

Java EE 6

JSR-316

Mathieu ANCELIN

- Master ICONE 2009
- Ingénieur d'études @ SERLI
- Open-source : Easybeans, JOnAS, GlassFish, JBoss Weld, Weld-OSGi, CDI-OSGi
- Membre «expert group» JSR-346 : CDI 1.1
- Leader du projet dependency-shot
- Equipe du poitou-charentes JUG
- Speaker : Solutions Linux, JavaOne 2010, JUGs
- @TrevorReznik

SERLI

- Fondée en 1981
- Société de conseil et d'ingénierie du SI située sur la technopole du Futuroscope
- Environ 60 employés
- Modèle économique unique basé sur la contribution
- Forte implication dans des projets open-source majeurs sur des fonctionnalités non triviales

Poitou-Charentes JUG

- Java User Group de la région poitou-charentes
- Réunir les acteurs du langage Java (pro, étudiants, chercheurs) afin d'échanger des idées sur les avancées de la plateforme
- Une conférence par mois
- Thèmes terrain et veille
- Une évènement annuel d'une journée
 - le JUG Summer Camp (à la rochelle :)

Plan

- Concepts de base
- Cas d'étude
- Java EE 6
- Serveurs d'applications
- TP

Plan

- **Concepts de base**
- Cas d'étude
- Java EE 6
- Serveurs d'applications
- TP

JCP

- Java Community Process
- Crée par Sun en 1998
- Coordonne l'évolution du langage Java et des technologies associées
- Composé d'entreprises du domaine Java
- et de particulier (adhésion gratuite)

JSR

- Java Specification Request
- Système normalisé pour faire évoluer la plateforme
- Description détaillée des spécifications et des technologies
- Cycle de vie
 - initiation
 - early draft
 - final release
 - maintenance

JSR

- Une JSR en phase finale doit proposer
 - une implémentation gratuite sous forme de code source (implémentation de référence)
 - un ensemble de tests (Technology Compatibility Kit) permettant de vérifier la compatibilité d'un implémentation tierce

JavaBeans

- Composant logiciel réutilisable et manipulable visuellement dans un outil de conception
- Classe java sérialisable
- Getter et setter sur les propriétés privées
- Constructeur par défaut
- Callback d'évènements si nécessaire

POJO

- Plain Old Java Object - Bon vieil objet Java
- Objet java standard avec un nom cool :)
- Très simple avec peut-être quelques annotations
- Un POJO n'est pas lié à un framework compliqué (voir EJB2)

Annotations

- Ajoute des propriétés à certains éléments du code

```
@Target(FIELD)  
@Retention(RUNTIME) @Email  
public interface Email {} private String email;
```
- Méta données du code
- Alternative aux fichiers XML
- Disponibles à la compilation ou à l'exécution
- Accessible grâce à l'API d'introspection ou via les annotations processors

IoC

- Principe d'Hollywood : ce n'est pas vous qui nous appelez, c'est nous qui vous appelons
- Délégation du cycle de vie des composants au framework
 - souvent appelé «conteneur» dans ce cas
- Gestion de la configuration par le conteneur
- Quand on «arrive» dans une portion de code, les objets sont censés être prêt à fonctionner

Injection de dépendances

- Création dynamique de dépendances à l'exécution
 - Casse les dépendances directes entre les composants (via interfaces) et cache les implems.
 - Construit le graphe d'objet à votre place (remplace le « **new** »)
 - Résolution dynamique (conf.) des dépendances au runtime et injection des valeurs
- S'inscrit dans le principe de l'IoC. Ce n'est pas vous qui instanciez les objets, c'est le conteneur.

AOP

- Programmation orientée aspect
- Séparation des considérations techniques (dans le code) de la logique métier
- Définition (par configuration) de points d'insertion dans le code où du code technique va s'exécuter
- Ajout de fonctionnalités à des objets sans toucher au code
 - avant et / ou après un appel de méthode
- Application à la sécurité, la journalisation, etc ...

JNDI

- Service d'annuaire
- Associe des noms à des objets
- Permet de retrouver des objets grâce à leur nom (grosse Map)
- Fonctions de nommage et d'annuaire pour les application Java
- Pièce maîtresse de Java EE ...
- ... mais de manière transparente

JDBC

- Accès SGBDR à partir de Java de manière abstraite
- Gestion de la connexion
- Envoi de requête SQL
- Exploitation des résultats
- S'appuie sur des pilotes génériques ...
- ... ou spécifiques à un vendeur

RMI

- Appel de méthodes à distance
- Utilisation en parallèle de JNDI
- Communication via IIOP entre les objets Java physiquement éloignés (JVM distinctes)
- Facilite le développement d'applications distribuées
- Masque la communication client / serveur au développeur

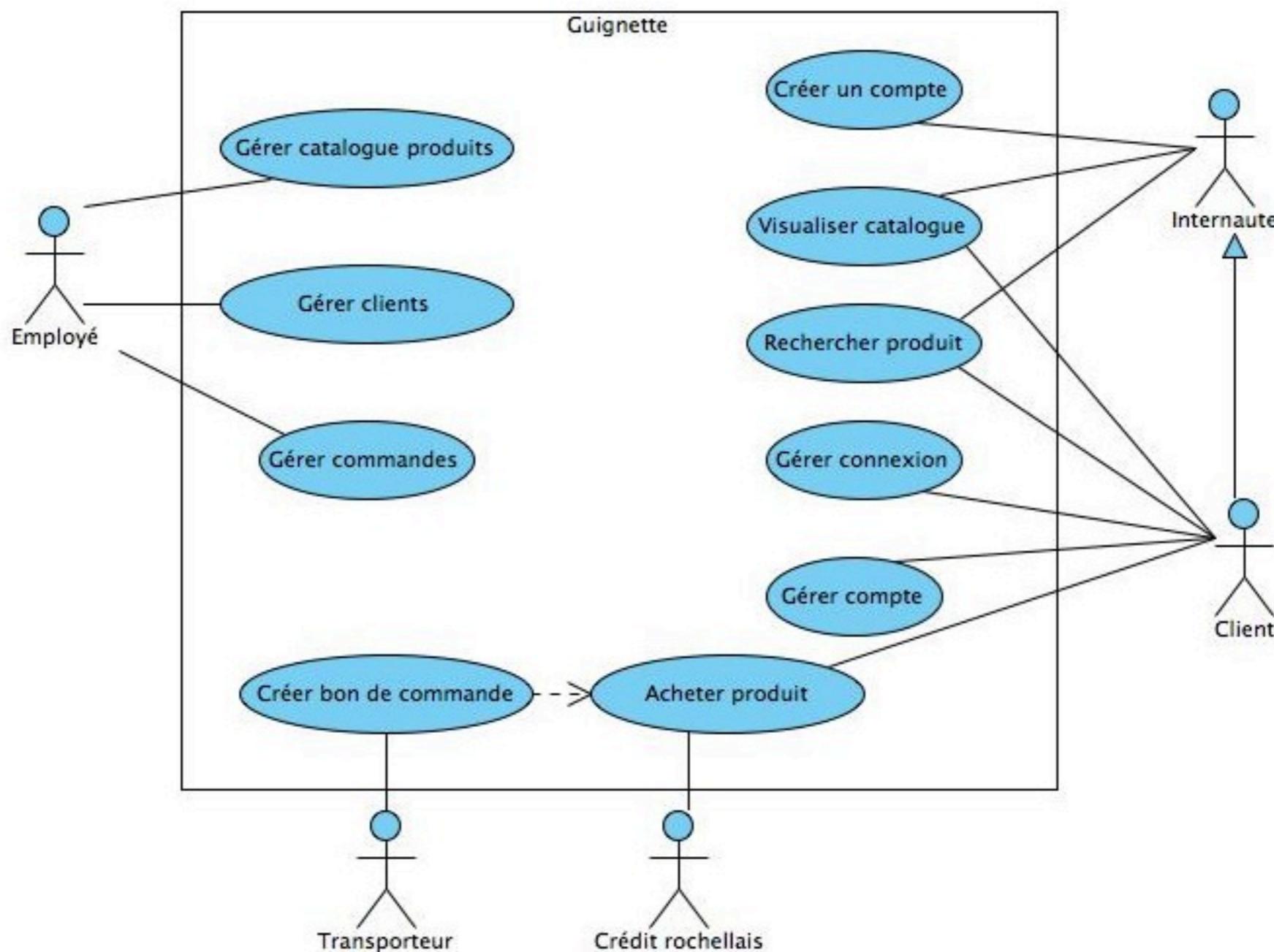
Plan

- Concepts de base
- **Cas d'étude**
- Java EE 6
- Serveurs d'applications
- TP

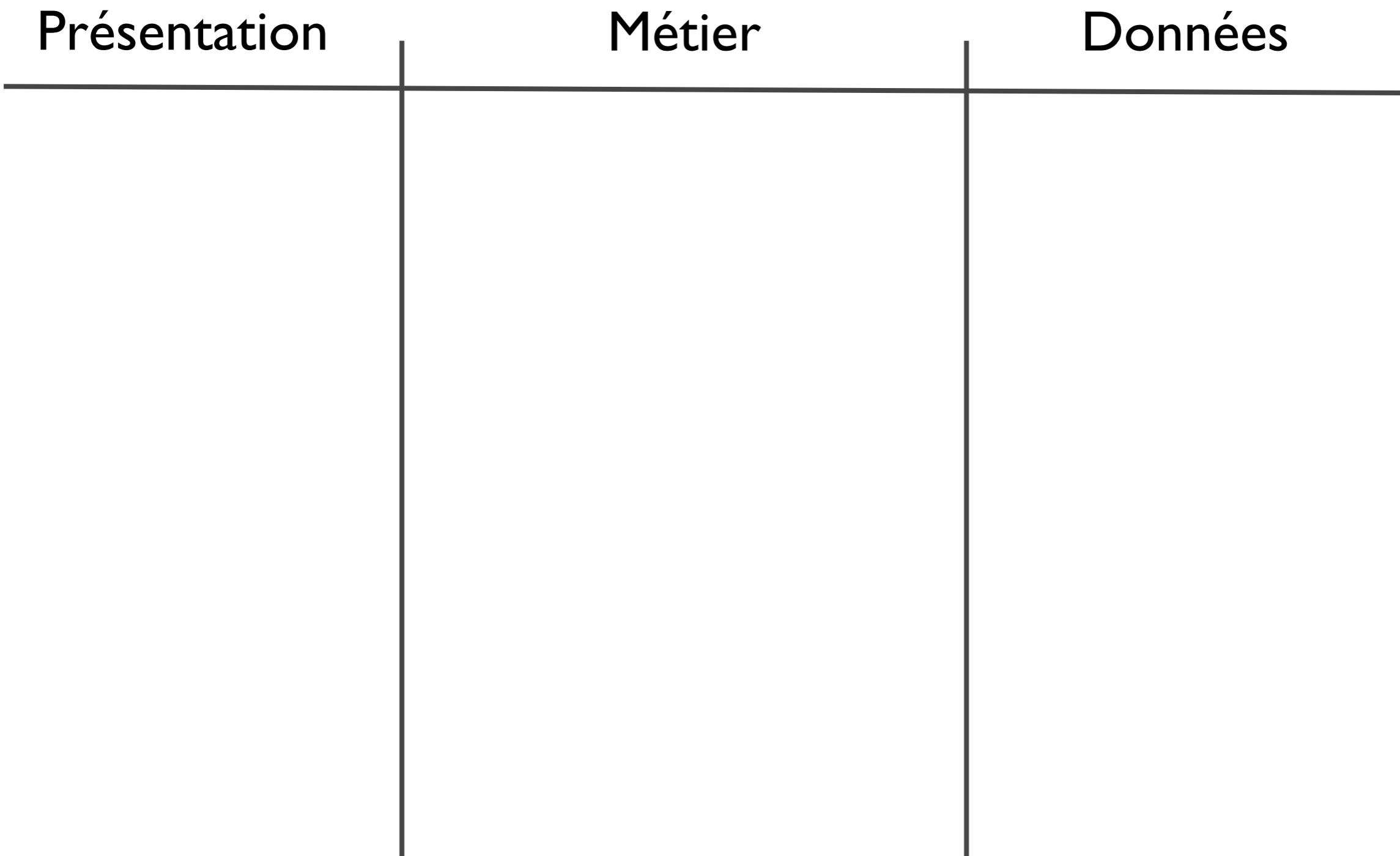
guignette-online.fr

- Bar de la guignette à La Rochelle
- Commerce local
- Envie de faire connaitre le produit
- Solution
 - boutique online

