

Java EE 7



Mathieu ANCELIN
@TrevorReznik



Mathieu ANCELIN

- Développeur @SERLI
- Scala, Java, web & OSS
 - ReactiveCouchbase, Weld-OSGi, Weld, etc ...
 - Poitou-Charentes JUG
- Membre de l'expert group CDI 1.1 (JSR-346)
- Membre de l'expert group OSGi Enterprise
- @TrevorReznik



- Société de conseil et d'ingénierie du SI
- 75 personnes
- 80% de business Java
- Contribution à des projets OSS
- 10% de la force de travail sur l'OSS
- Membre de l'EG JSR-346
- Membre de l'OSGi Alliance
- www.serli.com @SerliFr





Termes

- JCP / JSR
- Annotation
- JavaBean / Bean
- POJO

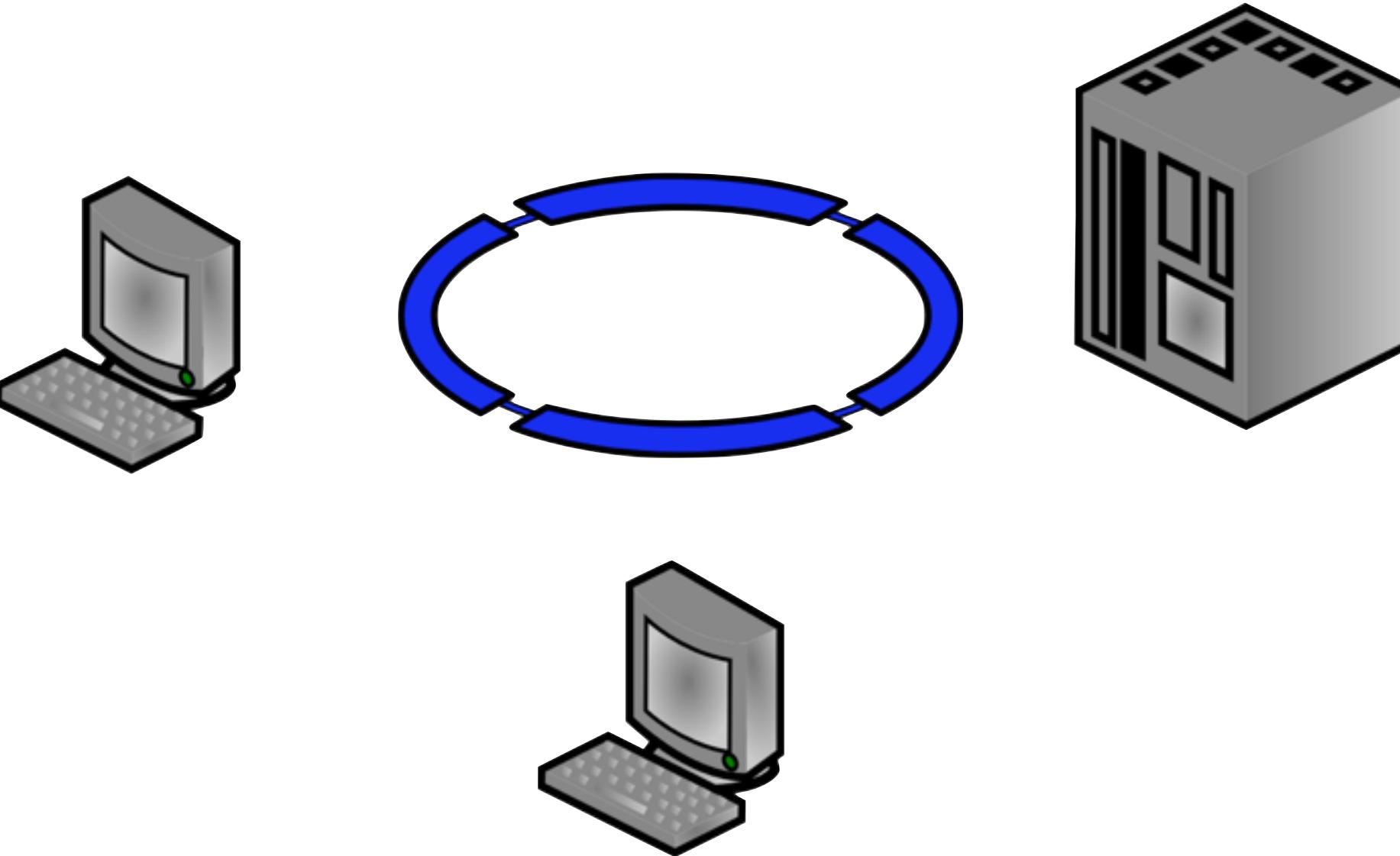


Un peu d'histoire



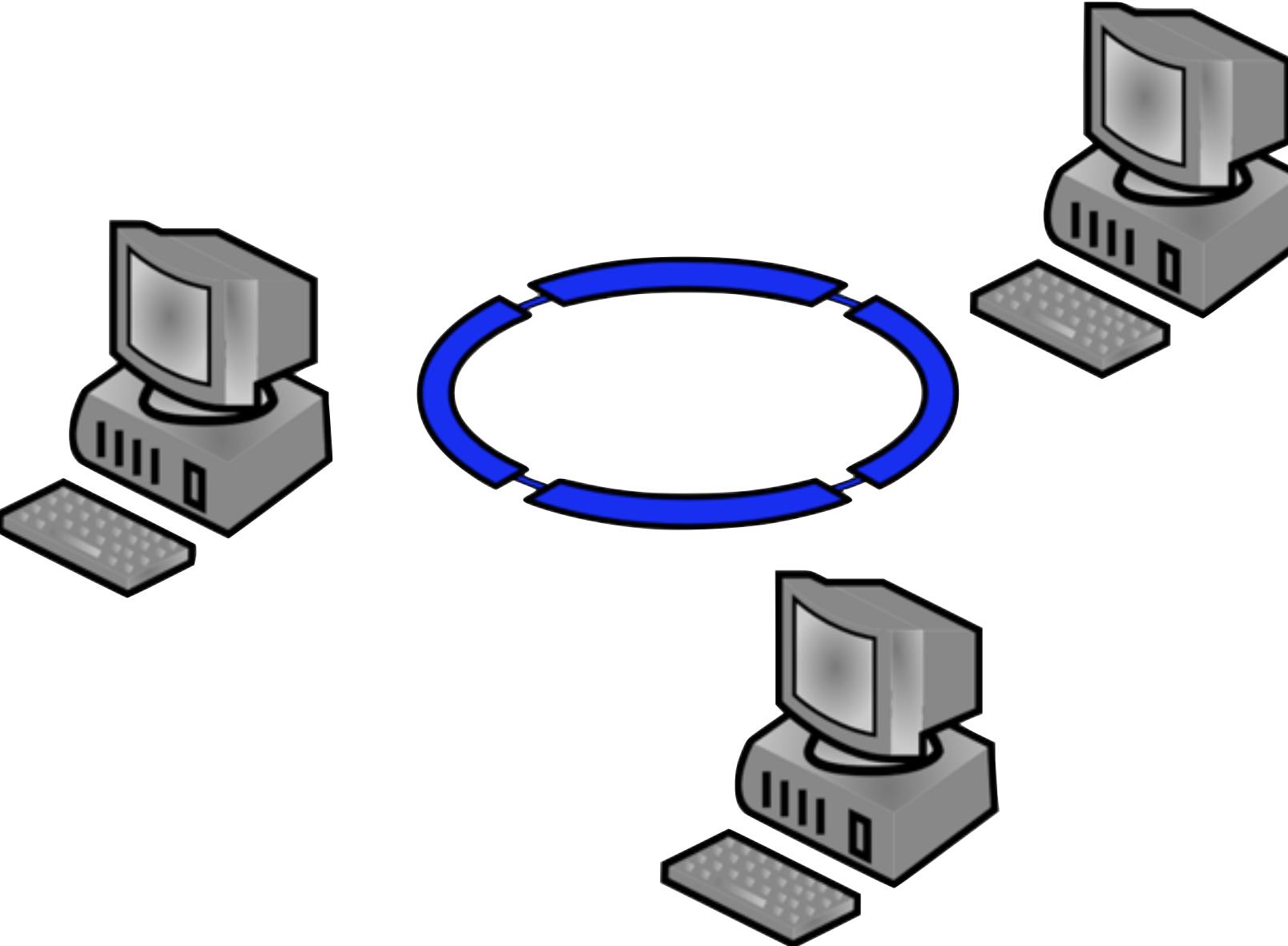


Un peu d'histoire



