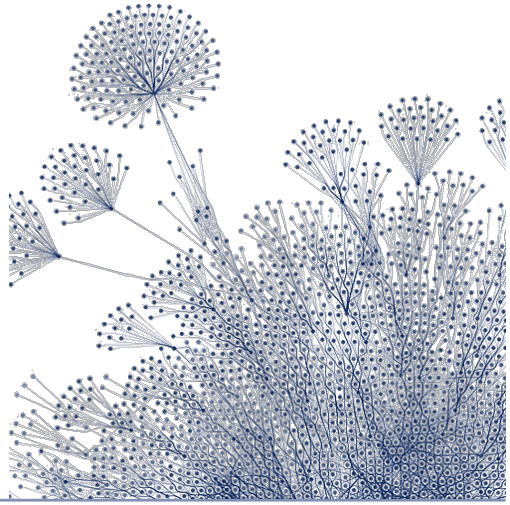
③ Comment les informations sont-elles échangées au sein du SI ?

- Différents types de flux, caractéristiques :
  - Périmètre
  - Granularité
  - Fréquence
  - Modalité

④ Quels types d'architectures supportent les échanges ?

- Point à point
- Bus
- Intégrées, avec en particulier la solution de type ESB

- + Besoin de combiner les solutions d'échange !



Contraintes de l'architecte

- *Un architecte est rarement amené à créer des architectures « from scratch »…*
- L'architecte doit tenir compte des contraintes exprimées et du contexte…
  - Contraintes exprimées
    - Besoins fonctionnels des utilisateurs
    - Exigences extra-fonctionnelles (performance, disponibilité…)
    - Référentiels d'entreprise
    - Stratégie économique (budget, time to market…)
  - Contexte = contraintes liées à l'existant
    - Existant applicatif
    - Environnement technique : technologies de développement, plateformes…
    - Organisation : compétences des équipes, externalisation…

☛ Généralement, nécessité de définir de plusieurs scénarios d'architecture

- Différents arbitrages possibles (priorités…)
- Problèmes de compatibilité entre contraintes, de réalisme…

Ce qui n'a (malheureusement) pas été abordé…

- Quelques-uns des autres sujets incontournables lorsque l'on parle d'architecture applicative :
  - Sécurité
  - Haute-disponibilité
    - Exemple : un million de fichiers sauvegardés toutes les 15 minutes sur Dropbox
  - Migration
  - …

- Comment évaluer si une architecture est « bonne » ?
  - Agilité / extensibilité
  - Evolutivité
  - Utilisation des standards
  - Sécurité
  - Coûts


MERCI!
THANK YOU!
FRAPAR.