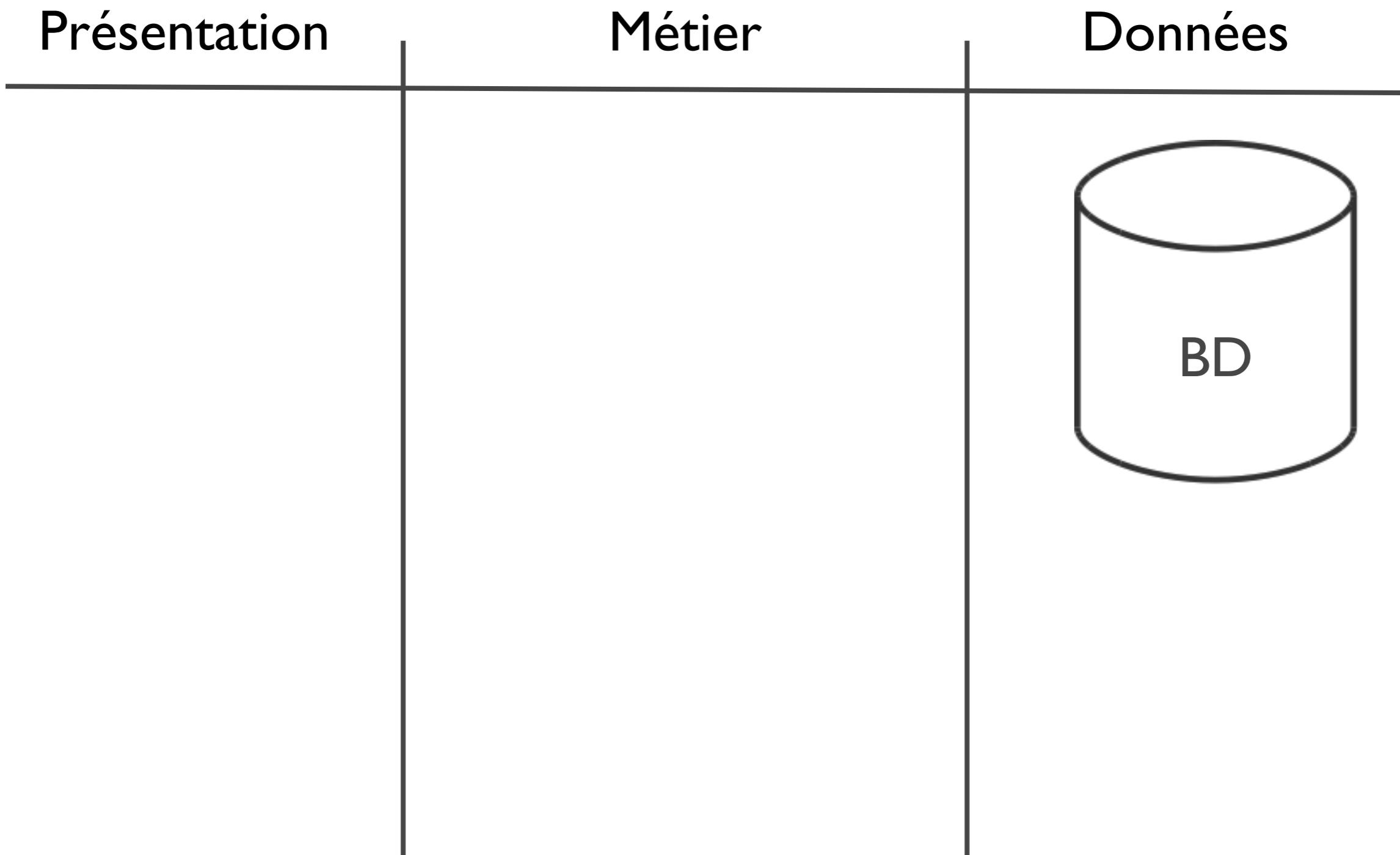
guignette-online.fr



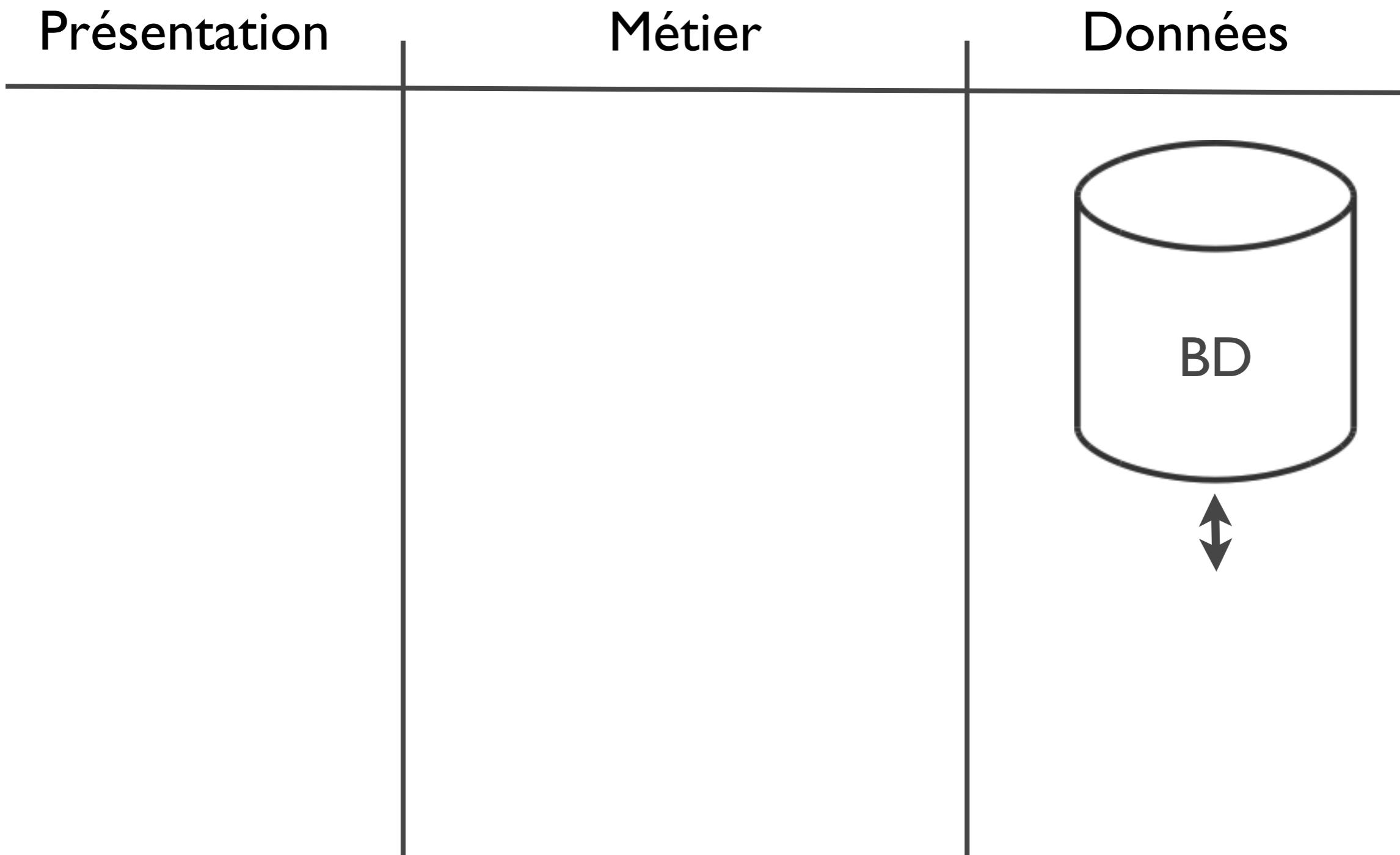
guignette-online.fr



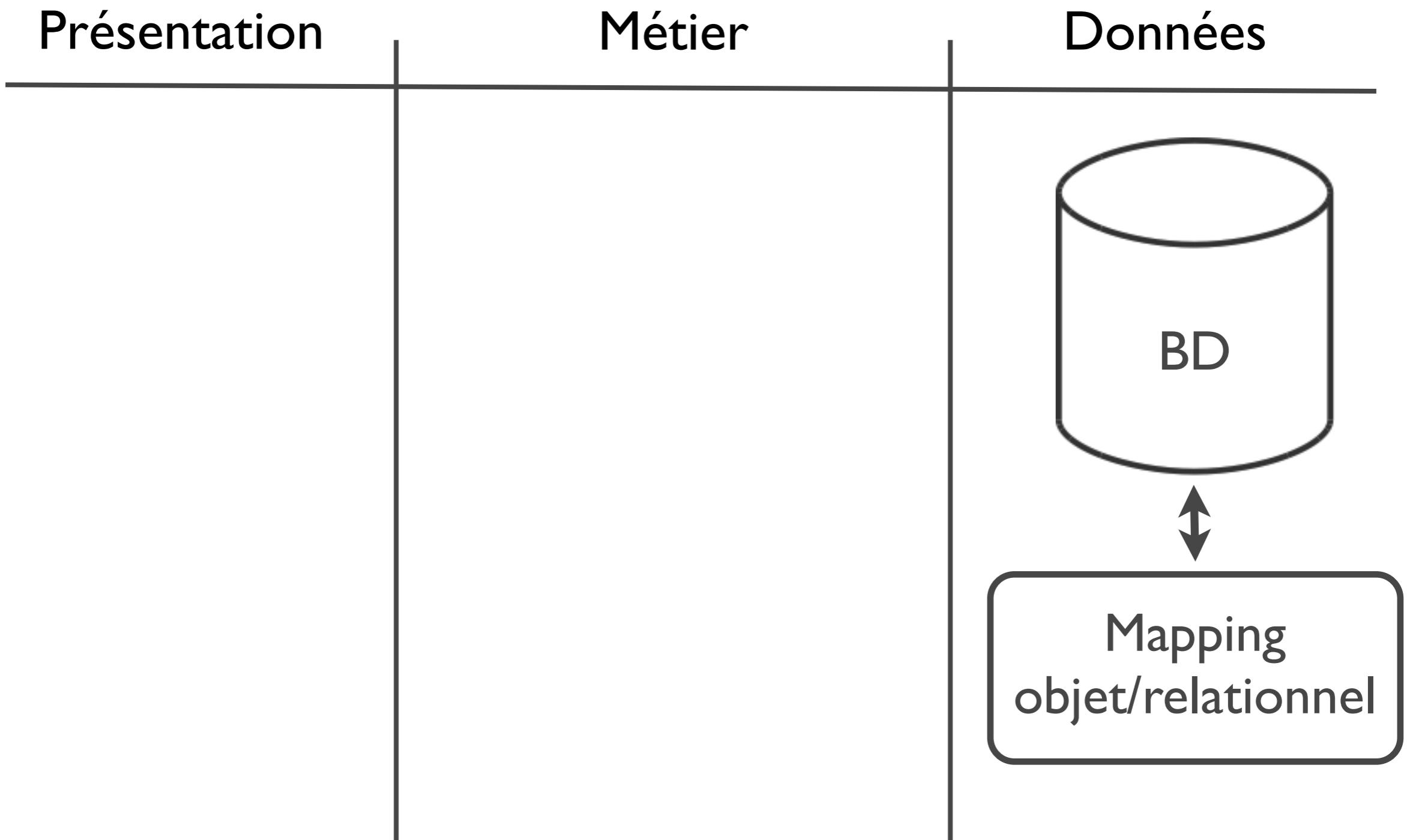
guignette-online.fr



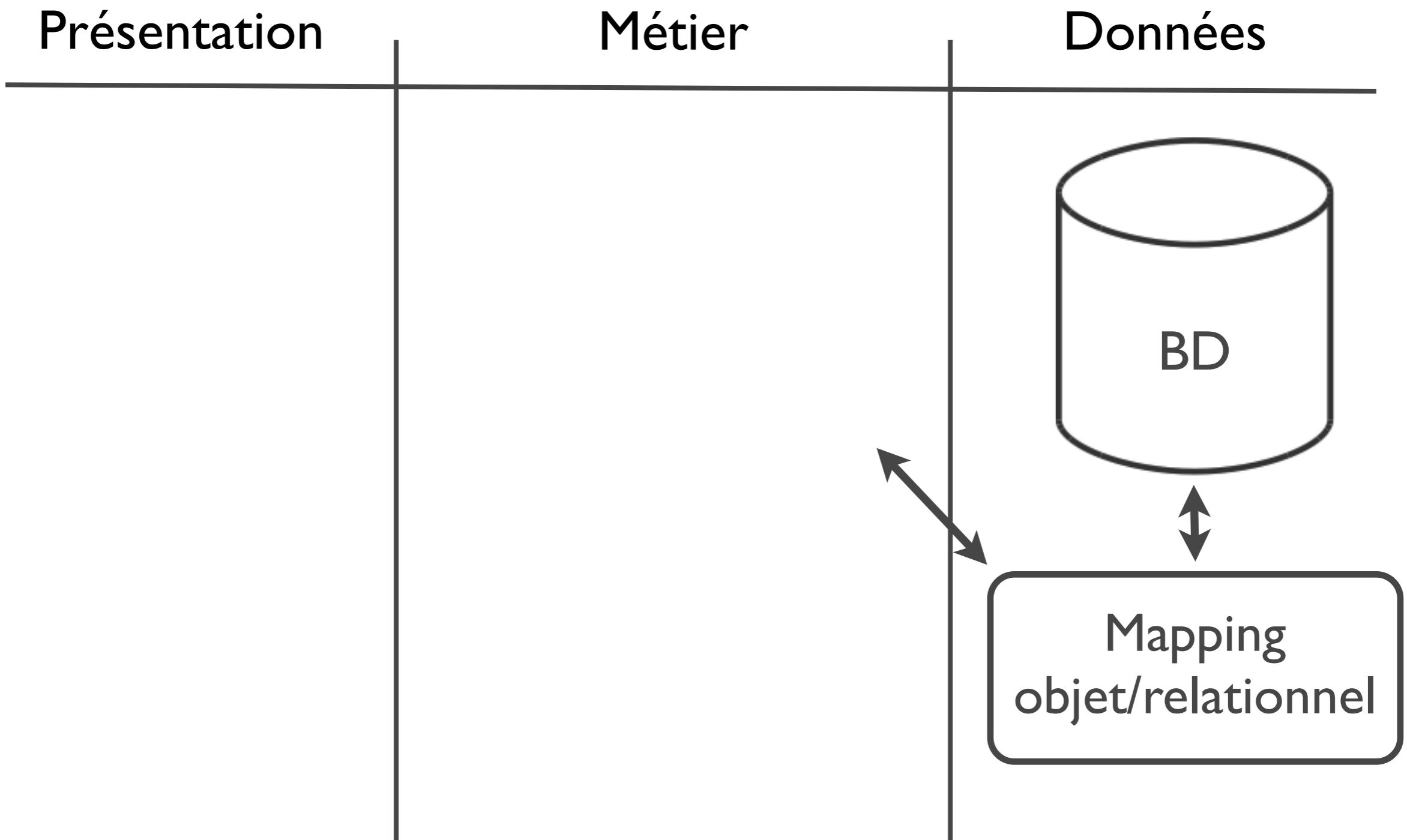
guignette-online.fr



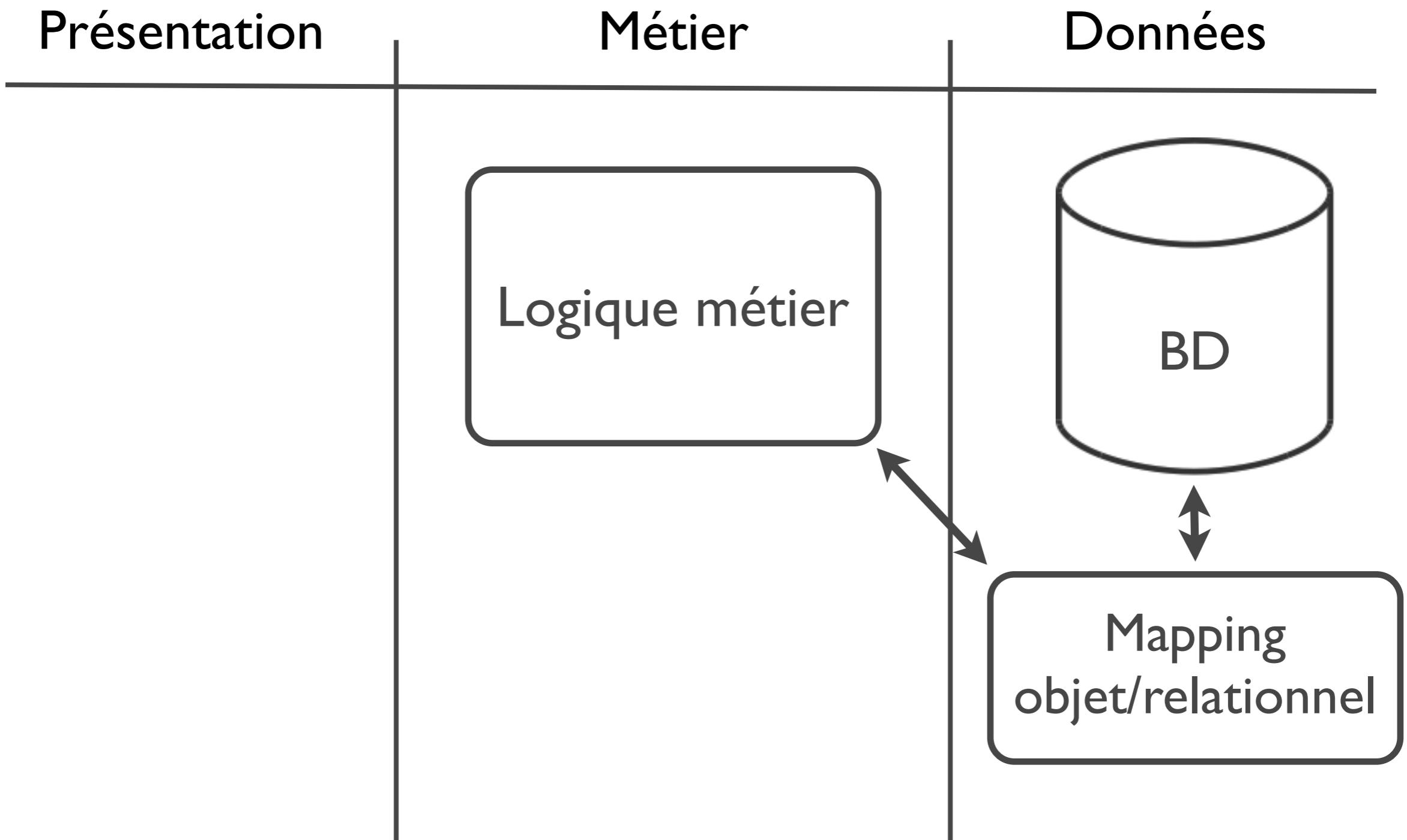
guignette-online.fr



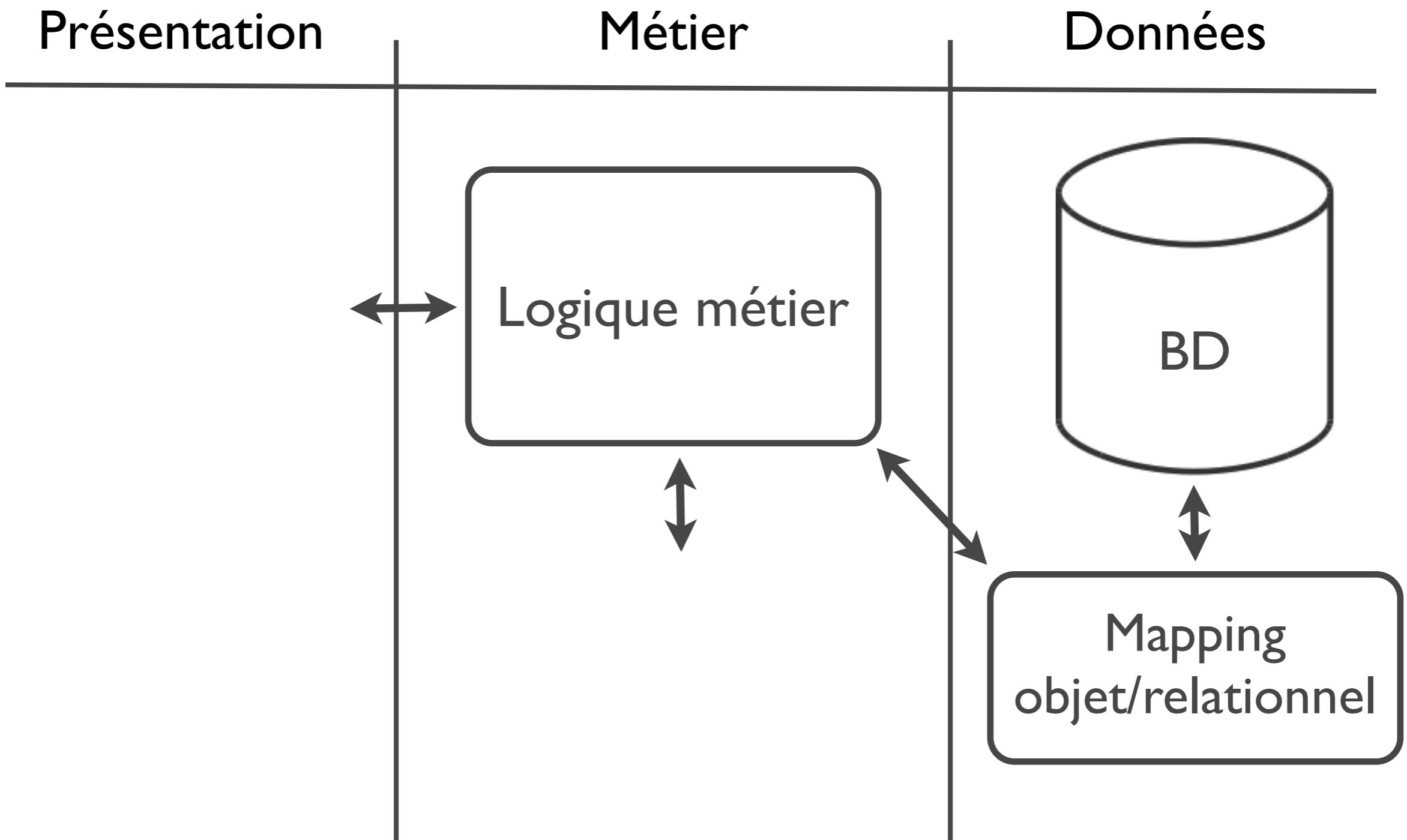
guignette-online.fr



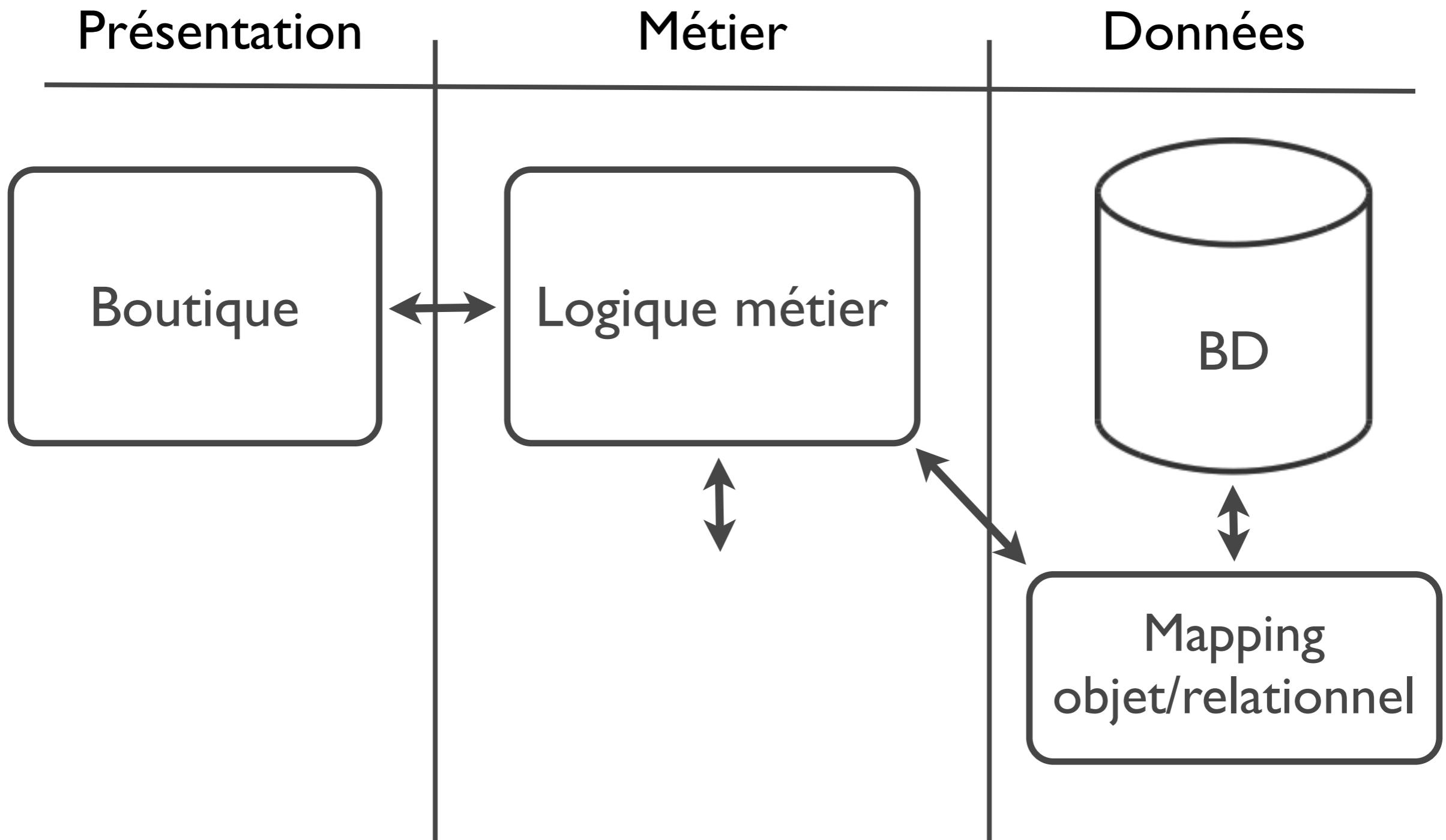
guignette-online.fr



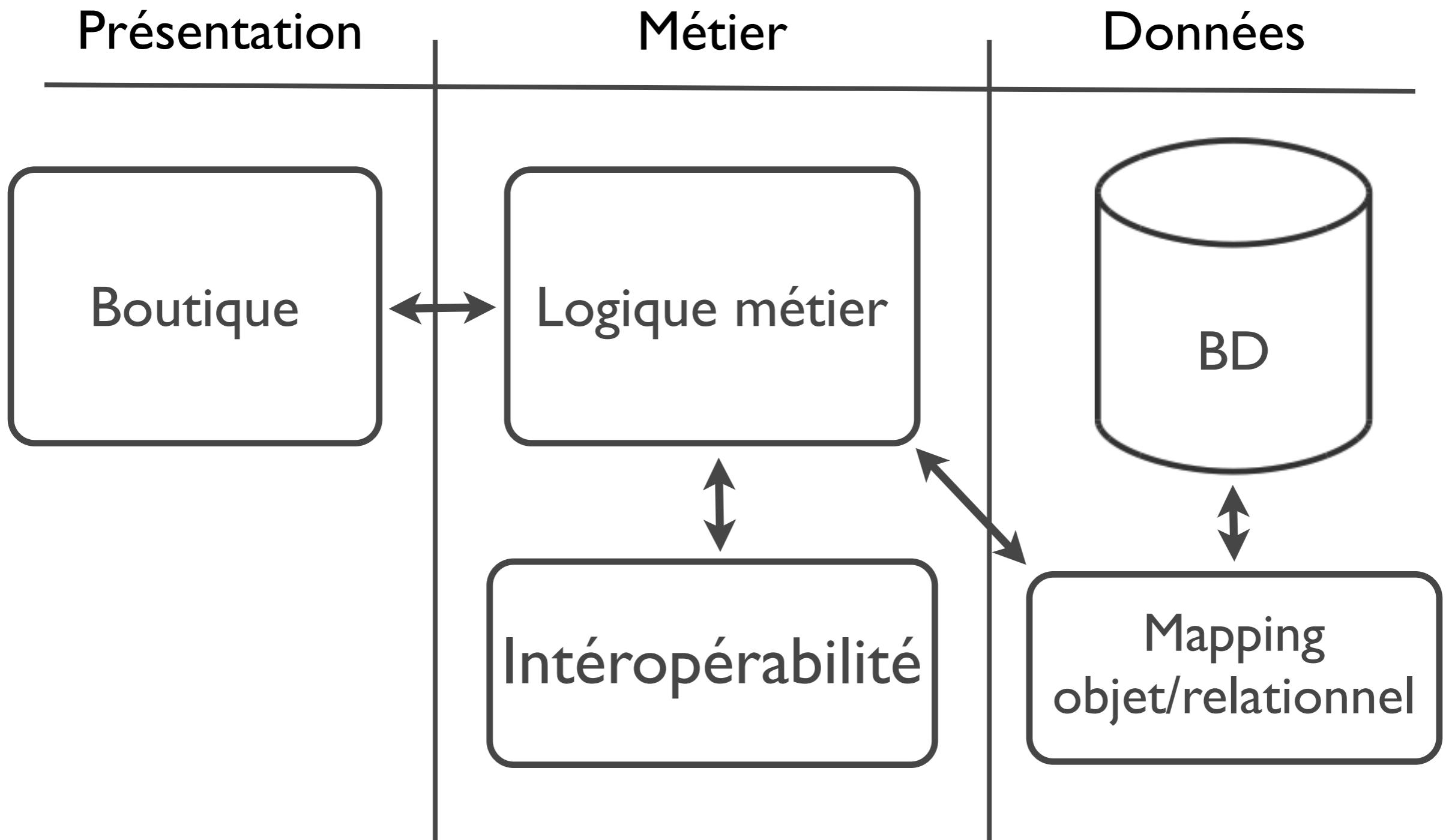
guinette-online.fr



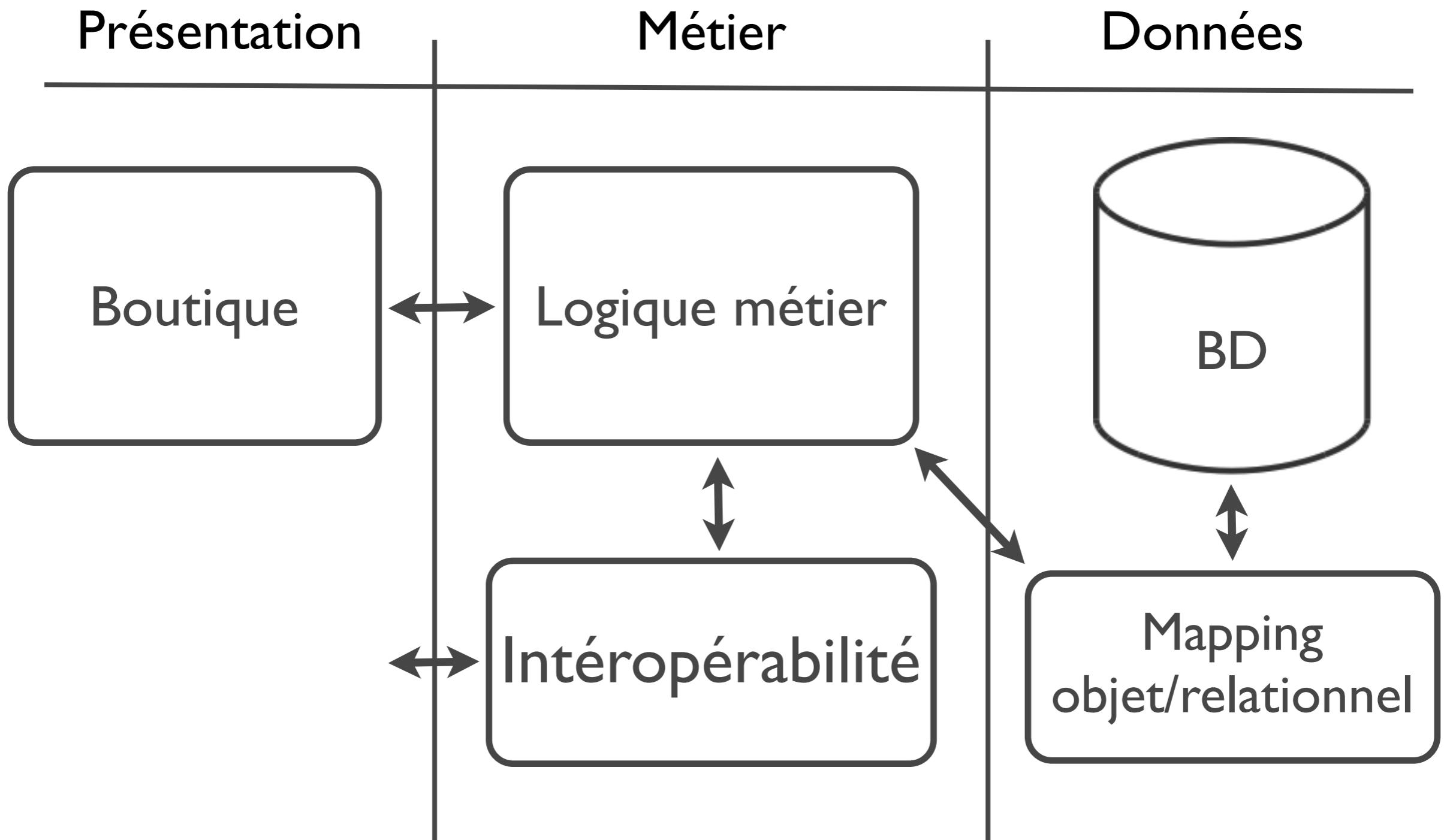
guignette-online.fr



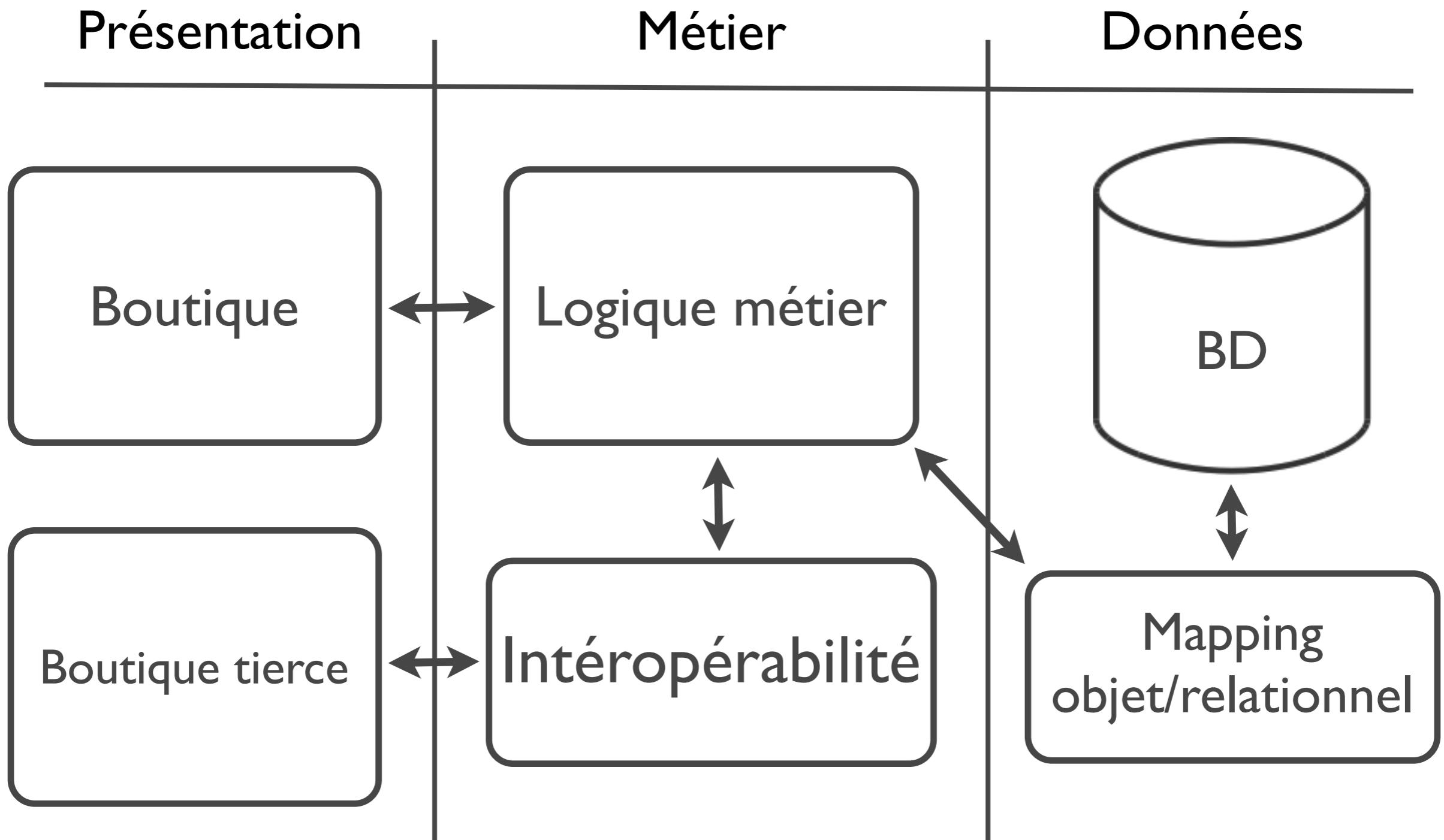
guignette-online.fr



guignette-online.fr



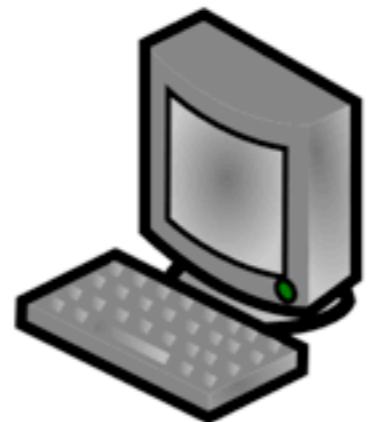
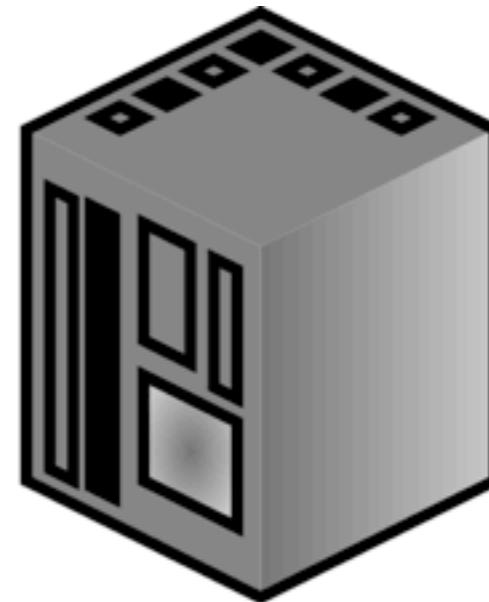
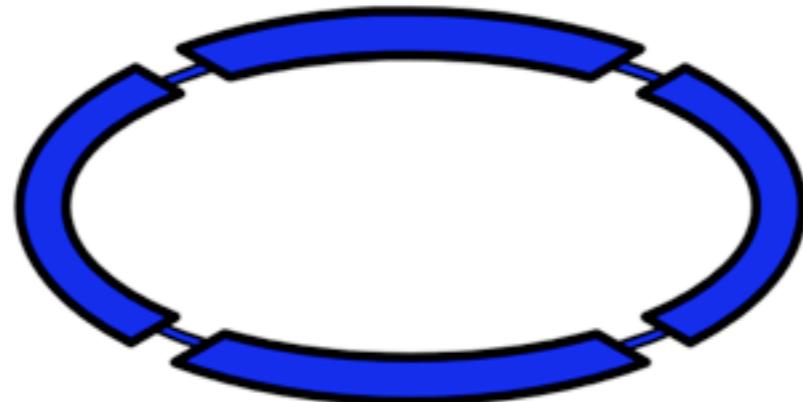
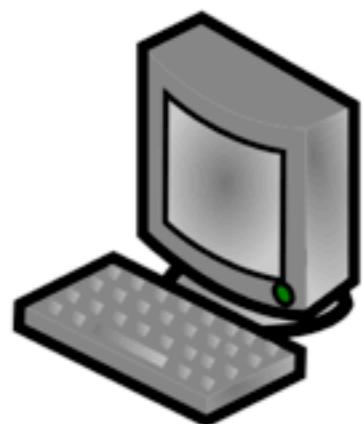
guignette-online.fr