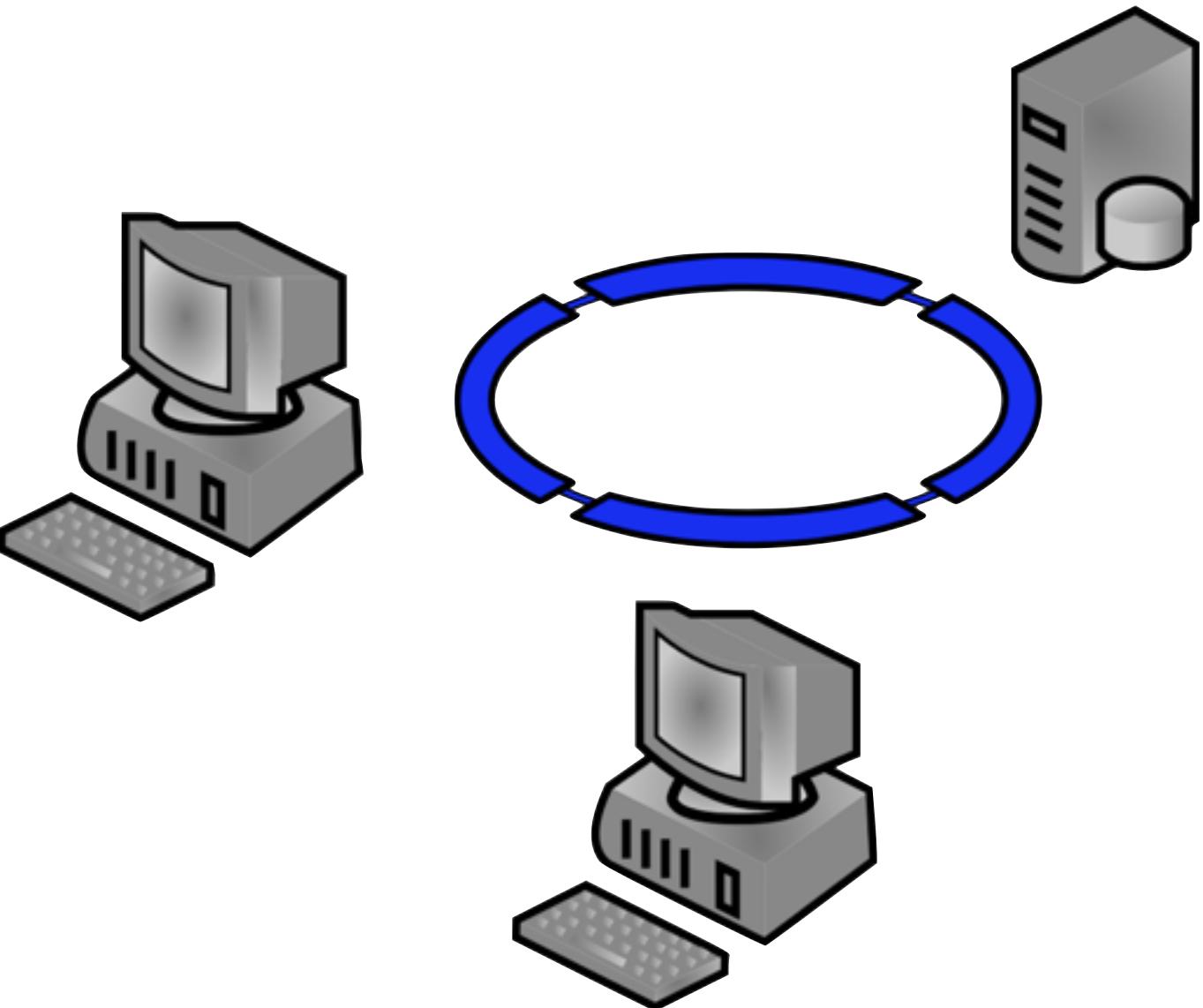


Un peu d'histoire

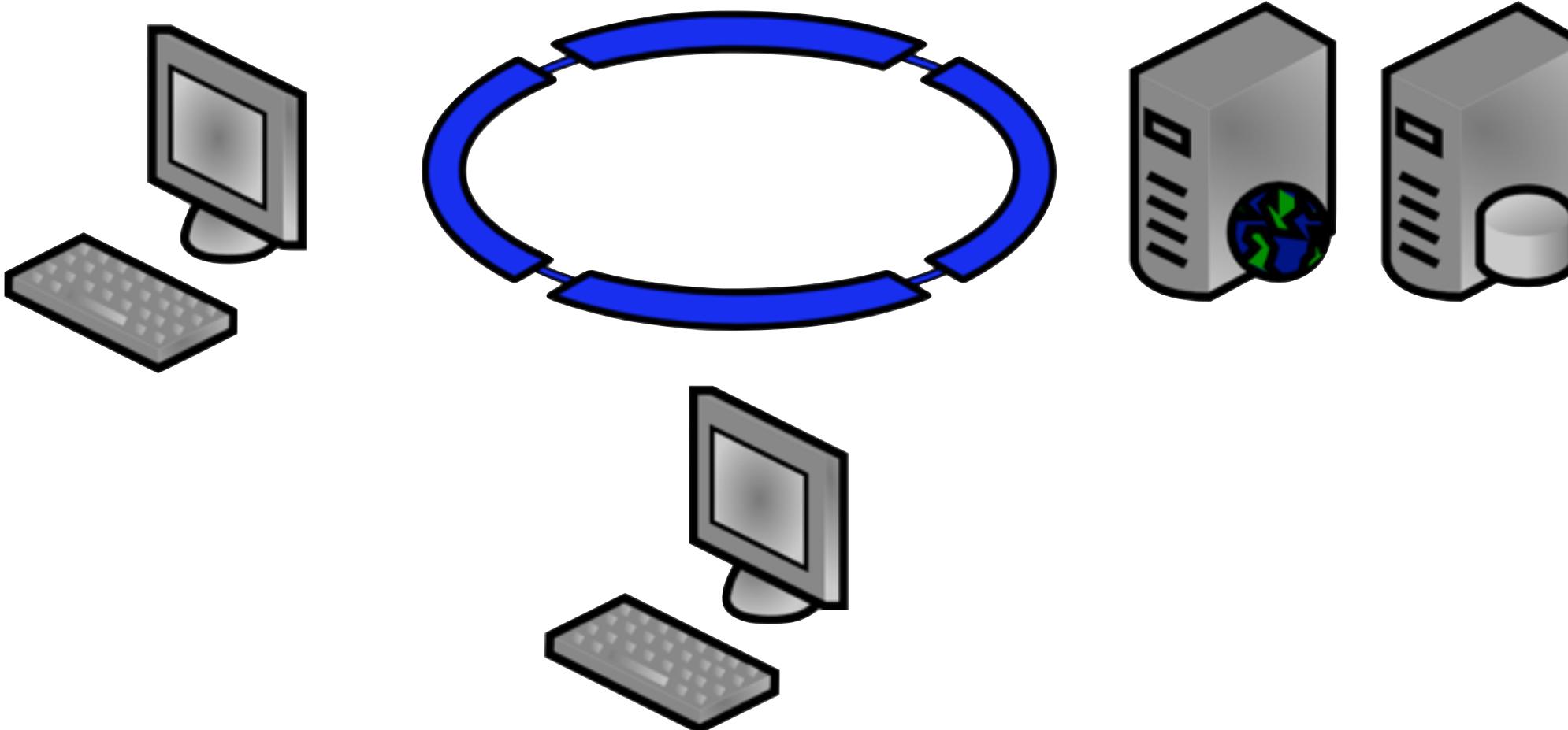




Un peu d'histoire

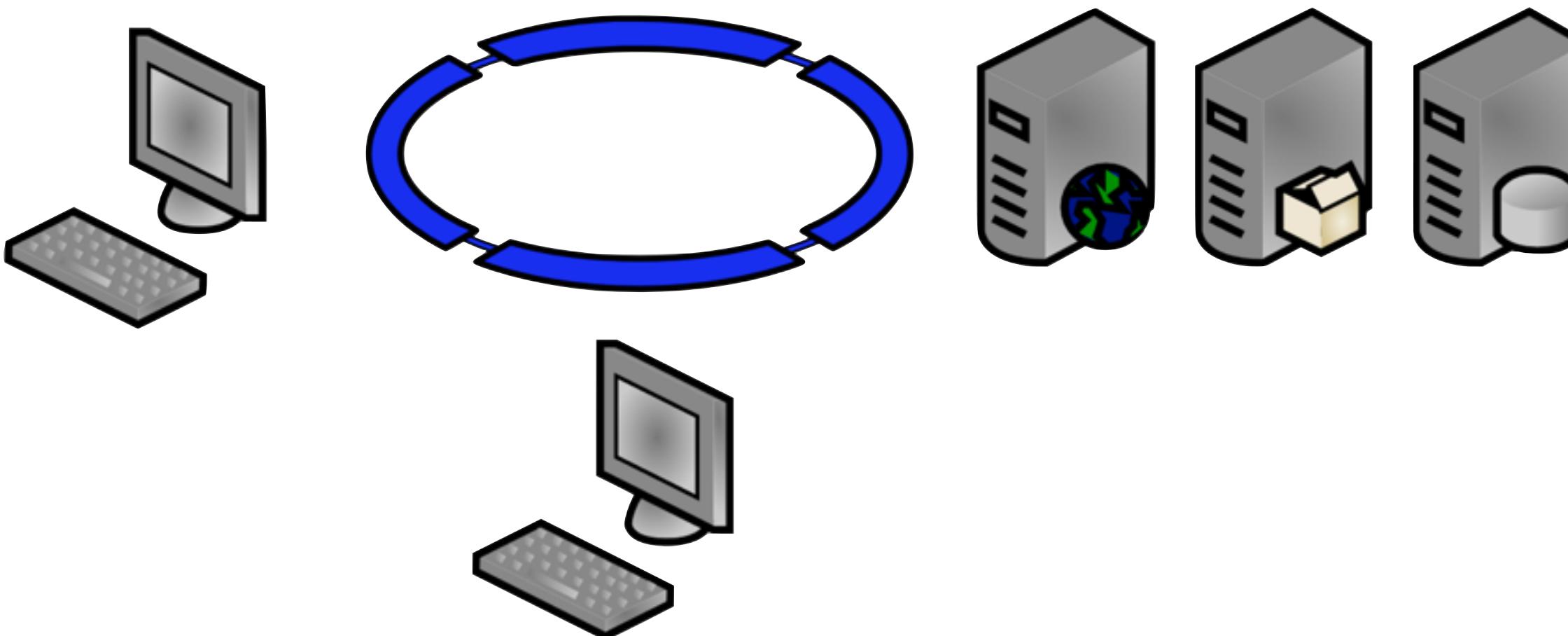