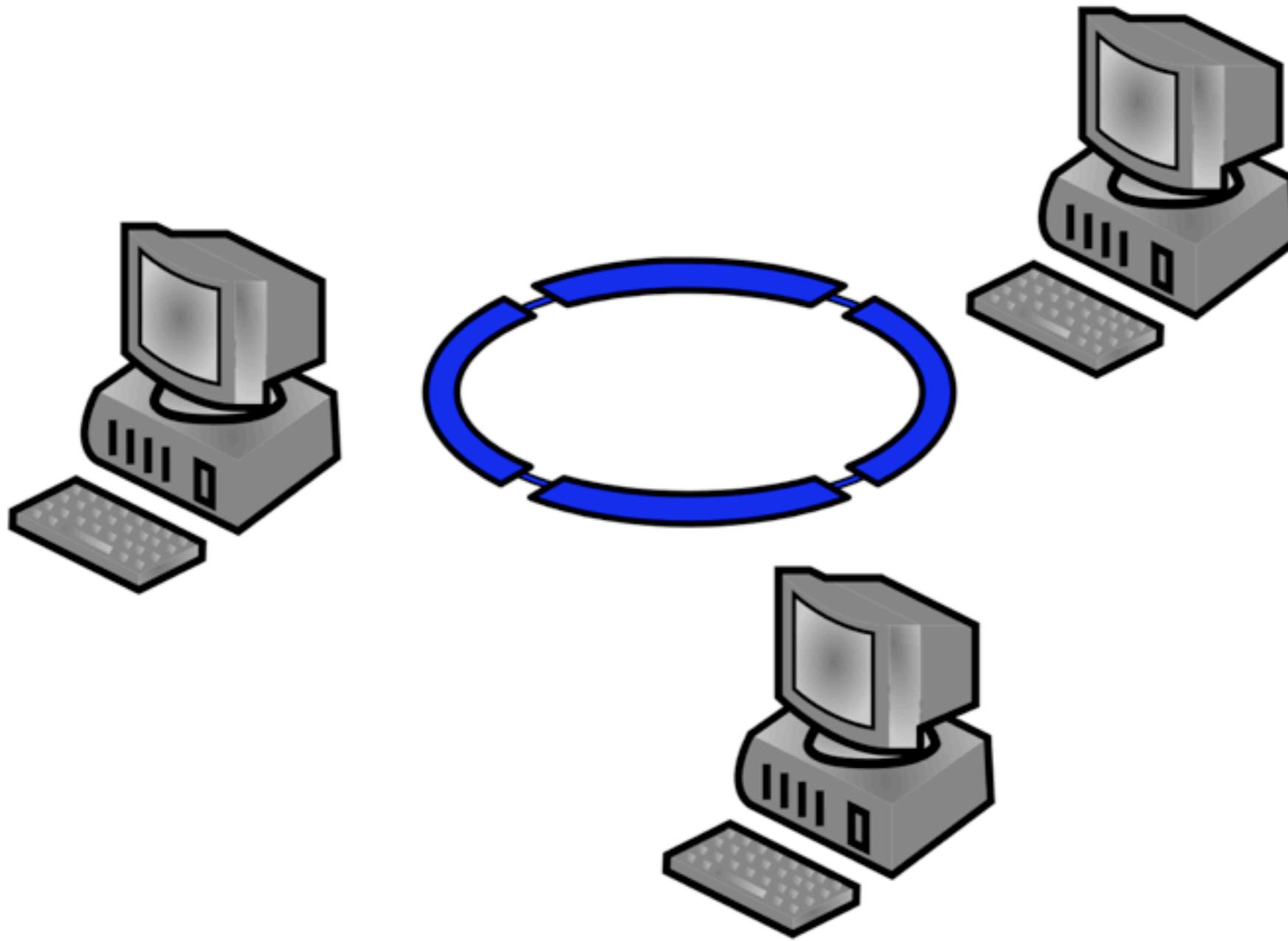
Plan

- Concepts de base
- Cas d'étude
- **Java EE 6**
- Serveurs d'applications
- TP

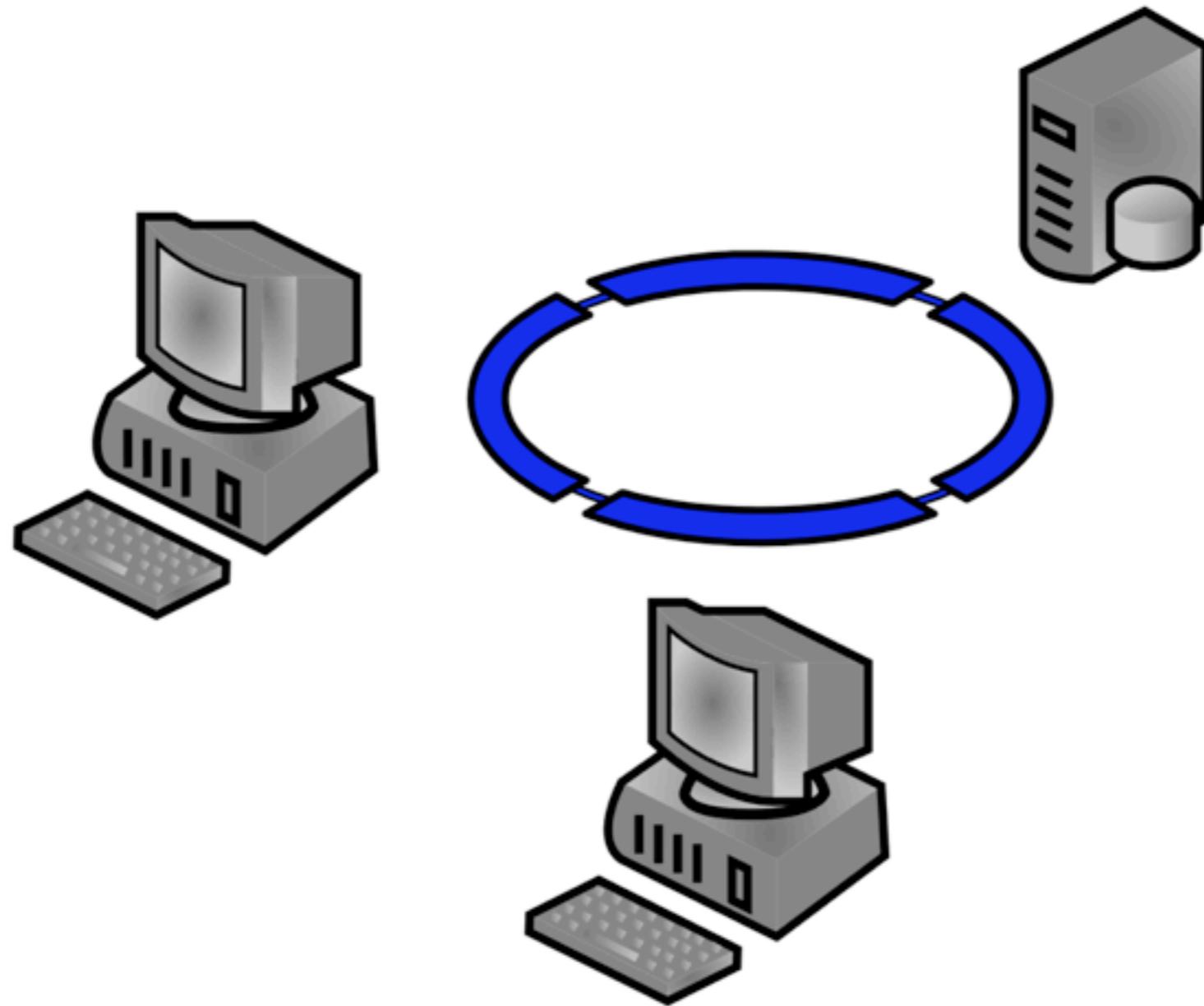
Un peu d'histoire



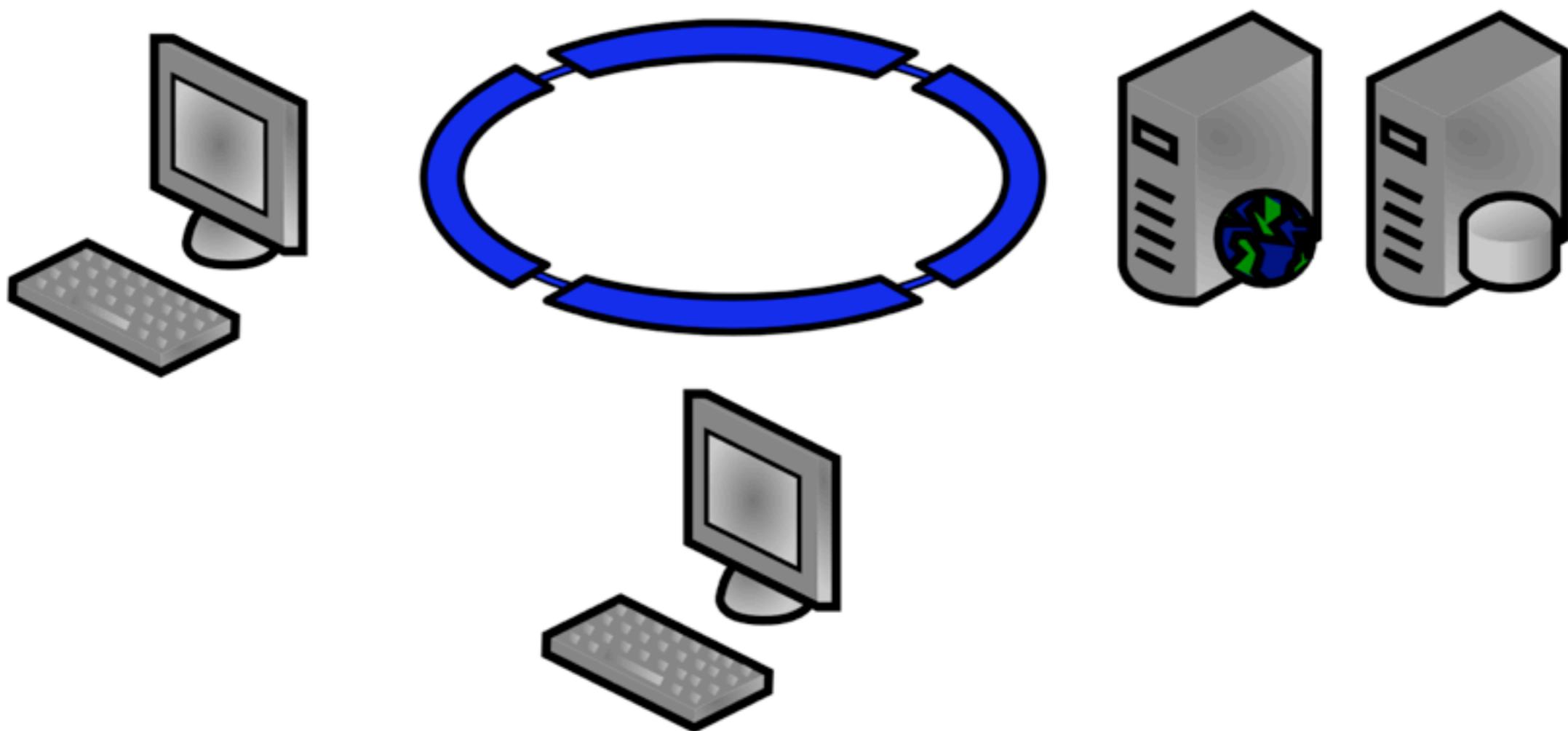
Un peu d'histoire



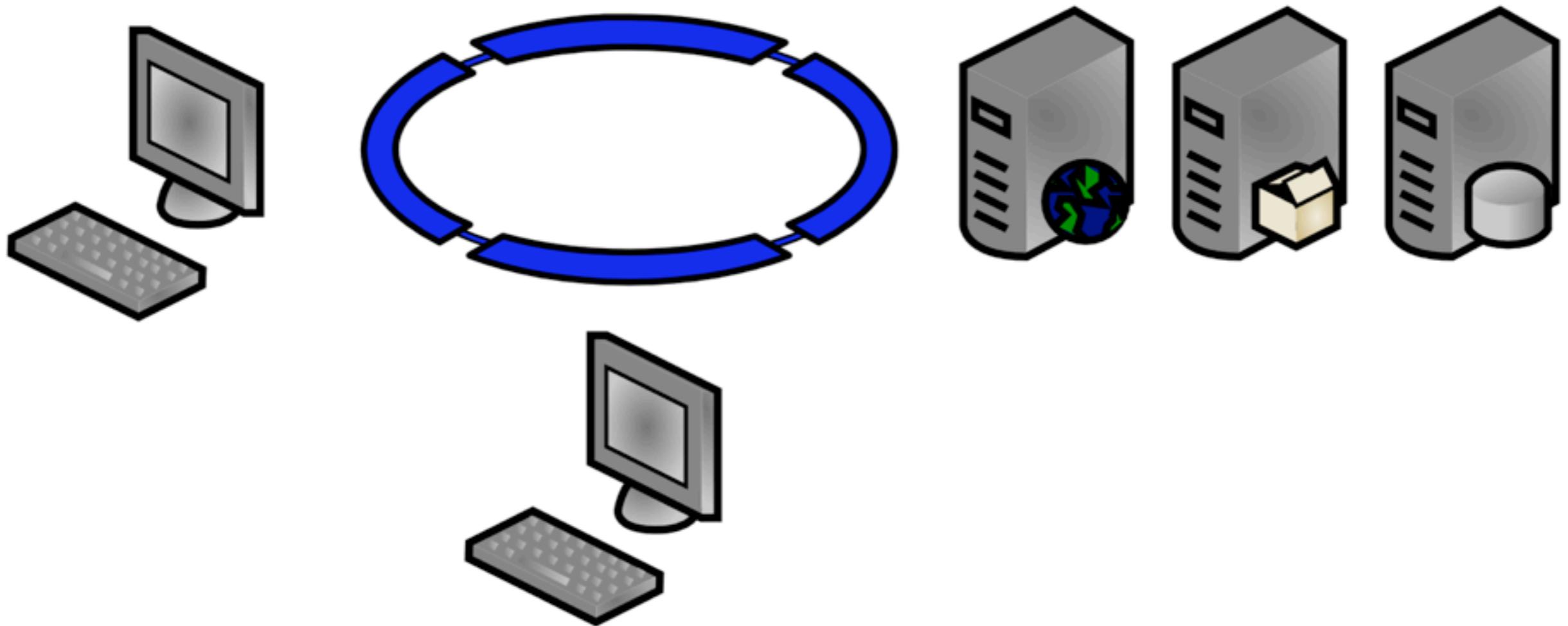
Un peu d'histoire



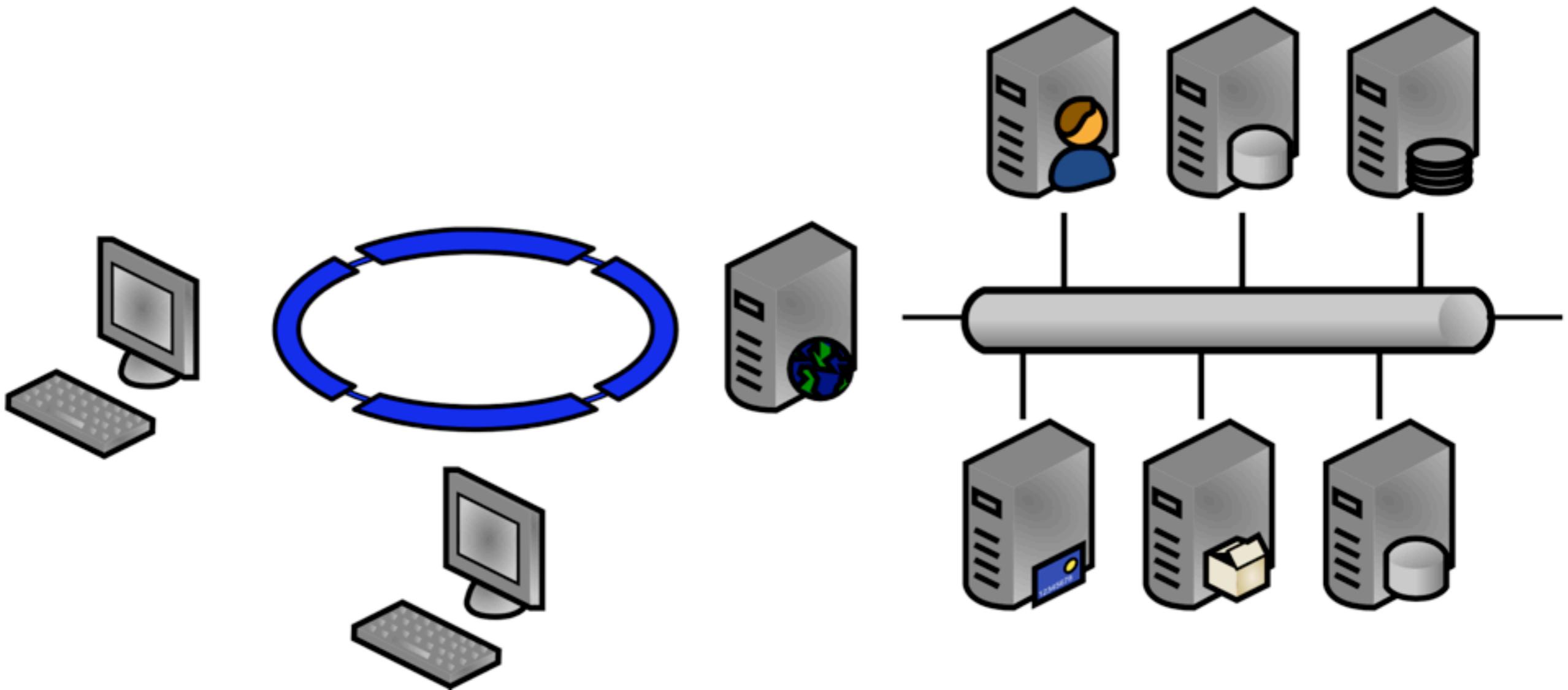
Un peu d'histoire



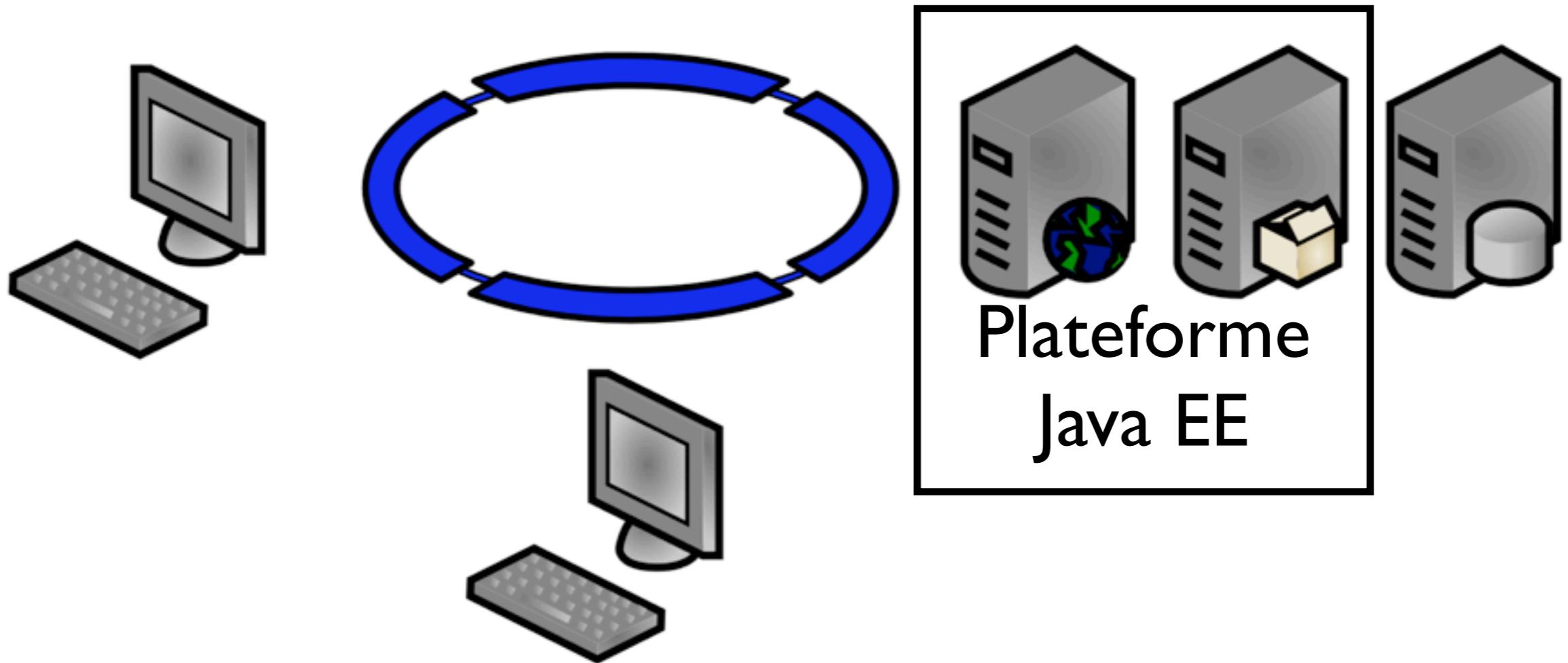
Un peu d'histoire



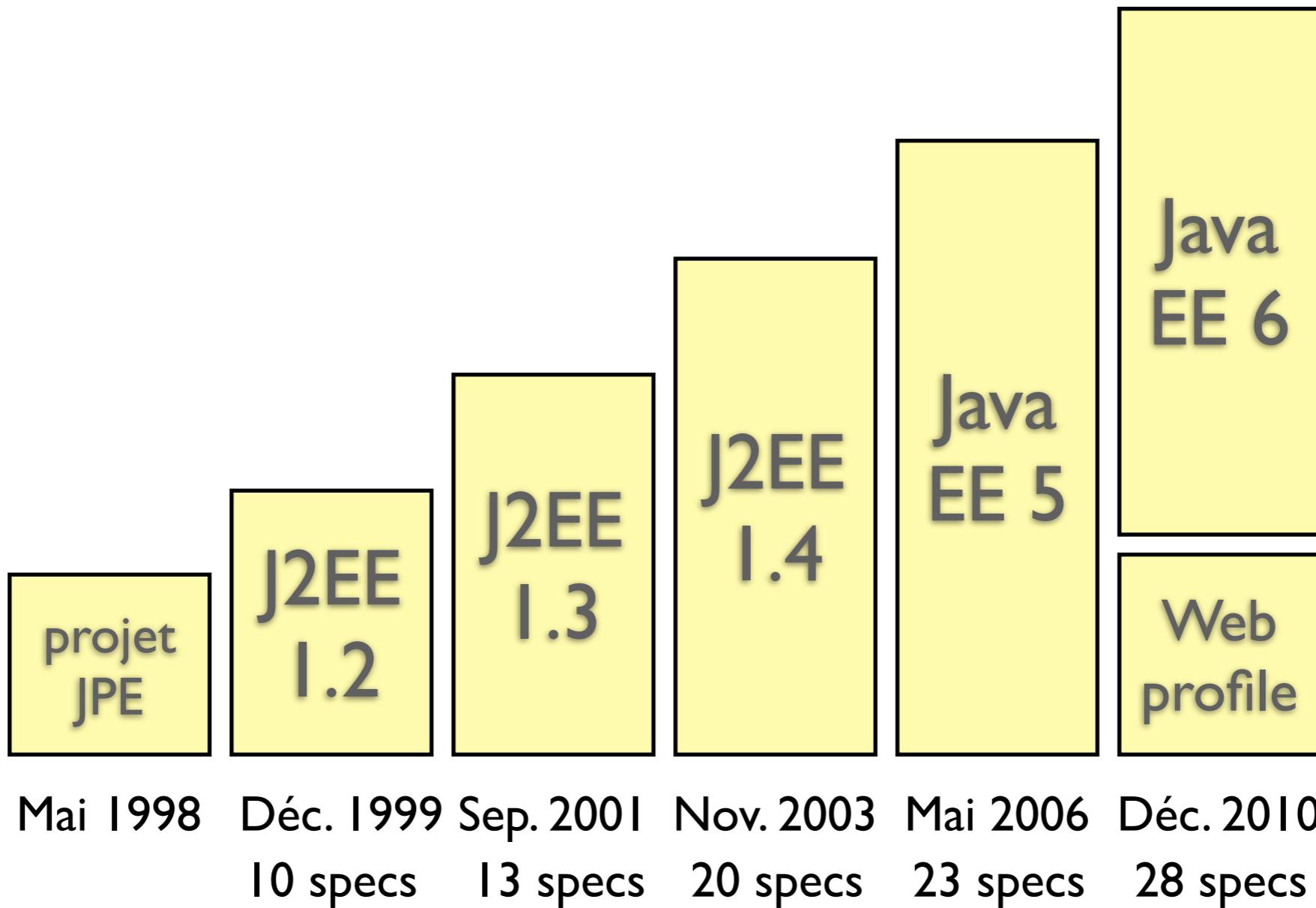
Un peu d'histoire



Un peu d'histoire



Un peu d'histoire



Java EE 6

- Plus riche
 - Plus de specs
 - Points d'extensions standards (1er)
- Plus facile
 - POJO
 - Moins de XML (configuration par exception)
- Plus léger
 - pruning
 - EJB Lite

Profils

- Possibilité de créer des ensembles cohérents de technologies
- Java EE, un profil parmi d'autres
- Web Profile
 - Servlet 3.0 / JSP 2.2 / EL 1.2 / JSTL 1.2 / EJB Lite 3.1 / JTA 1.1 / JPA 2.0 / JSF 2.0
- D'autres profils à venir
 - Minimal
 - Portail
 - ...

Pruning

- Rend certaines spécifications optionnelles
- Pour Java EE 6
 - Entity CMP 2.x
 - JAX-RPC
 - JAX-R
 - JSR 88
- Plus « fort » que l'annotation `@Deprecated`
- Plus facile pour l'écriture de nouveaux conteneurs

Java EE 6

- Composants métiers
- Accès aux données
- Couche de présentation
- Interopérabilité
- Autre

Java EE 6

- **Composants métiers**
- Accès aux données
- Couche de présentation
- Interopérabilité
- Autre

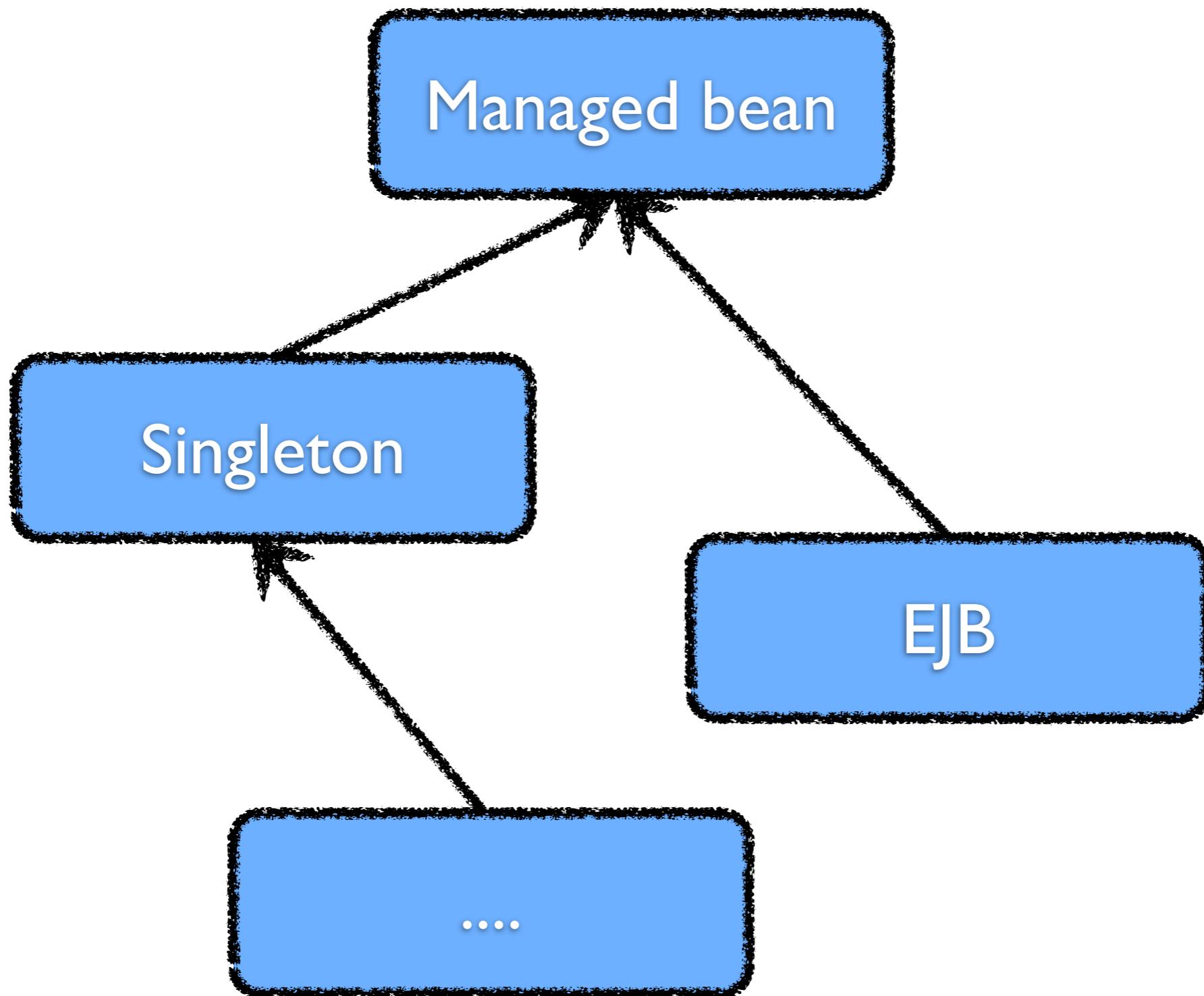
Le(s) conteneur(s)

- La plateforme Java EE fonctionne via un ensemble de conteneur
- Ils vont être garant de la gestion d'un type de composant
- Assureront les notions d'IoC, DI et AOP
- Mettent à disposition des ressources spécifiques
- Ce sont eux qui appelleront les bons composants suivant les requêtes des utilisateurs

Managed beans

- Composant de base d'une application Java EE 6
 - Géré par le conteneur
 - Simple et universel (POJO)
 - Si et seulement si il y a un fichier beans.xml dans META-INF ou WEB-INF
- Tout composant Java EE 6 est un managed bean plus ou moins spécialisé
- Peut-être défini par l'annotation `@ManagedBean`
- Supporte le nommage JNDI

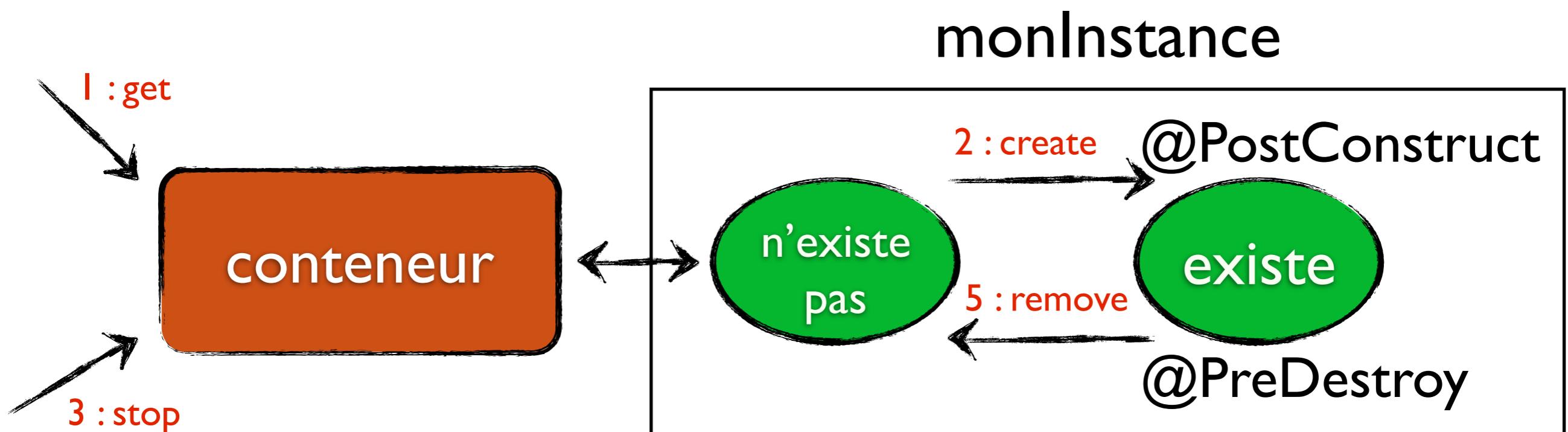
Managed beans



Managed beans

- Gestion du cycle de vie
 - callbacks liés à la création et la destruction des managed beans
 - Annotation sur des méthodes
 - `@PostConstruct`
 - `@PreDestroy`

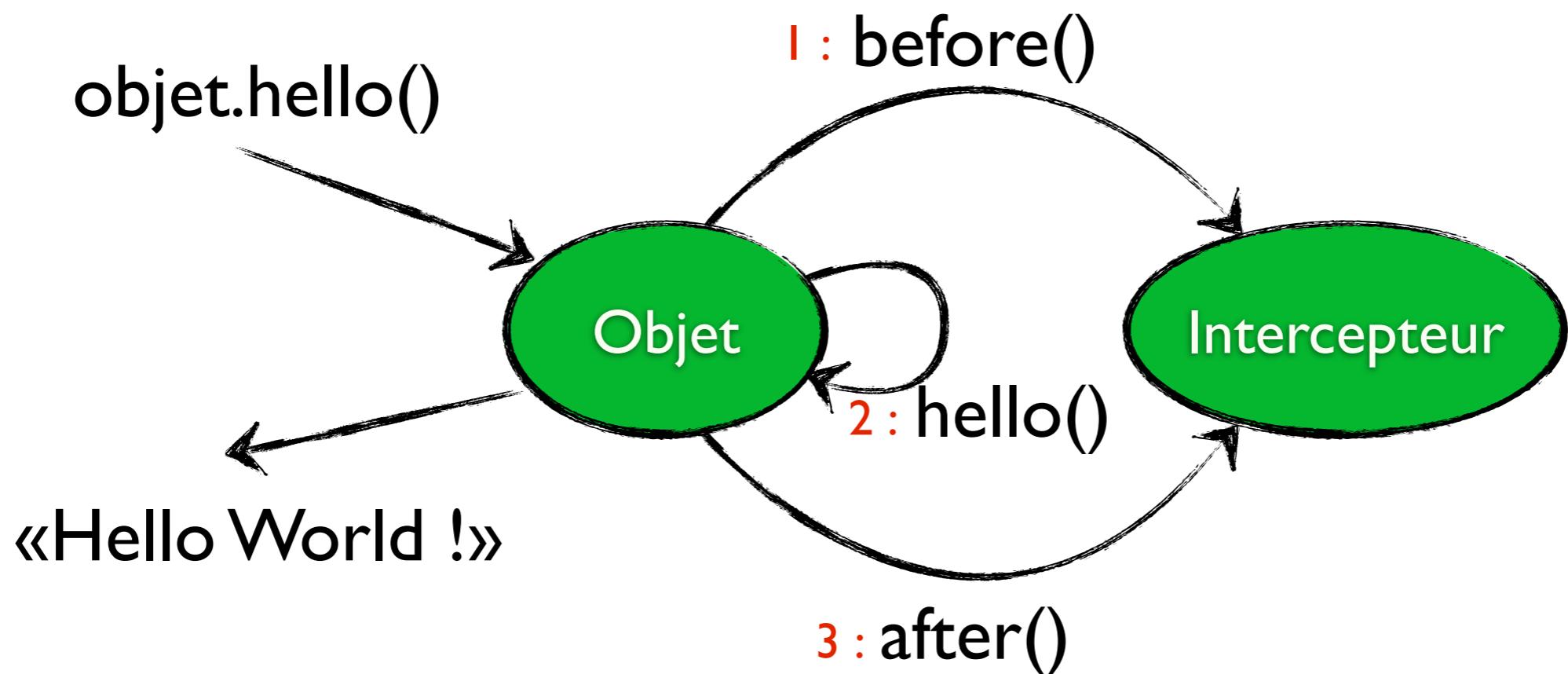
Managed beans



Managed beans

- Interception de méthodes
 - programmation AOP sur les composant
 - définition de code à appeler avant et après les méthodes du managed bean
 - permet de ne pas surcharger du code métier avec du code technique
 - défini au niveau de la classe ou de la méthode (`@Interceptors`)
 - `@AroundInvoke` sur les méthodes d'interception

Managed beans



CDI

- Context and Dependency Injection
- Ensemble de services permettant d'améliorer la structure du code applicatif
 - cycle de vie pour objets stateful
 - injection de dépendance typesafe
 - décoration des objets injectés
 - association d'intercepteurs à des objets
 - modèle de notifications par événements
 - contexte web conversationnel
 - extensions portables standard

@Inject

- Annotation d'injection de dépendance standard introduite par la JSR-330
- Peut-être apposée sur
 - un constructeur
 - un «setter»
 - un attribut

@Qualifier

- Annotation de qualification d'injection introduite par la JSR-330
- Va permettre de choisir l'injection d'une implémentation spécifique plutôt qu'une autre
- Annotation à créer, annotée par @Qualifier (méta annotation)
 - @Paypal, @LoggedIn,

@Produces

- Annotation standard pour la production d'un type de bean
- Permet de définir une méthode «factory» pour un type de bean
- configuration spécifique d'un bean
 - @New
 - choix de l'implémentation par rapport à l'injection
 - InjectionPoint

@Scope

- Annotation permettant de définir le contexte d'injection (méta annotation)
- Récupérer une instance spécifique par contexte
- Scopes en standard
 - @Dependent, @Singleton
 - @SessionScoped, @RequestScopped
 - @ApplicationScoped, @ConversationScoppped

@InterceptorBinding

- Permet de définir une annotation valide pour la définition d'un intercepteur (méta annotation)
- Classe d'interception annotée par `@Interceptor` et l'annotation annotée `@InterceptorBinding`
 - `@Secure`
- Annoter la classe / méthode métier
- Utile pour avoir de l'interception avec un sens adapté à la responsabilité

@Stereotype

- Meta annotation permettant de regrouper plusieurs annotations en une seule

@RequestScoped

@Named

@Secure

@Transactional

@Stereotype

@Target(TYPE)

@Retention(RUNTIME)

public @interface Action {}

@Decorator

- Application du pattern décorateur
- Une sorte d'AOP
- Doit être déclaré dans le fichier beans.xml
- Doit absolument comporter un point d'injection du type décoré annoté par
 @Delegate

@Observes

- Système évènementiel léger basé sur les annotations (traitements asynchrones)
- Injection d'évènements
 - `@Inject Event<MyEvent> e;`
- Lancer évènement
 - `e.fire(new MyEvent("Hello"));`
- Réception évènement
 - `public void receive(@Observes MyEvent e)`

SPI

- Ensemble de points d'extensions définis dans la spécification
- Permet d'étendre de manière portable et standard le cœur de Java EE
- Grand capacité d'intégration avec d'autres frameworks
 - découverte de beans, contexts, etc

EJB 3.1

- Enterprise JavaBeans
- Évite de se préoccuper avec :
 - Les transactions
 - La sécurité
 - La concurrence
 - La communication
 - La gestion des ressources
 - La gestion des erreurs

Structure

- Peut être livré sous forme d'un .jar standard
- Peut également être une classe comme une autre au sein d'un .war

EJB 3.1

- Il existe 2 types d'EJB
- EJB Session
 - Stateless
 - Stateful
- Message Driven Beans

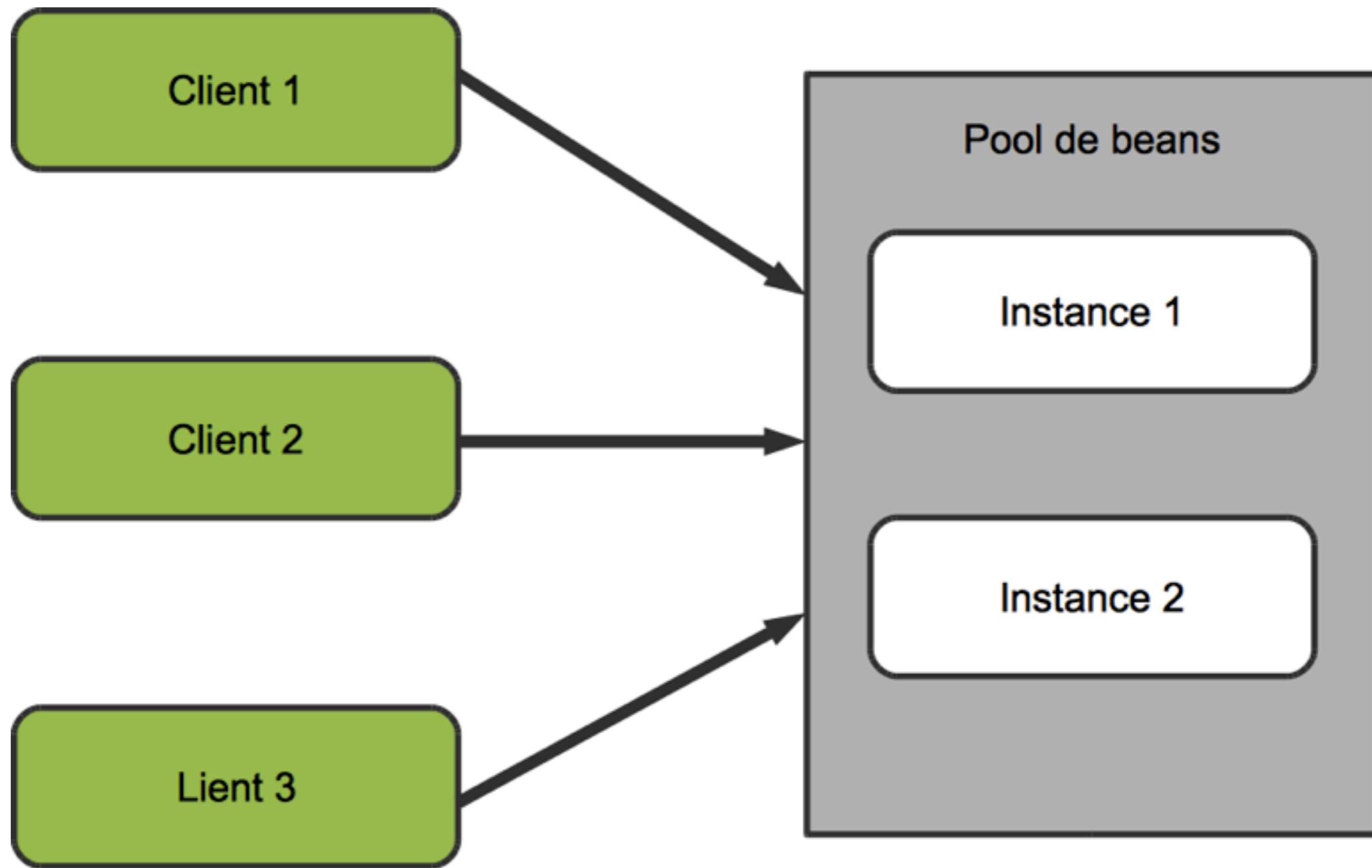
EJB Session

- Conçut pour encapsuler la logique métier
- EJB les plus utilisés
- Deux types
 - Stateless
 - Statefull

Stateless

- Stateless(sans état) → les attributs de l'EJB sont réinitialisés entre chaque appel même s'il s'agit du même client
- Sont spécialement pensés pour être robustes et fiables lorsqu'il y a beaucoup d'appels en concurrence
- Lorsqu'un client appelle l'EJB, une instance de ce dernier sert le client, puis, retourne dans le pool d'EJB (cette dernière est donc prête à être réutilisée pour un autre client)
- À utiliser le plus souvent possible (par rapport aux Stateful) → cycle de vie

Stateless



Stateless

```
1 package stateless;
2
3
4 import javax.ejb.Local;
5
6 @Local
7 public interface Hello {
8     void sayHello();
9 }
10
```

interface «locale»
non obligatoire

```
1
2 package stateless;
3
4 import javax.ejb.Stateless;
5
6 @Stateless
7 public class HelloBean implements Hello {
8     public void sayHello() {
9         System.out.println("Hello World!");
10    }
11 }
12
```

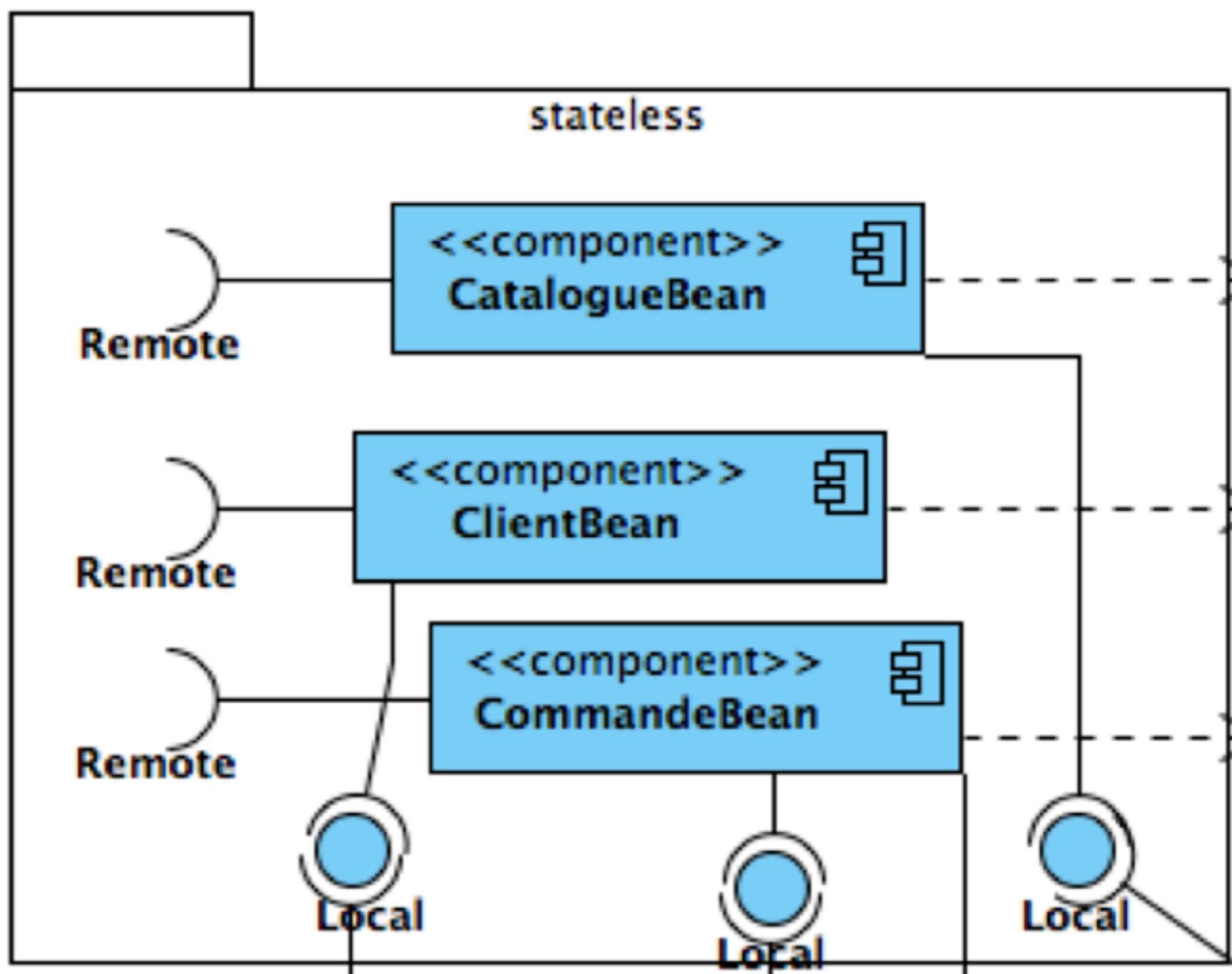
Stateless

```
1 package stateless;
2
3
4 import javax.ejb.EJB;
5
6 public class HelloUser {
7
8     @EJB
9     private Hello hello;
10
11    public void sayHelloWorld (){
12        hello.sayHello();
13    }
14}
15
```

Cas d'étude

- Les stateless beans vont être utilisés :
 - Pour la gestion du catalogue
 - Pour la gestion des utilisateurs
 - Pour la gestion des commandes
- Ce sont les composants qui vont-être les plus sollicitées, l'usage de stateless beans et donc fortement recommandé.

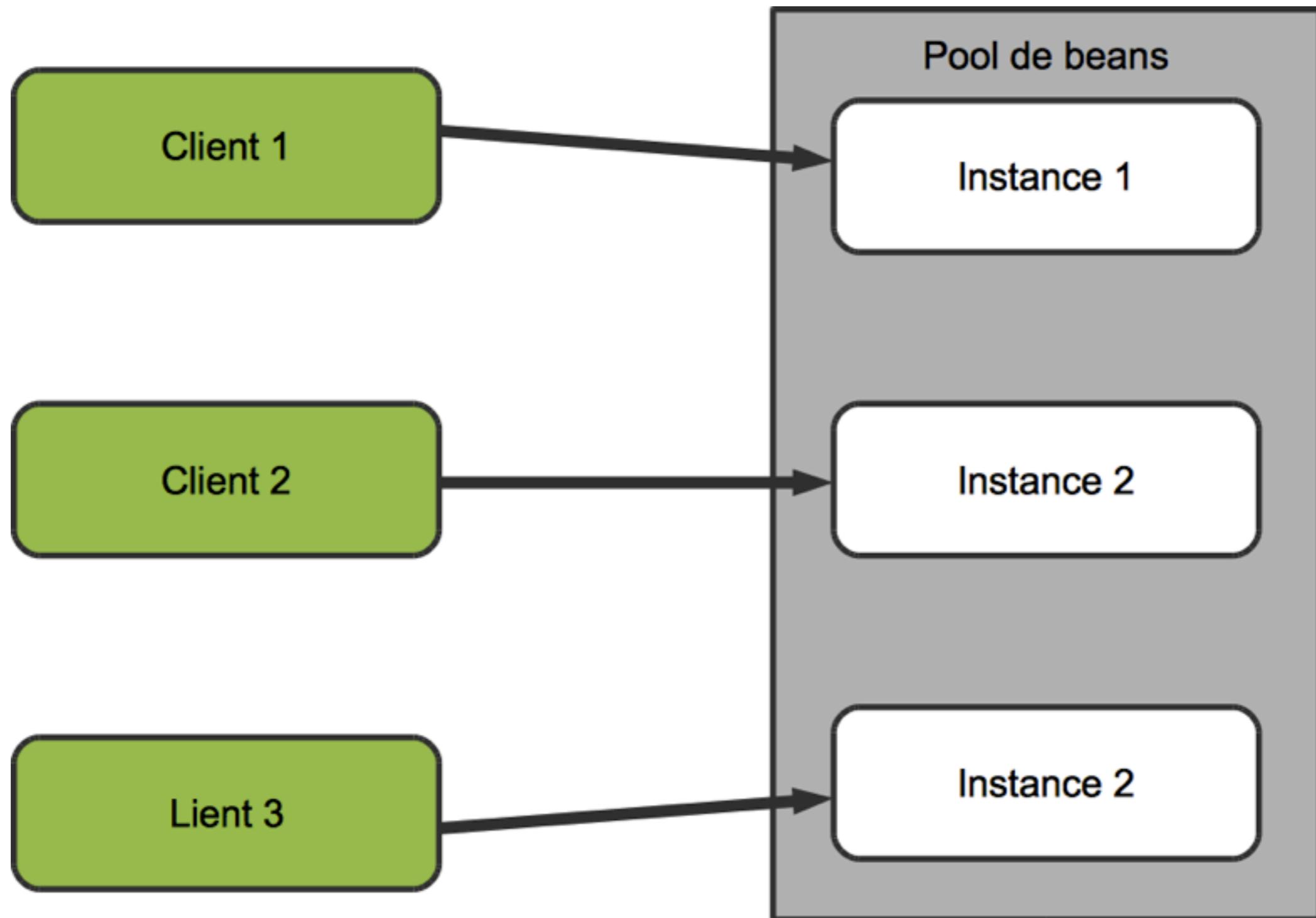
Cas d'étude



Stateful

- Stateful (avec état) → les attributs de l'EJB sont sauvegardés durant toute la session
- Lorsqu'un client appelle l'EJB, une instance de ce dernier est créée, puis sert le client. Cette instance reste disponible pour les futurs appels de ce client uniquement. Cette instance sera détruite à la fin de la session (timeout ou appel à une méthode portant l'annotation `@Remove`)
- S'il y a trop d'instances d'un EJB en mémoire, ces dernières peuvent être sorties de la mémoire de travail. Elles passent ainsi en mode passif (sauvées sur disque → tous les attributs doivent être sérialisables)

Stateful



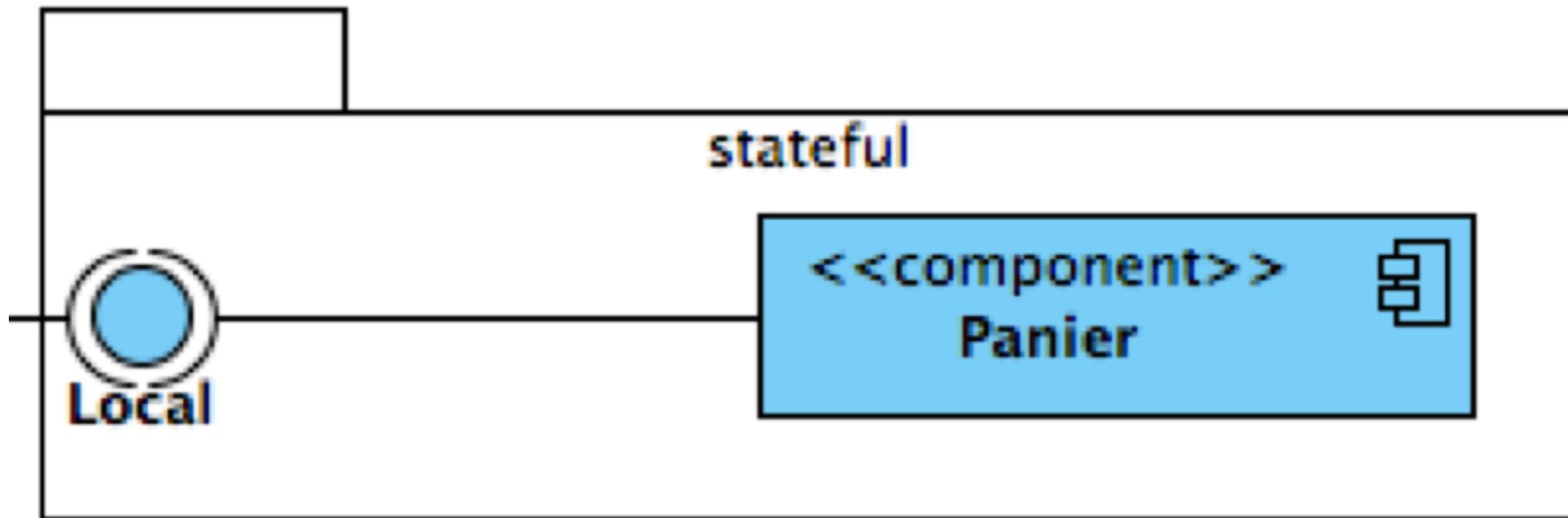
Stateful

```
2 package stateful;
3
4 import javax.ejb.Local;
5
6 @Local
7 public interface Hello {
8     void sayHello();
9     void sayHello(String name);
10 }
11
12
13
14
15
16
17
18
19 }
```

Cas d'étude

- Les stateful session beans vont être utiles dans notre application au niveau de la session d'achats
- Gestion du panier
 - L'utilisateur doit pouvoir garder ses produits dans le panier le temps de ses achats
 - C'est la validation de la commande qui va vider le panier

Cas d'étude



Local vs. Remote

- Spécification de la manière d'accéder à l'EJB → l'implémentation des services reste identique
- Local
 - Appel à un EJB se trouvant sur la même JVM
- Remote
 - Utilise RMI et JNDI pour accéder à l'EJB
 - Plus lent qu'un appel local

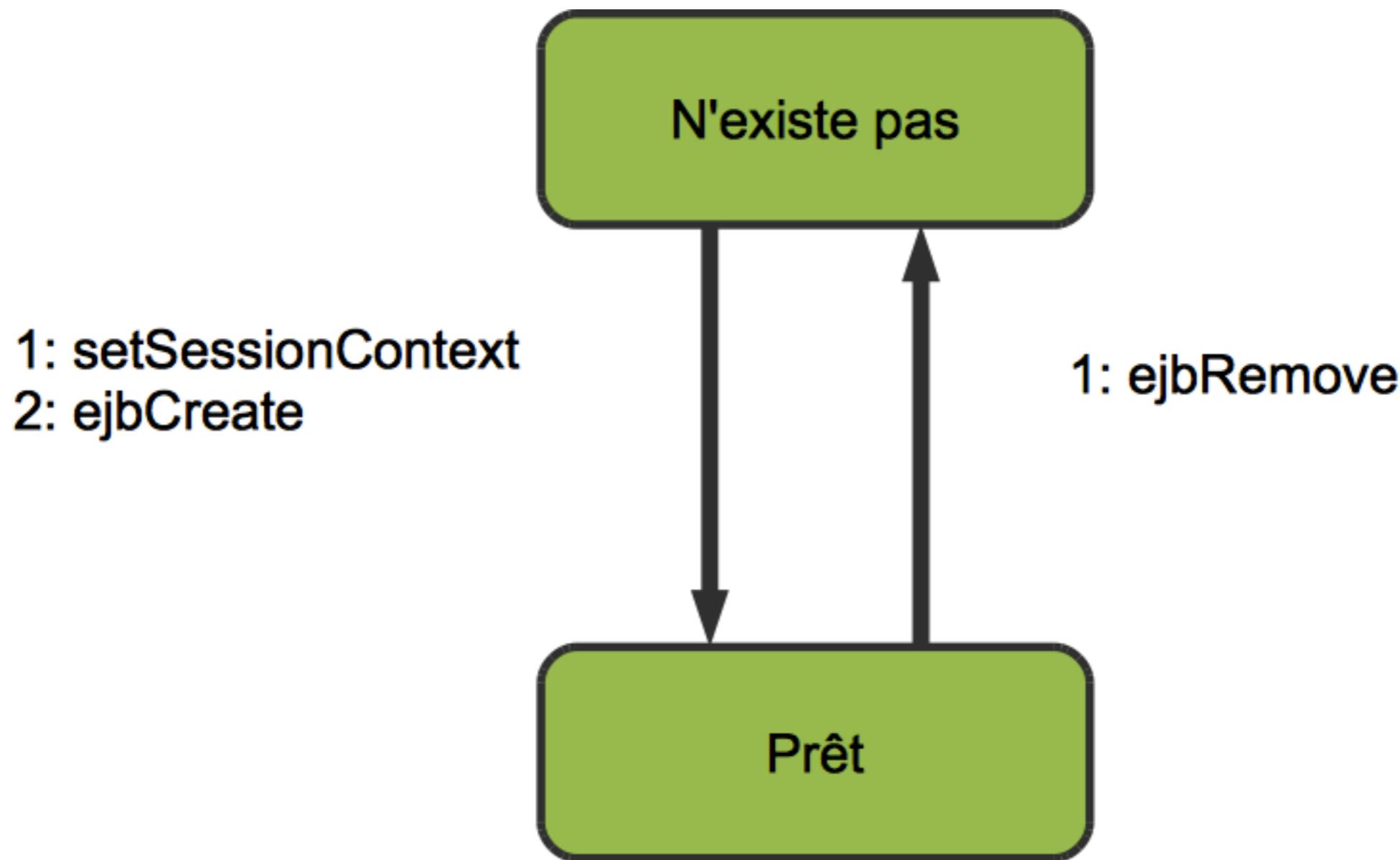
Local vs. Remote

```
6 @Remote  
7 public interface Hello {  
8     void sayHello();  
9 }  
  
6 @Stateless(mappedName = "ejb>HelloBean")  
7 public class HelloBean implements Hello {  
8     public void sayHello() {  
9         System.out.println("Hello World!");  
10    }  
11 }  
  
8 public class HelloUser {  
9  
10    private void sayHello() throws Exception {  
11        Context initialContext = new InitialContext(new Properties());  
12  
13        Object o = initialContext.lookup("ejb>HelloBean");  
14        if (o instanceof Hello) {  
15            ((Hello) o).sayHello();  
16        }  
17        throw new Exception("Cannot cast object");  
18    }  
19 }
```

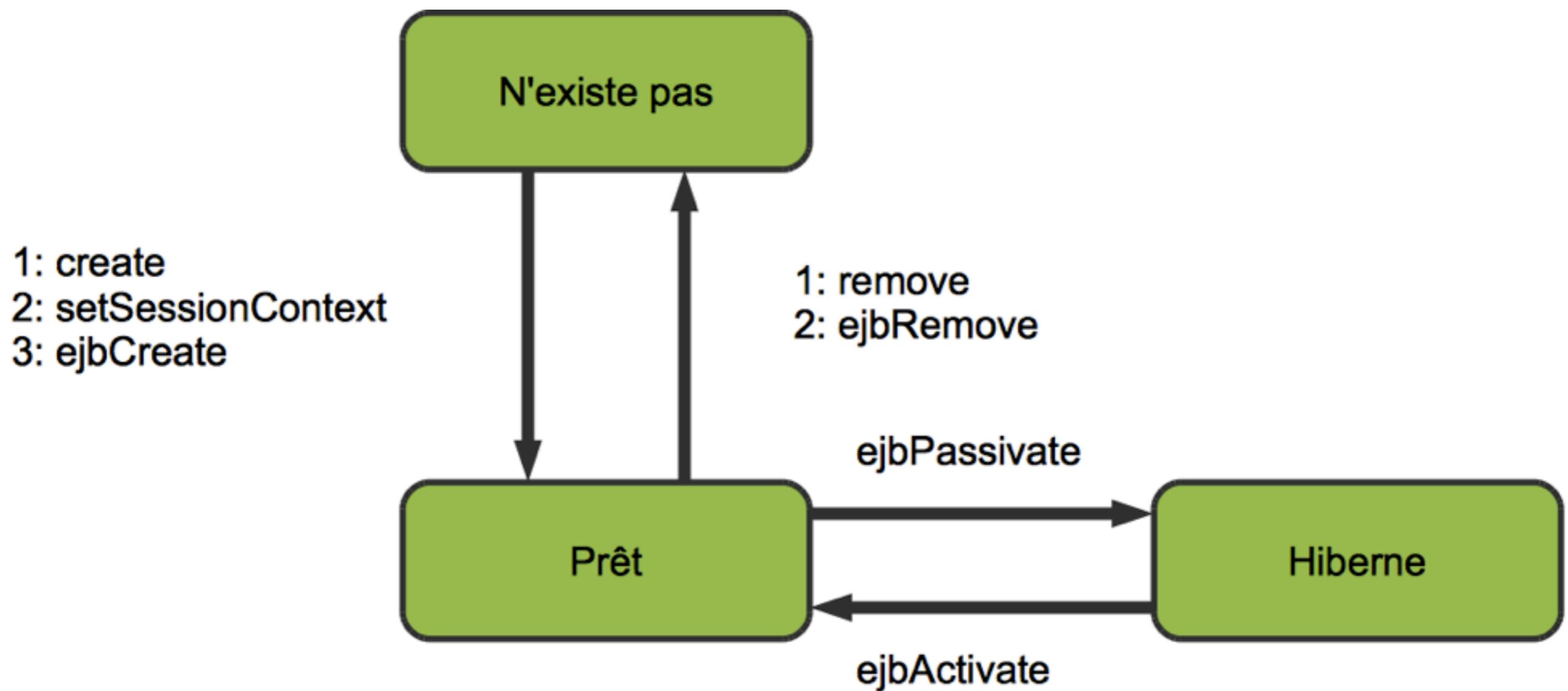
Cycle de vie

- Les EJB étant des managed beans, ils supportent les callbacks sur le cycle de vie
- Après la construction (@PostConstruct)
- Avant la destruction (@PreDestroy)
- Avant hibernation (@PrePassivate) → Stateful
- Après hibernation (@PostActivate) → Stateful

Cycle de vie



Cycle de vie



@Singleton

- Type particulier d'EJB session
- Stateful by design
- Une seule instance dans le conteneur
- Attention : différent du @Singleton
JSR-330 / CDI

@Asynchronous

- Annotation sur une méthode EJB
- Uniquement si le type de retour est void ou Future<T>
- Retour de la méthode immédiat
- Traitement en asynchrone
- Retour un résultat via Future

@Schedule

- Annotation sur une méthode d'un EJB
- Cette méthode sera appelée périodiquement
- Valeur de l'annotation comparable à une expression cron
- Ne fonctionne pas en statefull

Conteneur embarquable

- Possibilité d'utiliser le conteneur EJB 3.1 en mode embarqué
 - `EJBContainer.createEJBContainer();`
- Très utile pour effectuer des tests unitaires dans un mode purement EJB
 - Sécurité
 - Transactions

MDB

- EJB piloté par messages
- Programmation faiblement couplée
- Programmation asynchrone
- Concept de Message-Oriented middleware
 - MOM
- Utilisation de JMS au niveau du conteneur

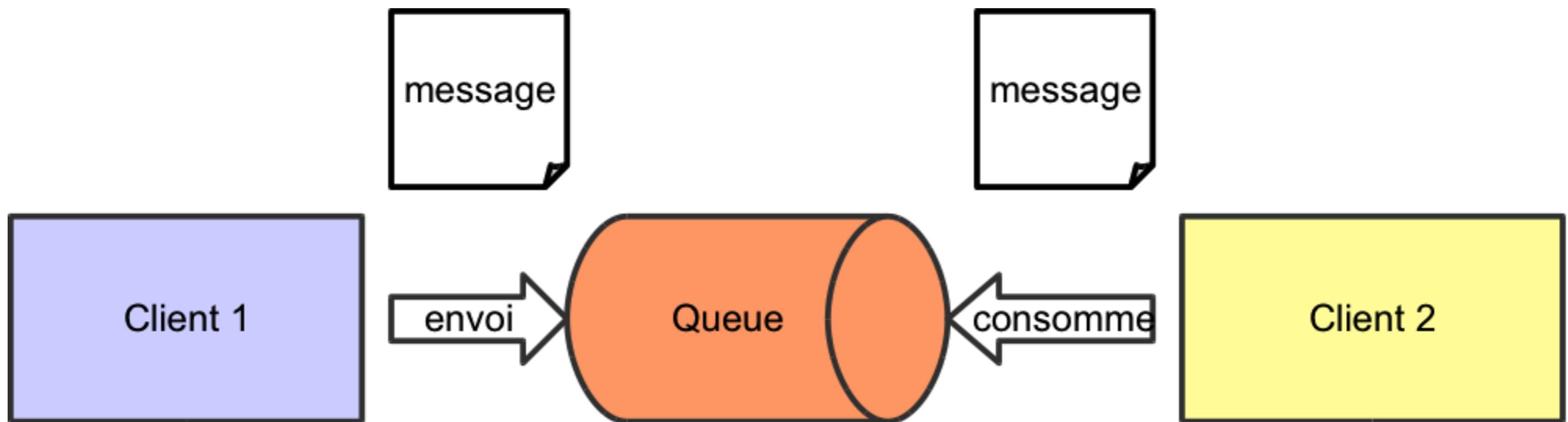
JMS

- Java Message System → JSR-914
- Spécification permettant la manipulation de messages distribués
- Programmation faible couplage
- Consommation synchrone ou asynchrone
- Distribution fiable
 - Un message est garanti d'être délivré une et seulement une fois

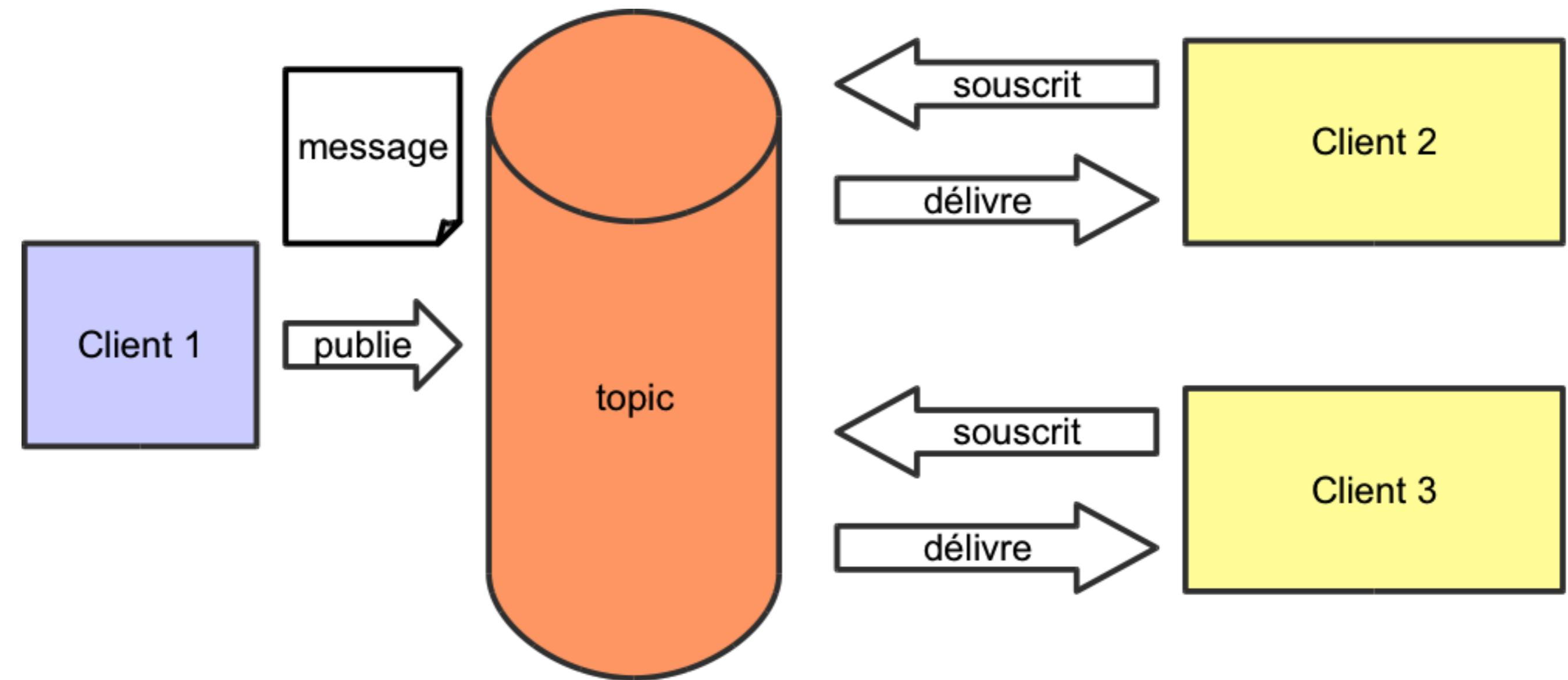
Types de communication

- Point-to-point
 - Construit autour du concept de files (queues) de messages
 - Chaque message n'a qu'un seul consommateur
- Publish-Suscribe
 - Utilise un “topic” pour envoyer ou recevoir des messages
 - Chaque message peut avoir plusieurs consommateurs

Point-to-point



Publish-suscribe



JMS message

- Message Header
 - Utilisé pour l'identification et le routage des messages
 - Données spécifiques au distributeur du système de message et à l'application
 - Typiquement ce sont des paires nom/valeur
- Message Properties (optional)
- Message Body (optional)
 - Contient les données du message
 - Différents types possibles

Types de messages

- TextMessage
- MapMessage
- BytesMessage
- StreamMessage
- ObjectMessage

Configuration

```
20  private Session session;
21  private Context jndiContext;
22  private ConnectionFactory connectionFactory;
23  private Connection connection;
24  private Queue queue1;
25  private MessageProducer producer;
26  private MessageConsumer consumer;
27  private MessageListener listener;
28
29  public void configure() {
30      try {
31          jndiContext = new InitialContext();
32          connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/ConnectionFactory");
33          queue1 = (Queue) jndiContext.lookup("jms/TestQueue");
34          connection = connectionFactory.createConnection();
35          session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
36          producer = session.createProducer(queue1);
37          consumer = session.createConsumer(queue1);
38          listener = new MyJMSListener();
39          consumer.setMessageListener(listener);
40      } catch (Exception ex) {
41          Logger.getLogger(JMSExemple.class.getName()).log(Level.SEVERE, null, ex);
42      }
43  }
```

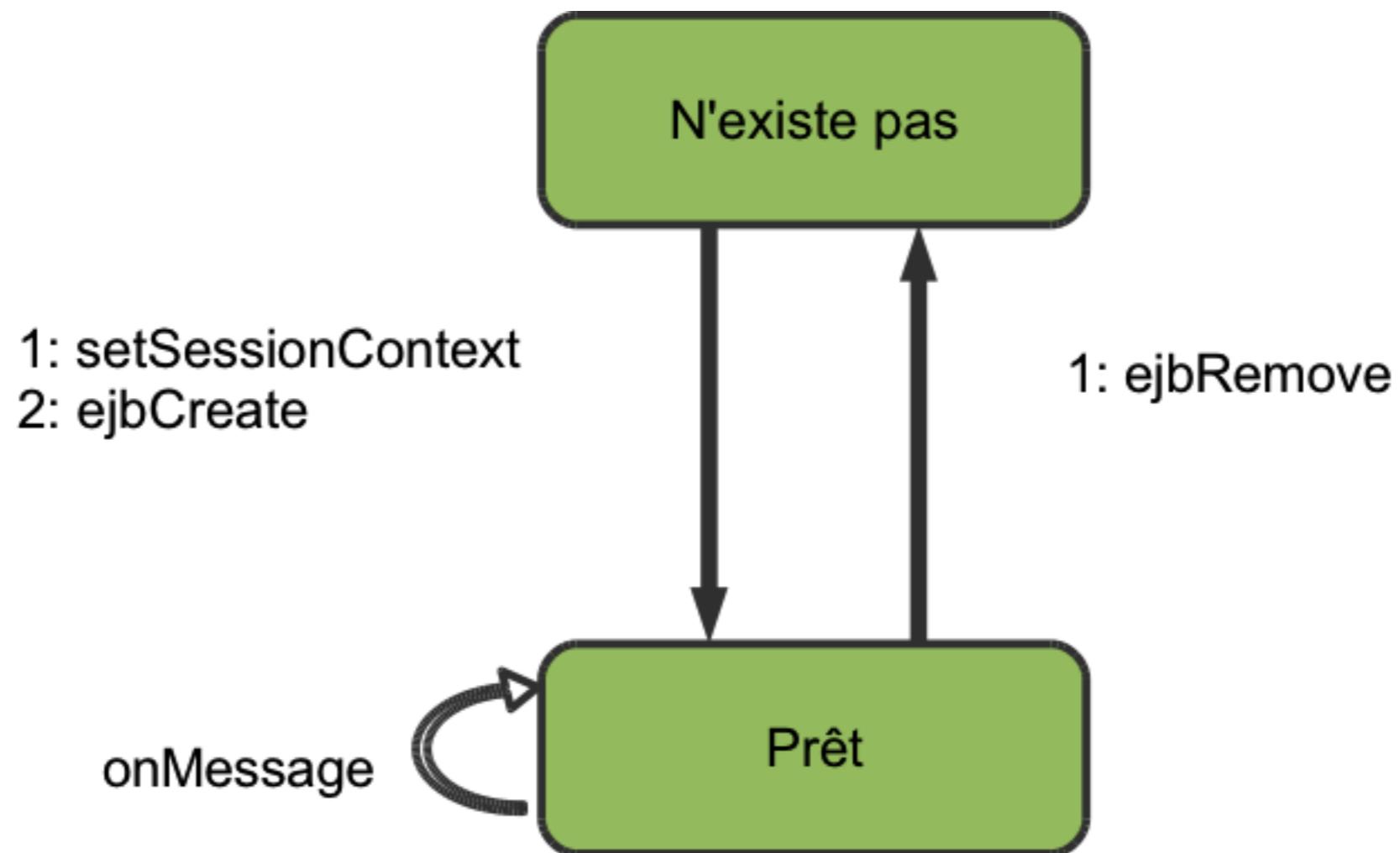
Utilisation

```
45  public void send() {
46      try {
47          TextMessage m = session.createTextMessage();
48          m.setText("just another message");
49          producer.send(m);
50      } catch (JMSEException ex) {
51          Logger.getLogger(JMSEExemple.class.getName()).log(Level.SEVERE, null, ex);
52      }
53  }
54
55  public void receive() {
56      try {
57          Message message = consumer.receive();
58          if (message instanceof TextMessage) {
59              System.out.println("Reception synchrone : "
60                  + ((TextMessage) message));
61          }
62      } catch (JMSEException ex) {
63          Logger.getLogger(JMSEExemple.class.getName()).log(Level.SEVERE, null, ex);
64      }
65  }
66
67  private class MyJMSListener implements MessageListener {
68      public void onMessage(Message message) {
69          if (message instanceof TextMessage) {
70              System.out.println("Reception asynchrone : "
71                  + ((TextMessage) message));
72          }
73      }
74  }
```

MDB

```
11 @MessageDriven(activationConfig = {  
12     @ActivationConfigProperty(propertyName = "destination",  
13         propertyValue = "SampleQueue"),  
14     @ActivationConfigProperty(propertyName = "destinationType",  
15         propertyValue = "javax.jms.Queue"))  
16 public class MessageDrivenBean implements MessageListener {  
17  
18     ①     public void onMessage(final Message message) {  
19         String txt = "Message nommé " + message;  
20         if (message instanceof TextMessage) {  
21             try {  
22                 txt += " avec le contenu "  
23                     + ((TextMessage) message).getText();  
24             } catch (JMSEException ex) {  
25                 System.out.println(ex);  
26             }  
27         }  
28         System.out.println(txt);  
29     }  
30 }
```

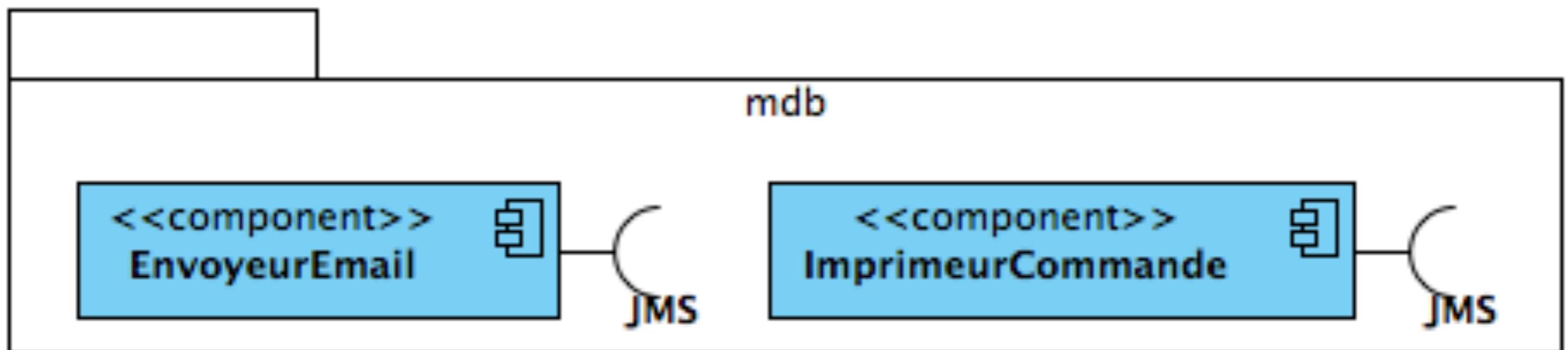
Cycle de vie



Cas d'étude

- Traitements asynchrones de l'application
 - Envoyer les emails
 - Imprimer les bons de commande
- Utilisation du mode publish-suscribe
- En plus : système d'alarmes visuelles dans le client d'administration pour de grosses commandes ?

Cas d'étude



EJB Lite

- Sous ensemble de l'api EJB 3.1
- Utilisé dans le web profile
- Fonctionnalités
 - local session beans
 - persistance + JTA
 - sécurité
 - injection de dépendance
 - intercepteurs

Java EE 6

- Composants métiers
- **Accès aux données**
- Couche de présentation
- Interopérabilité
- Autre

JPA 2

- Java Persistence API
- Gère la persistance sur le concept d'object-relationnal mapping (ORM) → illusion de travailler avec une BD objet
- Management de la persistances des objets en base
- Faisait partie de la spécification EJB 3.0 (JSR-220)
- JPA 2 dans une spécification autonome (JSR-317)

@Entity

- Attaché à une classe
 - Signifie qu'une classe est persistante
 - Pour avoir des attributs persistants
 - Suivre la convention JavaBeans

```
8  @Entity
9  public class Person implements Serializable {
10
11     @Id
12     private Long id;
13
14     private String nom;
15
16     public String getNom() {
17         return nom;
18     }
19
20     public void setNom(String nom) {
21         this.nom = nom;
22     }
23 }
24 }
```

@Id

- Chaque entity doit posséder un ID
- L'ID peut être généré automatiquement
 - `@Id(generate=GeneratorType.AUTO)`
- La stratégie de génération AUTO est la plus “portable”
- Autres stratégies possibles
 - `GeneratorType.SEQUENCE`
 - `GeneratorType.IDENTITY`

ORM

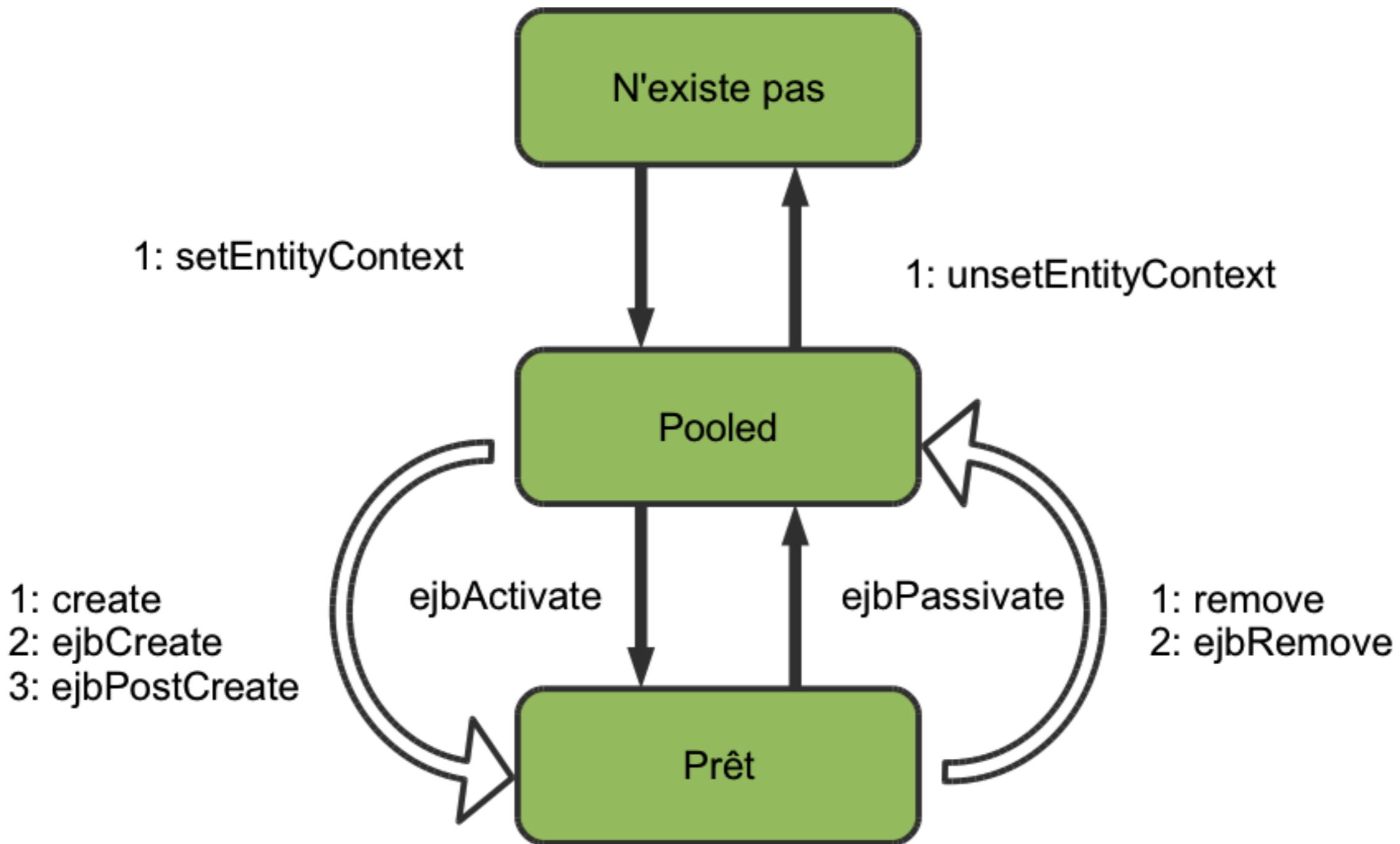
- Quatre relations possibles
 - `@OneToOne`
 - `@OneToMany`
 - `@ManyToOne`
 - `@ManyToMany`
- Dans la majorité des cas, l'annotation suffit
- Pour les autres, il y aura un peu de paramétrage à faire

ORM

Cycle de vie

- Comme pour les EJB session et les Message-Driven, il existe des callbacks pour les Entities
- Avant / après la persistance d'une Entity(@PrePersist / @PostPersist)
- Après avoir chargé une Entity lors d'une requête ou d'un refresh(@PostLoad)
- Avant / après la mise à jour d'une Entity(@PreUpdate / @PostUpdate)
- Avant / après la suppression d'une Entity(@PreRemove / @PostRemove)

Cycle de vie



Entity Manager

- Passerelle vers les objets en base de données
- Permet l'utilisation de requêtes JPQL
- En dehors des sessions beans, l'entity manager fournit des facilités à l'utilisation de transactions
- Configuré via un fichier XML
- META-INF/persistence.xml

persistence.xml

```
3  <persistence version="1.0"
4      xmlns="http://java.sun.com/xml/ns/persistence">
5      <persistence-unit name="entity" transaction-type="JTA">
6          <provider>org.hibernate.ejb.HibernatePersistence</provider>
7          <jta-data-source>jdbc/geolocationDS</jta-data-source>
8          <class>org.ow2.util.geolocation.entity.Geolocation</class>
9          <class>org.ow2.util.geolocation.entity.ProductName</class>
10         <class>org.ow2.util.geolocation.entity.ProductVersion</class>
11         <class>org.ow2.util.geolocation.entity.ManagedProduct</class>
12         <properties>
13             <property name="hibernate.dialect"
14                 value="org.hibernate.dialect.PostgreSQLDialect"/>
15             <property name="hibernate.hbm2ddl.auto" value="update"/>
16         </properties>
17     </persistence-unit>
18 </persistence>
```

Entity Manager

```
10 @PersistenceContext  
11 private EntityManager entityManager;  
12  
13 public void creerPersonne() {  
14     Adresse adresse = new Adresse();  
15     adresse.setValue("Technoforum 17071 LA ROCHELLE");  
16     Personne personne = new Personne();  
17     personne.setNom("Maurice ANDRE");  
18     personne.setAdresse(adresse);  
19     entityManager.persist(adresse);  
20     entityManager.persist(personne);  
21     entityManager.flush();  
22 }
```

Requêtes

```
10 @PersistenceContext  
11 private EntityManager entityManager;  
12  
13 public Collection<Personne> trouverPersonne(Long id) {  
14     List personnesList =  
15         entityManager.createQuery(  
16             "SELECT p FROM Personne AS p WHERE p.id "  
17             + "LIKE :custId").setParameter("custId", id).getResultList();  
18     return personnesList;  
19 }
```

Criteria

- Utilisée pour la définition de requêtes dynamiques via une API purement programmatique
 - Fortement typée
 - Plus aucune chaîne de caractères
 - reste tout de même possible
- Utilise des métamodèles
 - Chaque entité X possède un métamodèle nommé `class X`
 - Génération par Annotation Processor
- Peut être utilisé sans métamodèle

Criteria

```
7  @Entity
8  public class Customer {
9      @Id
10     private Integer custId;
11     private String name;
12     private String shippingAddress;
13
14     public Integer getCustId() {...}
15     public void setCustId(Integer custId) {...}
16     public String getName() {...}
17     public void setName(String name) {...}
18     public String getShippingAddress() {...}
19     public void setShippingAddress(String shippingAddress) {...}
20 }
21
22
23
24
25
26
27
28
29
30
31
32 }
```

```
10  public class CriteriaTest {
11      @PersistenceContext
12      private EntityManager entityManager;
13
14      public Collection<Customer> find() {
15          QueryBuilder queryBuilder = entityManager.getQueryBuilder();
16          CriteriaQuery query = queryBuilder.createQuery();
17          Root<Customer> customer = query.from(Customer.class);
18          query.select(customer.get(Customer_.shippingAddress))
19              .where(queryBuilder.equal(customer.get(Customer_.name), "Peter"));
20          return entityManager.createQuery(query).getResultList();
21      }
22  }
```

Bean Validation

- JSR-303, inspirée de Hibernate Validator
- Permet de faire de la validation déclarative
- Définition d'une contrainte pour valider partout
 - Tiers de présentation
 - Tiers business
 - Tiers de données
- Contrainte → extension du typage Java

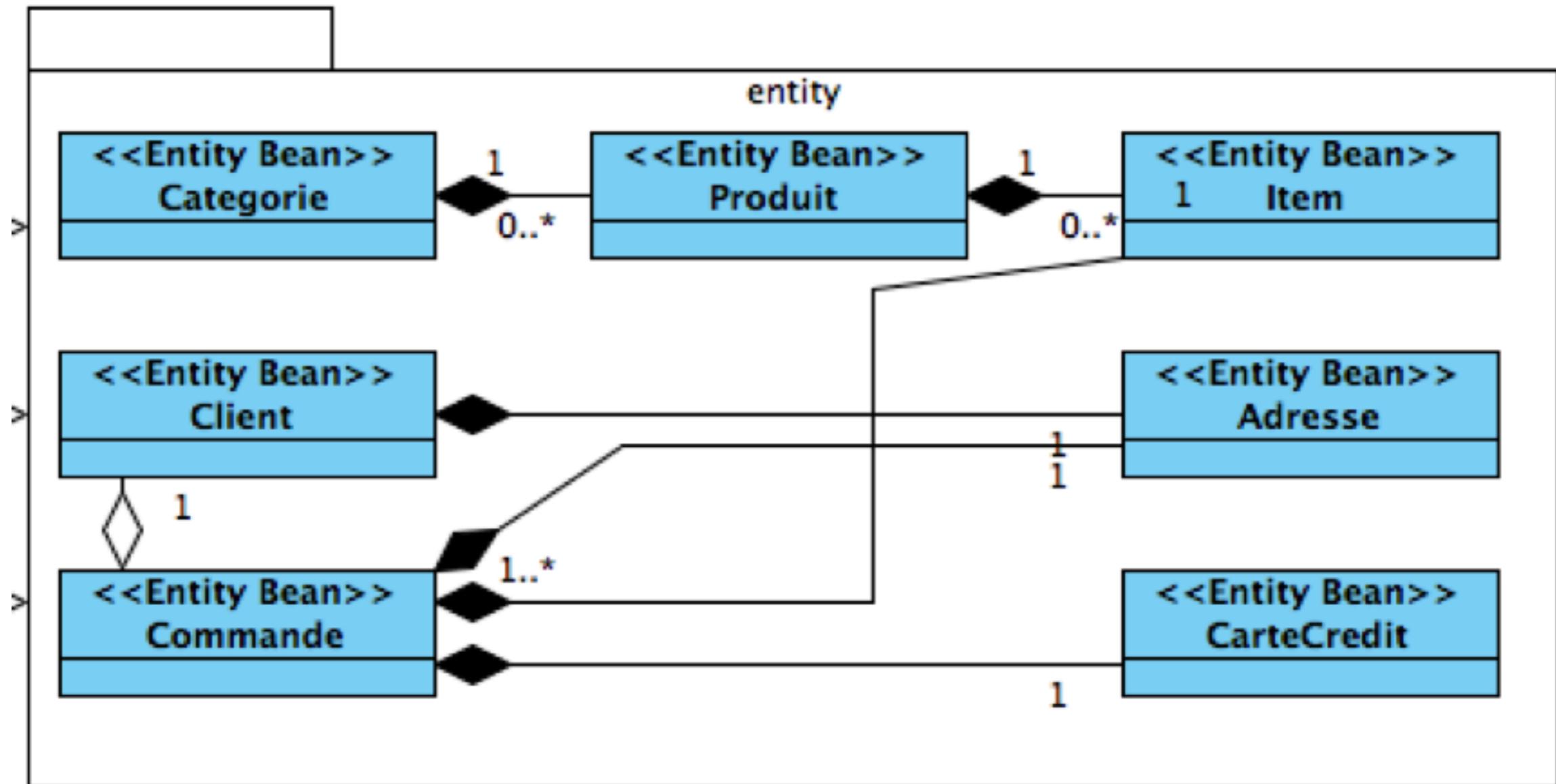
Bean Validation

- Contraintes par annotation sur les entités
 - `@NotNull`
 - `@Null`
 - `@Size`
 - `@Min, @Max`
 - `@Pattern`
 - contraintes customisées

Cas d'étude

- Toute la partie données de l'application est concernée
- Les entités possibles sont :
 - Client
 - Produit
 - Commande
 - Catégorie
 - Adresse
 - CarteDeCrédit

Cas d'étude



Transactions

- Une transaction est une série d'opérations qui apparaissent comme une une grosse opération atomique
- Propriété d'ACIDité
 - (A)tomicité
 - Opérations indispensables donc forment un tout atomique
 - (C)ohérence
 - La base est toujours dans un état cohérent
 - (I)solation
 - Pas de dépendances entre les transactions simultanées
 - (D)urabilité
 - Les modifications sont définitives

JTA

- Java Transaction API → JSR-907
- JTA est une API de haut niveau, indépendant de l'implémentation et du protocole qui permet l'accès aux transactions pour des applications ou des serveurs d'applications
- Avec Java EE, la gestion des transaction est déléguée au conteneur (CMT)
- Cependant, possibilité de les gérer “à la main” (BMT)

BMT

```
18     private InitialContext context;
19     private UserTransaction uts;
20
21     public void configure() {
22         try {
23             context = new InitialContext();
24             uts = (UserTransaction) context.lookup("UserTransaction");
25         } catch (NamingException ex) {
26             Logger.getLogger(BMT.class.getName()).log(Level.SEVERE, null, ex);
27         }
28     }
29
30     public void transferer() {
31         configure();
32         try {
33             uts.begin(); // début de la transaction
34             transfersDeFonds(10000000, EURO)
35                 .depuis(MON_COMPTE_EN_FRANCE)
36                 .vers(MON_COMPTE_EN_SUISSE);
37             transfersDeFonds(30000000, EURO)
38                 .depuis(MON_COMPTE_EN_FRANCE)
39                 .vers(MON_COMPTE_AUX_CAIMANS);
40             uts.commit(); // validation de la transaction
41         } catch (Exception e) {
42             try {
43                 uts.rollback(); // invalide ce qui a été fait
44             } catch (IllegalStateException ex) {
45                 Logger.getLogger(BMT.class.getName()).log(Level.SEVERE, null, ex);
46             } catch (SecurityException ex) {
47                 Logger.getLogger(BMT.class.getName()).log(Level.SEVERE, null, ex);
48             } catch (SystemException ex) {
49                 Logger.getLogger(BMT.class.getName()).log(Level.SEVERE, null, ex);
50             }
51         }
52     }
```

BMT + DI

```
18 @Resource
19 private SessionContext sctx;
20
21 public void transferer() {
22     UserTransaction uts = sctx.getUserTransaction();
23     try {
24         uts.begin(); // début de la transaction
25         transfersDeFonds(10000000, EURO)
26             .depuis(MON_COMPTE_EN_FRANCE)
27             .vers(MON_COMPTE_EN_SUISSE);
28         transfersDeFonds(30000000, EURO)
29             .depuis(MON_COMPTE_EN_FRANCE)
30             .vers(MON_COMPTE_AUX_CAIMANS);
31         uts.commit(); // validation de la transaction
32     } catch (Exception e) {
33         try {
34             uts.rollback(); // invalide ce qui a été fait
35         } catch (IllegalStateException ex) {
36             Logger.getLogger(BMTDI.class.getName()).log(Level.SEVERE, null, ex);
37         } catch (SecurityException ex) {
38             Logger.getLogger(BMTDI.class.getName()).log(Level.SEVERE, null, ex);
39         } catch (SystemException ex) {
40             Logger.getLogger(BMTDI.class.getName()).log(Level.SEVERE, null, ex);
41         }
42     }
43 }
```

CMT

- Politiques transactionnelles
 - REQUIRED
 - Crée une nouvelle transaction si aucune commencée
 - REQUIRED_NEW
 - Crée une nouvelle transaction
 - MANDATORY
 - Nécessite une transaction (l'appelant doit en avoir une)
 - NOT_SUPPORTED
 - Ne crée, ni ne propage de transaction
 - SUPPORTS
 - Utilise la transaction en cours
 - NEVER

CMT

```
7  @Stateless
8  @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
9  public class TransfersBean implements TransfersRemote {
10
11     private final static int EURO = 1;
12     private final static int MON_COMPTE_EN_FRANCE = 20987678;
13     private final static int MON_COMPTE_EN_SUISSE = 56787654;
14     private final static int MON_COMPTE_AUX_CAIMANS = 98767897;
15
16     (1)  public void transferer() {
17         transfersDeFonds(10000000, EURO)
18             .depuis(MON_COMPTE_EN_FRANCE)
19             .vers(MON_COMPTE_EN_SUISSE);
20         transfersDeFonds(30000000, EURO)
21             .depuis(MON_COMPTE_EN_FRANCE)
22             .vers(MON_COMPTE_AUX_CAIMANS);
23     }
24
25     (1)  public TransfersBean transfersDeFonds(double somme, int monnaie) {...}
26
27     (1)  public TransfersBean depuis(int numCompte) {...}
28
29     (1)  public TransfersBean vers(int numCompte) {...}
30
31     }
32
33     }
34
35     }
36 }
```

Cas d'étude

- Utilisation des transaction aux niveau des SessionBeans
- Transactions de type REQUIRED
- Transactions gérées par le conteneur EJB
- Application à toute méthode “métier”

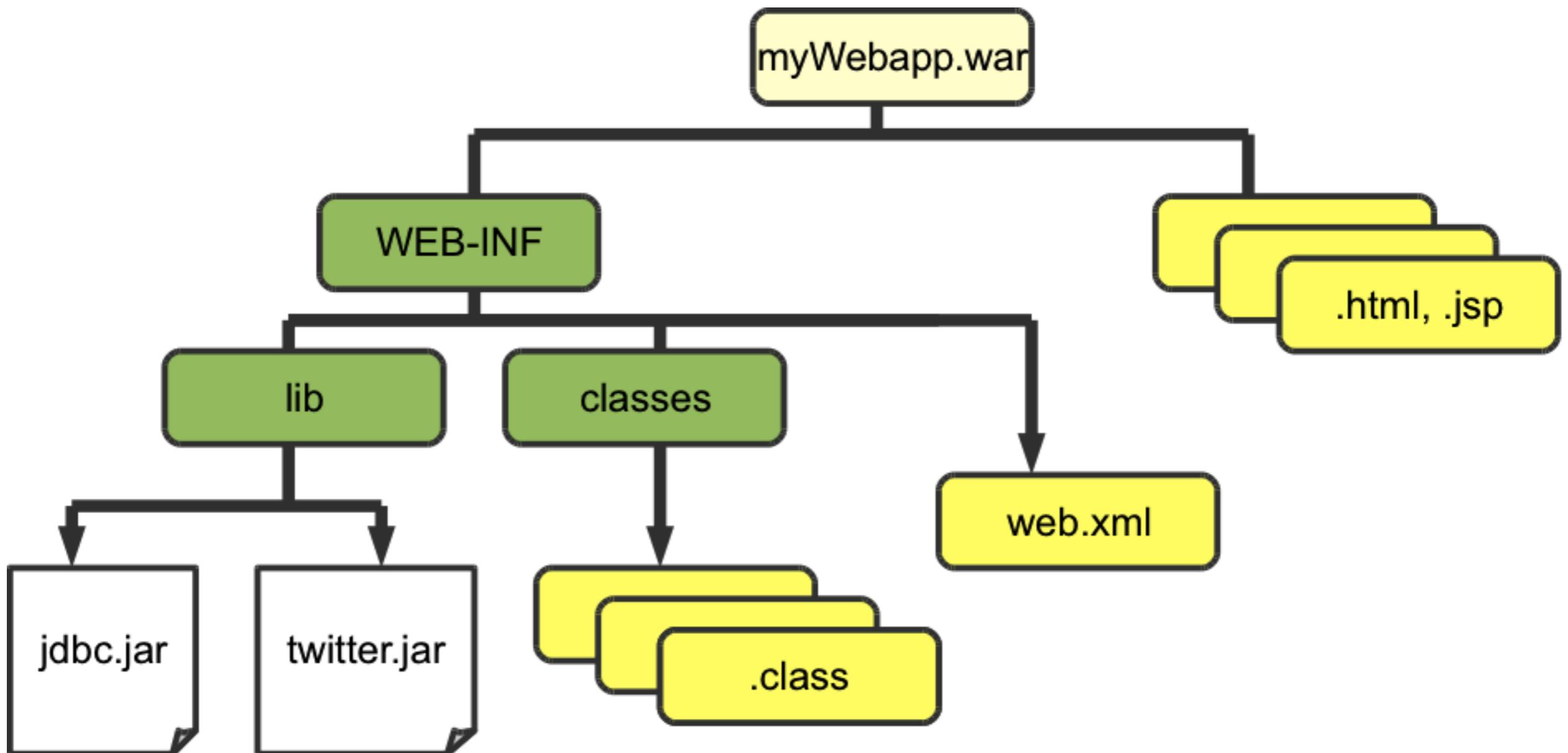
Java EE 6

- Composants métiers
- Accès aux données
- **Couche de présentation**
- Interopérabilité
- Autre

Conteneur Web

- Egalement appelé moteur de servlets ou conteneur de servlets
- Assure la gestion du cycle de vie des servlets
- Capable
 - De récupérer les requêtes HTTP
 - De déléguer le traitement des requêtes aux servlets
 - De renvoyer la réponse générée
- C'est le “moteur“ de toute application web simple (ie. sans EJB)

Structure



Servlet 3.0

- Composant logiciel écrit en Java fonctionnant du côté serveur
- Au même titre nous trouvons
 - CGI (Common Gateway Interface)
 - Langages de script côté serveur PHP, ASP, etc ...
- Permet de gérer des requêtes HTTP et de fournir au client une réponse HTTP
- Une Servlet s'exécute dans un moteur de Servlet ou conteneur de Servlet permettant d'établir le lien entre la Servlet et le serveur Web

Servlet 3.0

- Créer des pages HTML dynamiques, générer des images, générer des documents XML, etc ...
- Effectuer des tâches de type CGI (traitements applicatifs côté serveur web)
 - Manipulation de BD
 - ...
- Respecter les principes d'une architecture
 - Applet
 - Terminal mobile
 - Navigateur

Servlet 3.0

```
12 @WebServlet(name="TestServlet", urlPatterns={"/test"})
13 public class TestServlet extends HttpServlet {
14
15     @Override
16     protected void doGet(HttpServletRequest request, HttpServletResponse response)
17             throws ServletException, IOException {
18         response.setContentType("text/html");
19         PrintWriter out = response.getWriter();
20         out.println("<html>");
21         out.println("<body>");
22         out.println("<title>Page generee par une servlet</title>");
23         out.println("</head>");
24         out.println("<body>");
25         out.println("<h1>Bonjour</h1>");
26         out.println("</body>");
27         out.println("</html>");
28         out.close();
29     }
30 }
```

Cycle de vie

- `init()`
 - Appelée une fois après l'instanciation de la Servlet par le conteneur web
- `service(...)`
 - Appelée à chaque requête
- `destroy()`
 - Appelée à la destruction de la Servlet par le conteneur web

Filtres web

- Permettent d'intercepter les requêtes sur les Servlets (voire toute ressource web)
- Permettent d'exécuter du code avant et après l'exécution de la Servlet
- Permettent de ne pas surcharger le code avec des aspects techniques (AOP)
- Peuvent être définis par rapport à des pattern d'url
- Peuvent être enchaînés les uns après les autres

Filtres web

```
20 @WebFilter(filterName="TestFilter", urlPatterns={"/"})
21 public class TestFilter implements Filter {
22
23     @Override
24     public void doFilter(ServletRequest request, ServletResponse response,
25                          FilterChain chain)
26             throws IOException, ServletException {
27         System.out.println("Before request ...");
28         try {
29             chain.doFilter(request, response);
30         }
31         catch(Throwable t) {}
32         System.out.println("After request ....");
33     }
34 }
```

Collaboration

- Collaboration entre Servlets obtenue par l'interface `ServletContext`
- Conteneur d'informations unique
- Pour récupérer ce contexte → `getServletContext()`
- Méthodes utiles
 - `setAttribute(String name, Object o)`
 - `getAttribute(String name)`
 - `removeAttribute(String name)`

Suivi des utilisateurs

- HTTP est un protocole sans état
 - Pas d'historique des requêtes
- Une seule instance de chaque servlet existe
 - Les données contenues dans la classe sont partagées pour tous les utilisateurs
- Solution pour identifier un utilisateur
 - Génération d'un identifiant de session pour chaque utilisateur
 - Transmission de l'identifiant par cookie ou par réécriture d'URL

Suivi des utilisateurs

- L'interface HttpSession gère le suivi de l'utilisateur et la mémorisation de ses données personnelles
- Pour créer ou récupérer la session de l'utilisateur → `request.getSession()`
- Méthodes utiles
 - `setAttribute(String name, Object o)`
 - `getAttribute(String name)`
 - `removeAttribute(String name)`
 - `invalidate()`

Partage du contrôle

- Les Servlets peuvent partager ou distribuer le contrôle de la requête
- Deux types de distribution (RequestDispatcher)
 - Distribuer un renvoi : une servlet renvoie une requête entière (méthode forward())
 - Distribuer une inclusion : une servlet inclus du contenu généré (méthode include())
- Délégation des compétences
- Meilleure abstraction et plus grande souplesse
- Architecture MVC (Servlet = contrôleur et JSP = présentation)

Servlet 3.0

- Nouveautés
 - Web fragments
 - fichier web.xml virtuel
 - Ajout dynamique de servlet
 - de manière programmatique
 - Support du protocole comet
 - appels http asynchrones

JSP

- Java Server Page
- Génération dynamique de code HTML
- Java dans le fichier HTML
- Tags JSTL
- Etape de précompilation
 - Transformation des .jsp en Servlet
- Orienté présentation
- **A éviter !!!**

JSF 2

- JavaServer Faces
- Framework pour le développement d'applications web
- Notions de composants d'IHM à la SWING ou SWT
- Modèle d'architecture MVC
 - Contrôleurs → Managed beans
 - Vues → xhtml + JSF tags
 - Modèles → Entity ou autre
 - Ajout de règles de validation et de navigation

JSF 2

- JSF permet
 - Une séparation de la couche présentation des autres couches (MVC)
 - Un mapping entre l'HTML et l'objet
 - De se servir des composants riches et réutilisables
 - Une liaison simple entre les actions de l'utilisateur (event listener) et le code Java coté serveur
 - La création de nouveaux composants graphiques
 - De générer autre chose que du HTML (XUL, XML, WML, etc ...)

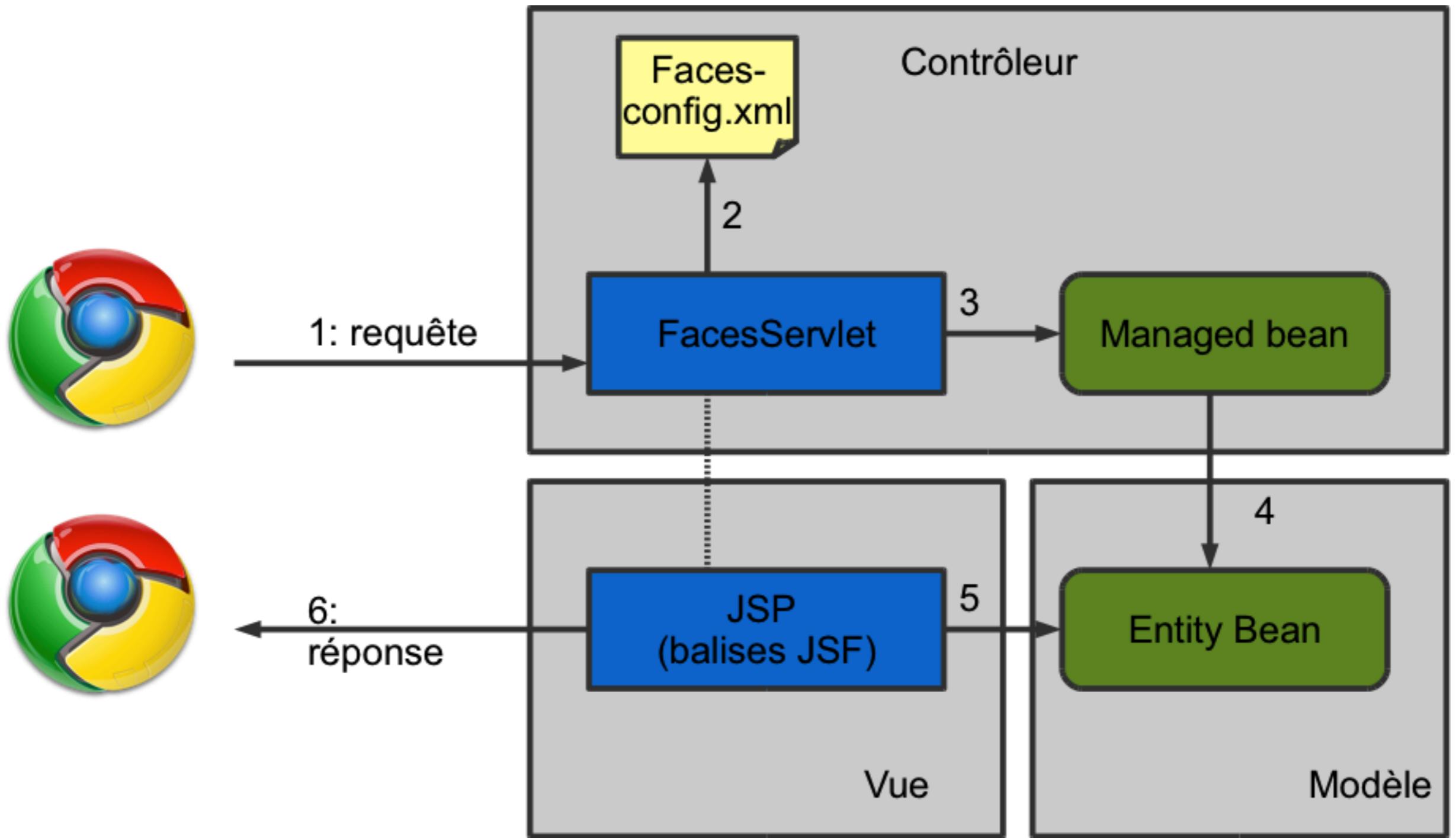
Contrôleur

- Managed bean avec un nom
- Propriétés accessibles via getters et setters
 - la vue les utilise pour afficher / mettre à jour les données
- Méthodes retournant une clé de navigation
 - nom du fichier
 - phase de la machine à états

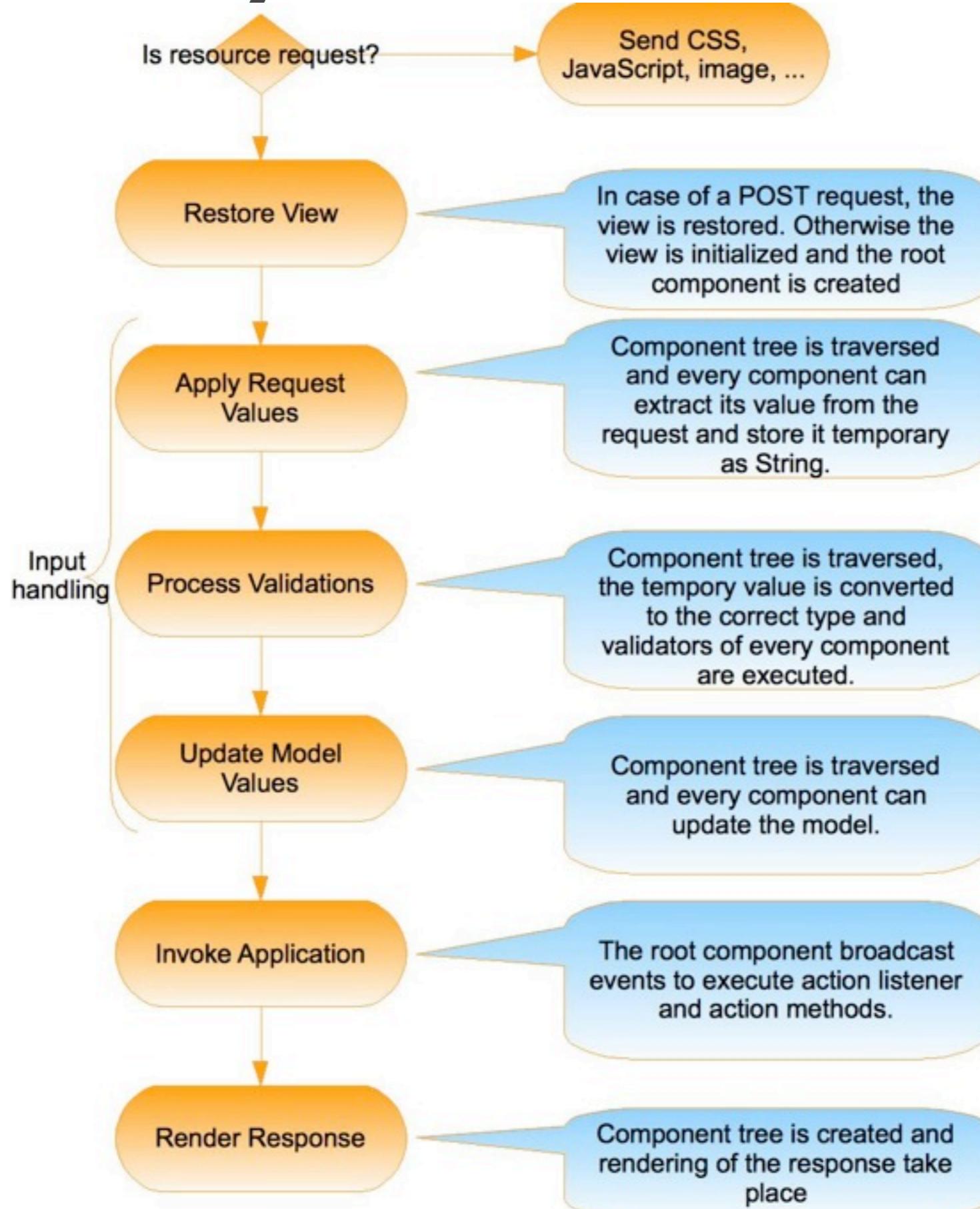
Vue

- Fichier xhtml contenant des tags JSF
 - <h:outputText>, <h:link>, <h:dataTable>
- Lecture depuis contrôleur via un EL
 - <h:outputText value="#{monControleur.text}" />
 - Classe MonControleur, getText()
- Ecriture vers le contrôleur
 - <h:inputText value="#{monControleur.text}" />
 - Classe MonControleur, setText()
- Action
 - <h:commandLink action="#{monControleur.doComputeText}" />

JSF 2



Cycle de vie



Requête



search.xhtml

```
<h:inputText value="#{catalog.keyword}" />
<h:commandButton
    action="#{catalog.search}" />
```

Requête



search.xhtml

```
<h:inputText value="#{catalog.keyword}"/>
<h:commandButton
    action="#{catalog.search}"/>
```

```
@ManagedBean
public class Catalog {
    String keyword;
    Collection<Item> items;
    @Inject Service itemService;

    public String search() {
        items = itemsService.search(keyword);
        return "results.xhtml";
    }
    // getters, setters
}
```

Requête



search.xhtml

```
<h:inputText value="#{catalog.keyword}" />
<h:commandButton
    action="#{catalog.search}" />
```

```
@ManagedBean
public class Catalog {
    String keyword;
    Collection<Item> items;
    @Inject Service itemService;

    public String search() {
        items = itemService.search(keyword);
        return "results.xhtml";
    }
    // getters, setters
}
```

Requête



search.xhtml

```
@ManagedBean  
public class Catalog {  
    String keyword;  
    Collection<Item> items;  
    @Inject Service itemService;  
  
    public String search() {  
        items = itemService.search(keyword);  
        return "results.xhtml";  
    }  
    // getters, setters  
}
```

```
<h:inputText value="#{catalog.keyword}" />  
<h:commandButton  
    action="#{catalog.search}" />
```

Requête



search.xhtml

```
@ManagedBean  
public class Catalog {
```

```
    String keyword;  
    Collection<Item> items;  
    @Inject Service itemService;
```

```
    public String search() {  
        items = itemService.search(keyword);  
        return "results.xhtml";  
    }  
    // getters, setters  
}
```

```
<h:inputText value="#{catalog.keyword}"/>  
<h:commandButton  
    action="#{catalog.search}"/>
```



results.xhtml

```
<h:dataTable value="#{catalog.items}"/>
```

Requête



search.xhtml

```
@ManagedBean  
public class Catalog {
```

```
    String keyword;  
    Collection<Item> items;  
    @Inject Service itemService;
```

```
    public String search() {  
        items = itemService.search(keyword);  
        return "results.xhtml";  
    }
```

```
    // getters, setters  
}
```

```
<h:inputText value="#{catalog.keyword}" />  
<h:commandButton  
    action="#{catalog.search}" />
```



results.xhtml

```
<h:dataTable value="#{catalog.items}" />
```

Requête



search.xhtml

```
@ManagedBean  
public class Catalog {
```

```
    String keyword;  
    Collection<Item> items;  
    @Inject Service itemService;
```

```
    public String search() {  
        items = itemService.search(keyword);  
        return "results.xhtml";  
    }  
    // getters, setters  
}
```

```
<h:inputText value="#{catalog.keyword}"/>  
<h:commandButton  
    action="#{catalog.search}"/>
```



results.xhtml

```
<h:dataTable value="#{catalog.items}"/>
```

Fonctionnalités

- Templating
- Création de composants personnalisés
- Intégration poussée avec CDI (scopes d'injection)
- Navigation (machine à état)
- Scope de notification
- Validation client
- AJAX

Java EE 6

- Composants métiers
- Accès aux données
- Couche de présentation
- **Interopérabilité**
- Autre

Web Services

- JAX-RPC : RPC basé sur XML
- **JAX-WS : Web Services basés sur XML**
- **JAXB : API de binding XML**
- **JAX-RS : Services REST**
- SAAJ :API SOAP avec pièces jointes
- StAX :API de streaming XML

JAX-B

- Java Architecture for Xml Binding → JSR-222
- Permet de faire correspondre un ensemble de classes à un document XML
- Et vice et versa
- Via des opération de sérialisation/désérialisation (marshalling/unmarshalling)
- Permet de manipuler du XML sans avoir à connaître de XML ou la manière dont le document XML est traité

JAX-B

```
6  @XmlRootElement
7  public class Client {
8      private String nom;
9      private String prenom;
10     private String telephone;
11     private String adresse;
12
13     public String getAdresse() {...}
14     public void setAdresse(String adresse) {...}
15     public String getNom() {...}
16     public void setNom(String nom) {...}
17     public String getPrenom() {...}
18     public void setPrenom(String prenom) {...}
19     public String getTelephone() {...}
20     public void setTelephone(String telephone) {...}
21
22 }
23 }
```

JAX-B

```
2 <?xml version="1.0" encoding="UTF-8"?>
3 <client>
4   <nom>Dusse</nom>
5   <prenom>Jean-Claude</prenom>
6   <telephone>0606060606</telephone>
7   <adresse>1 avenue ..... </adresse>
8 </client>
```

JAX-WS

- Java API for XML WebServices → JSR-224
Conçut pour remplacer JAX-RPC
- Utilisation d'annotations sur des POJO pour le paramétrage (@WebService)
- Utilisation de SOAP 1.2
- Utilisation de JAXB pour le binding XML (requêtes/réponses)
- Peut-être utilisé nativement dans Java SE 6
Utilisable directement sur les EJB session

JAX-WS

```
6  @Stateless  
7  @WebService  
8  public class CalculatorWS {  
9  
10     public int add(int a, int b) {  
11         return a + b;  
12     }  
13 }
```

- Toutes les méthodes publiques deviennent des opération du web service
- Le schéma WSDL est généré au déploiement

JAX-WS

```
8  @Stateless
9  @WebService (
10     name="Calculator",
11     portName="CalculatorPort",
12     serviceName="CalculatorService",
13     targetNamespace="http://calculator.univ-lr.fr")
14 public class CalculatorWSConf {
15     @WebMethod(operationName="addCalc")
16     public int add(@WebParam(name="param1") int a, int b) {
17         return a + b;
18     }
19 }
```

JAX-RS

- Services RESTful
- Basés sur des POJO et des annotations
- Tout est considéré comme des ressources
- CRUD basé sur HTTP

HTTP	ACTION	HTTP	ACTION
GET	lecture d'un ressource	PUT	update d'une ressource
POST	création d'une ressource	DELETE	efface la ressource

JAX-RS

```
7  @Path("/helloworld")
8  public class HelloWorldResource {
9      @GET
10     @Produces("text/plain")
11     public String sayHello() {
12         return "Hello World";
13     }
14 }
```

JAX-RS

Request

GET /helloworld HTTP/1.1

Host: example.com

Accept: text/plain

Response

HTTP/1.1 200 OK

Date:Wed, 12 Nov 2008 16:41:58 GMT

Server:Apache/1.3.6

Content-Type: text/plain; charset=UTF-8

Hello World

JAX-RS

Request

GET /helloworld HTTP/1.1

Host: example.com

Accept: text/plain

Response

HTTP/1.1 200 OK

Date:Wed, 12 Nov 2008 16:41:58 GMT

Server:Apache/1.3.6

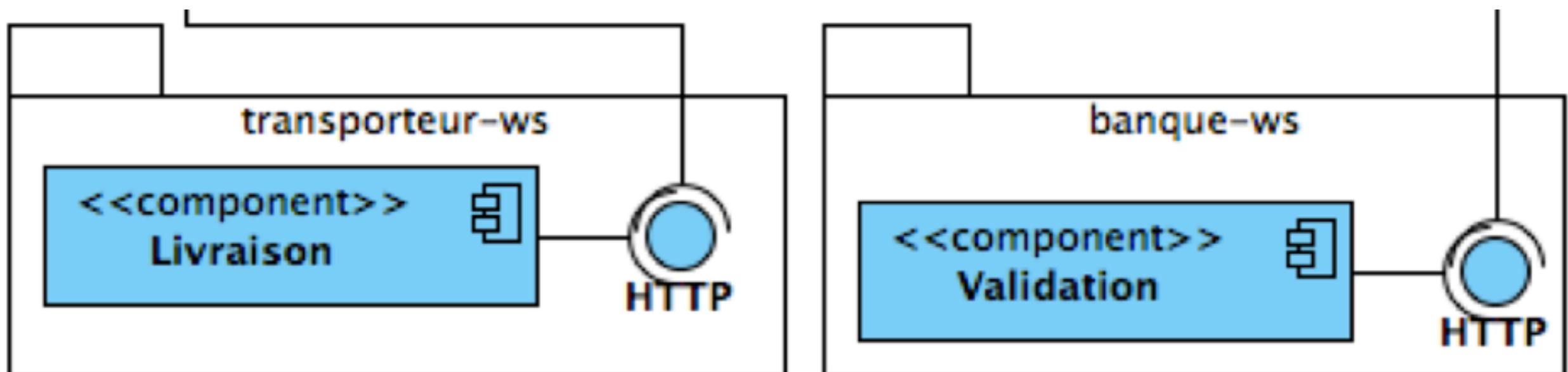
Content-Type: application/xml; charset=UTF-8

<message>hello</message>

Cas d'étude

- Utilisation de services tiers
 - Validation de la carte de crédit lors du paiement auprès d'une banque
 - Programmation d'une livraison auprès d'une compagnie de transport

Cas d'étude



Java EE 6

- Composants métiers
- Accès aux données
- Couche de présentation
- Interopérabilité
- **Autre**

JavaMail

- API de gestion des emails → JSR-919
- Ensemble de classes abstraites représentant un système de courrier électronique
- Framework indépendant de la plateforme
- Framework indépendant du protocole
- Supporte les protocoles de mails populaires
 - POP
 - IMAP
 - SMTP

JavaMail

```
14     private Transport transport;
15
16     public void sendMail() {
17         try {
18             Properties properties = new Properties();
19             properties.put("mail.smtp.host", "smtp.univ-lr.fr");
20             properties.put("mail.smtp.auth", "true");
21             Session session = Session.getInstance(properties, null);
22             javax.mail.Message msg = new MimeMessage(session);
23             msg.setFrom(new InternetAddress("no-reply@univ-lr.fr"));
24             msg.setRecipients(javax.mail.Message.RecipientType.TO,
25                               InternetAddress.parse("test@univ-lr.fr", false));
26             msg.setSubject("Sujet");
27             msg.setText("Un super email");
28             msg.setSentDate(new Date());
29             transport = session.getTransport("smtp");
30             transport.connect("smtp.univ-lr.fr", "mancelin", "password");
31             transport.sendMessage(msg, msg.getAllRecipients());
32         } catch (MessagingException me) { System.out.println(me); }
33         finally {
34             try {
35                 transport.close();
36             } catch (MessagingException e) { System.out.println(e); }
37         }
38     }
```

Cas d'étude

- Utilisation de JavaMail dans le cadre des relations avec le client et les internautes
- Utilisé en combinaison avec un MDB dédié à l'envoi d'email
- Développement d'un DSL (Fluent API)
- Confirmations d'inscriptions, de commandes, newsletters, etc ...

JCA

- J2EE Connector Architecture → JSR-112
- Applications Java EE
 - EJB
 - Modules web
 - Clients
- SI d'entreprise
 - ERP
 - MainFrames
 - SGBD
 - Application C, C++, COBOL, etc ...

JCA

- JCA permet de résoudre les problèmes d'intégration dans le SI
- JCA propose
- Un Resource Adapter
- Des contrats applicatifs entre les Resource Adapter et les modules Java EE
- Des contrats systèmes entre les serveurs d'applications et le Resource Adatper

JCA

- Un Resource Adapter est un driver entre le serveur d'applications et le SI
- Il est composé de :
 - Jars permettant de wrapper l'accès aux ressources du SI
 - Librairies natives permettant l'accès aux ressources du SI
 - D'un descripteur de déploiement (ra.xml)

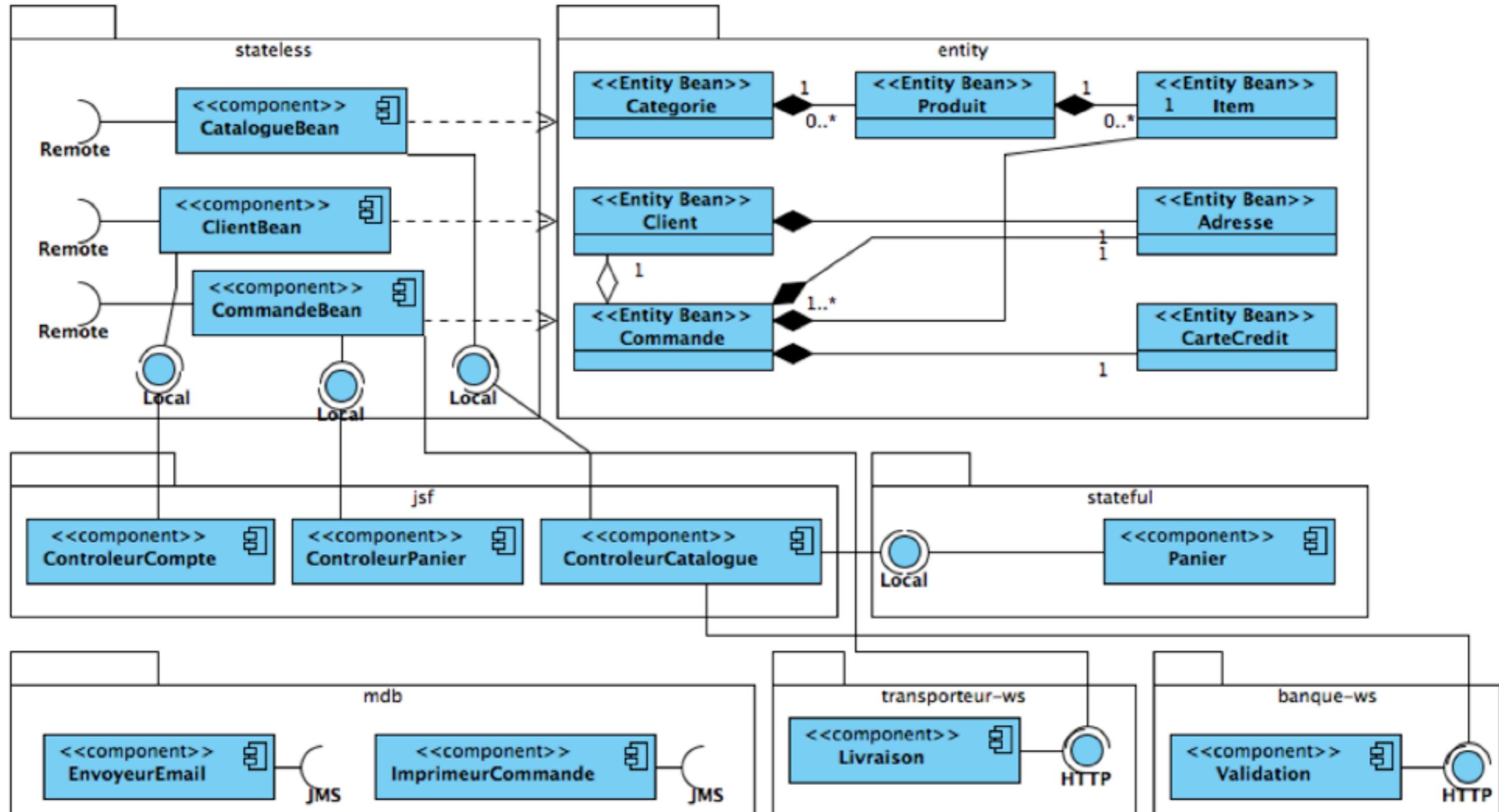
JCA

- Les contrats définissent
 - La gestion des connexions
 - La gestion des transactions
 - La gestion de la sécurité
 - La gestion des messages
 - La gestion du cycle de vie
 - La gestion des threads

Cas d'étude

- Utilisation de JCA pour intégrer des équipements existants
 - La caisse enregistreuse (gestion des stocks en direct)
 - Le logiciel de gestion de commandes propriétaire qui était utilisé avant la nouvelle architecture

Cas d'étude complet



Plan

- Concepts de base
- Cas d'étude
- Java EE 6
- **Serveurs d'applications**
- TP

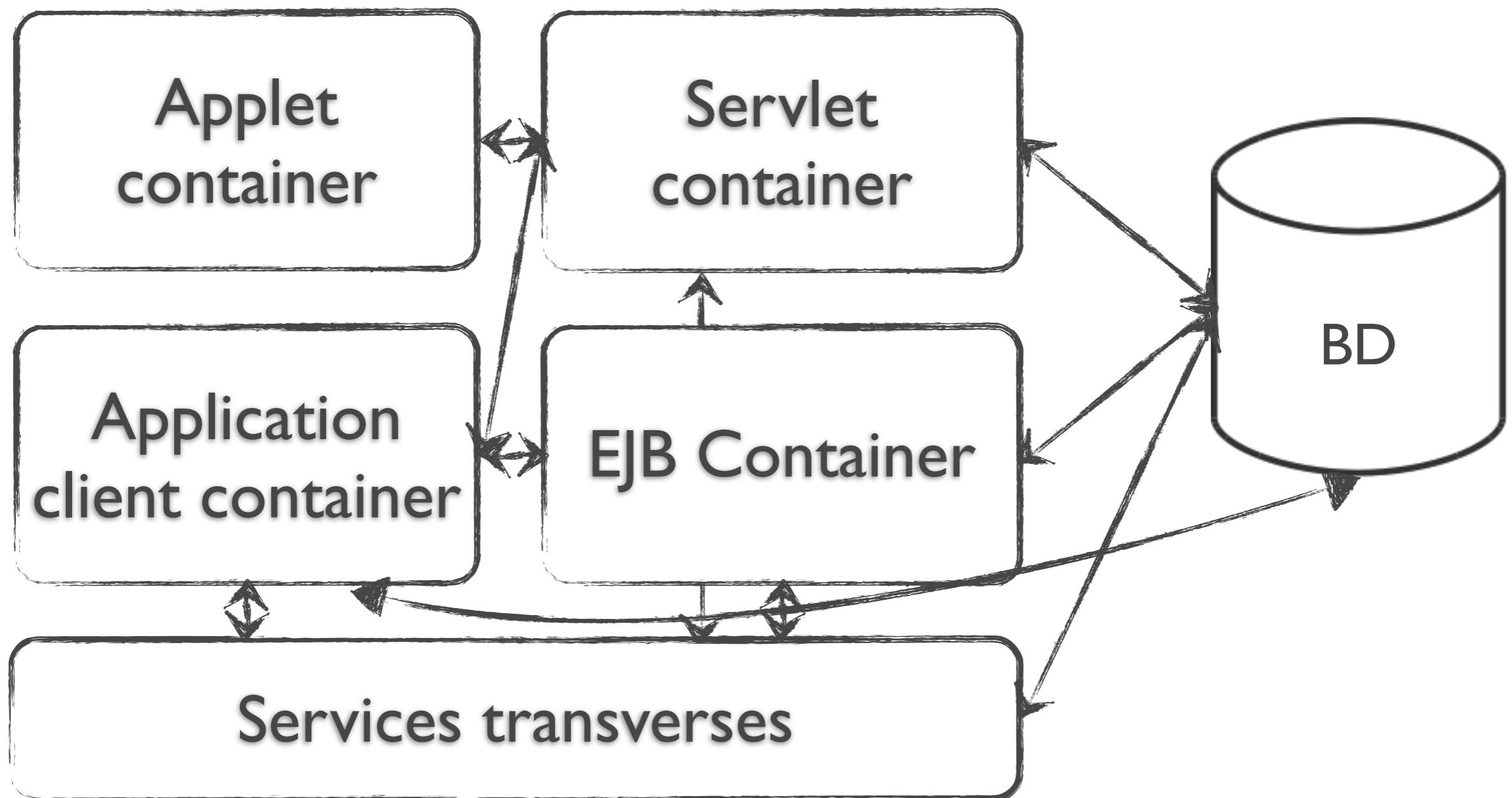
Serveur d'applications

- Expose la logique et les processus métier et les propose à des applications tiers
- Centralise les applications
- Assure la persistance des données au cours et entre différentes transactions
- Assure la persistance des données partagées
- Arbitre l'accès aux ressources (clients concurrents)

Avantages

- Intégrité du code et des données
- Configuration centralisée
- Sécurité
- Performances et scalabilité
- Support des transactions
- Coût total de propriété
- Indépendance vis à vis des clients

Serveur d'applications



Existant

- Borland ES
- ColdFusion
- Geronimo
- **Glassfish**
- JBoss
- JOnAS
- SAP NetWeaver
- Tomcat (OpenEJB)
- WebLogic
- WebObjects
- WebSphere
- Resin

...

GlassFish

- Serveur d'applications de Sun (puis Oracle) basé sur OSGi (serveur hybride)
- Implémentation de référence de Java EE 5 et Java EE 6
- Plateforme hautement dynamique et adaptable
- Pont vers le monde OSGi
- Supporte le clustering (version 3.1)
- Simple à administrer

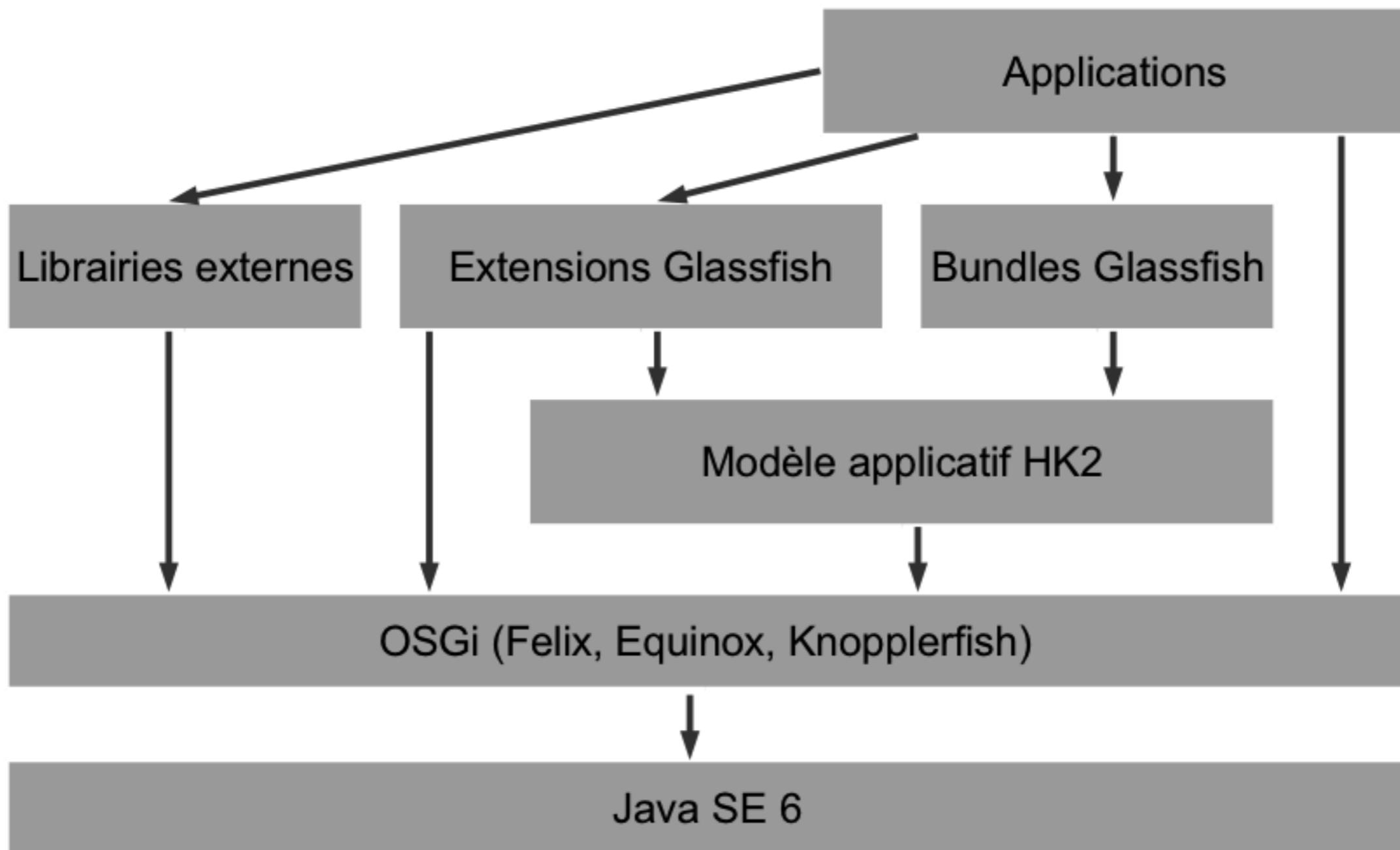
GlassFish

- Open-source (CDDL & GPL v2)
- Qualité d'un outil entreprise
- Base OSGi (modularité standard, 216 bundles)
- Démarrage / extinction dynamique des services (empreinte mémoire réduite)
- Temps de démarrage et empreinte mémoire réduits
- Utilisation intensive de HK2 (gestion des services)
- Extensions des services présents
- Bundles OSGi personnalisés (monitoring, logging, administration, commandes, etc ...)
- Consommation via @Resource

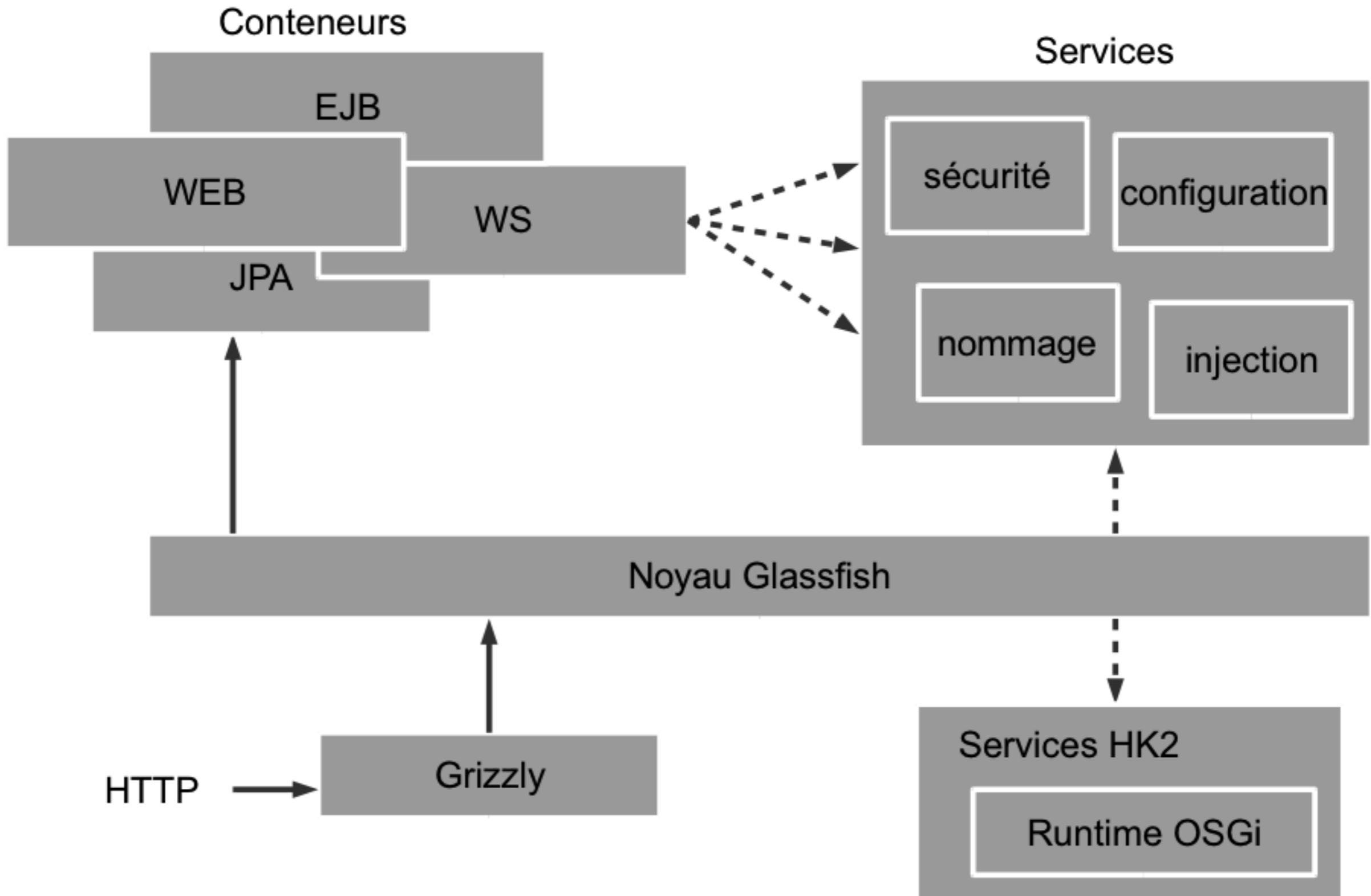
Points forts

- Profils Java EE 6 (full, web)
- Update center
 - Crédit de distributions Glassfish « à la carte »
- Choix de l'outillage (Eclipse, Netbeans, IntelliJ)
- Un seul fichier de configuration / domaine
- Système de versioning des applications
- Admin. ligne de commande et services REST
- Mode embarqué
- Support de différents langages (containers)
 - Ruby, Rails, Sinatra, merb, Grail, Groovy, PHP, Symphony, Python, Django, Scala, Lift

Architecture



Architecture



Informations

- Site GlassFish
 - <http://glassfish.java.net>
- Wiki GlassFish
 - <http://wikis.sun.com/display/GlassFish/GlassFishWiki>
- Versioning
 - <http://wikis.sun.com/display/GlassFish/VersioningDesignDocument>

JOnAS

- Serveur d'applications du consortium OW2 basé sur OSGi (serveur hybride)
- Plateforme hautement dynamique et adaptable
- Pont vers le monde OSGi
- JOnAS 5.1 certifié Java EE 5
- Supporte le clustering (avec CMI)
- Simple à administrer
- Possède un outil d'auto-management : JASMINe

JOnAS

- Développé en open-source par OW2
- OW2 : Consortium international dédié à la création de middleware open-source
- Contributeurs principaux : Bull, FranceTelecom, Peking University, INRIA, UJF, UNIFOR, SERLI
- Projets OW2 liés : EasyBeans, JASMINe, JORAM, JOTM, CMI, Shelbie
- Autres communautés open-source : Apache, CodeHaus, JBoss...

JOnAS



Spécificités

- Profils (full ou micro)
- Interface d'administration nextgen (WebOS like)
- Plans de déploiement
- Services on-demand
- Bridge OSGi (injection de ressources OSGi, EJB/ servlet comme services OSGi)
- ClassLoaders filtrant
- Versionning du déploiement

Informations

- Site JOnAS
 - <http://jonas.ow2.org>
- Site EasyBeans
 - <http://easybeans.ow2.org>
- Site OW2
 - <http://www.ow2.org/>
- Découverte de la 5.1
 - http://wiki.jonas.ow2.org/xwiki/bin/view/Doc/JOnAS+5_1_0

Plan

- Concepts de base
- Cas d'étude
- Java EE 6
- Serveurs d'applications
- TP

TP

Littérature

- Livre d'Antonio GONCALVES
 - Beginning Java EE 6 platform with GlassFish 3

Questions ?