Un peu d'histoire



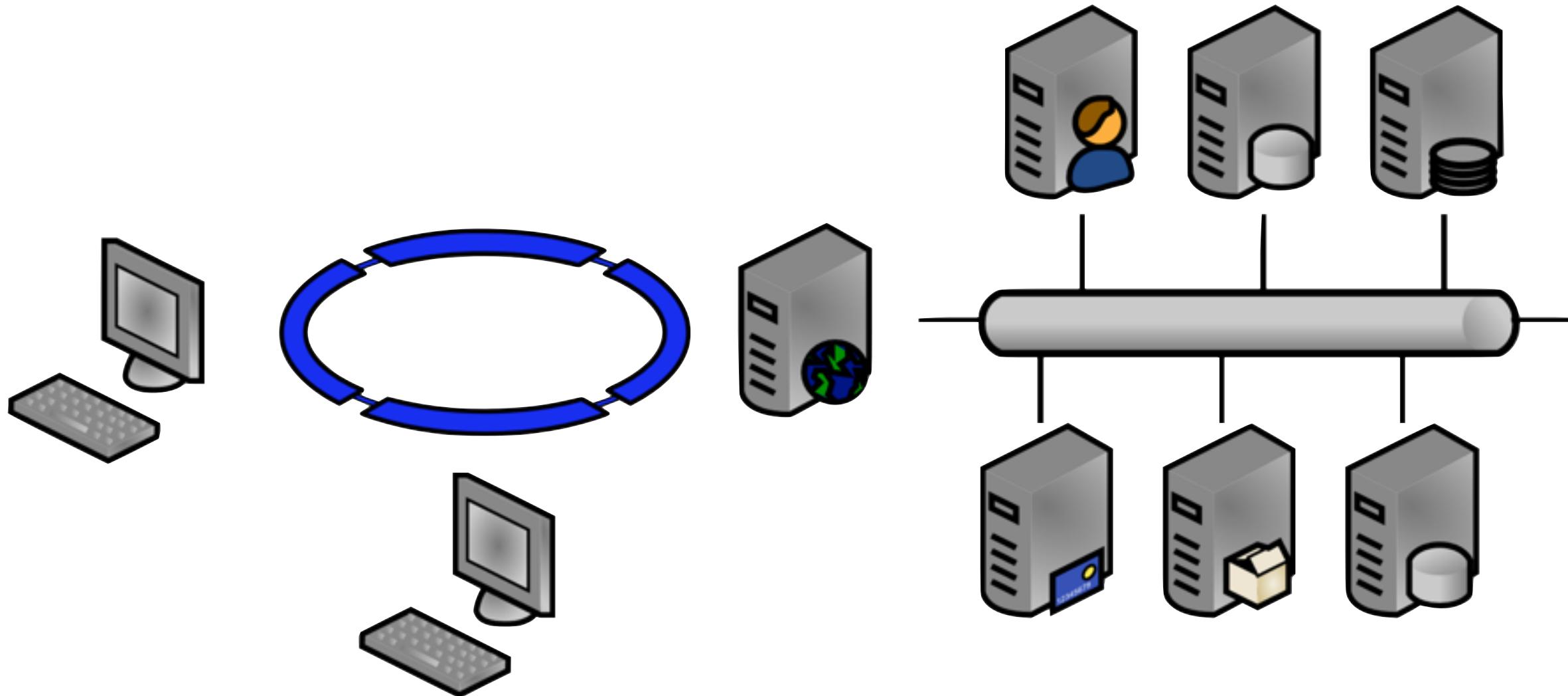


Un peu d'histoire

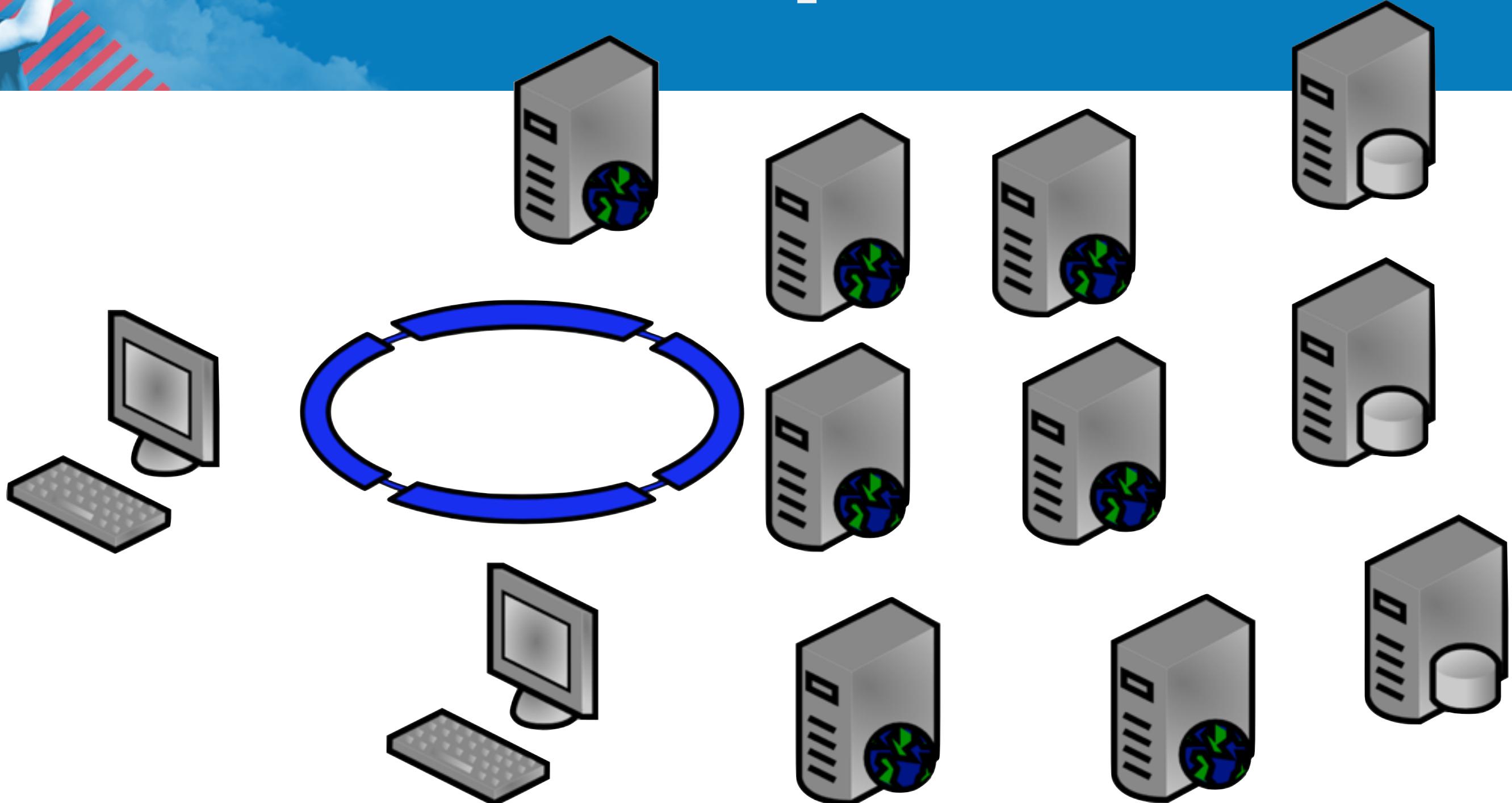




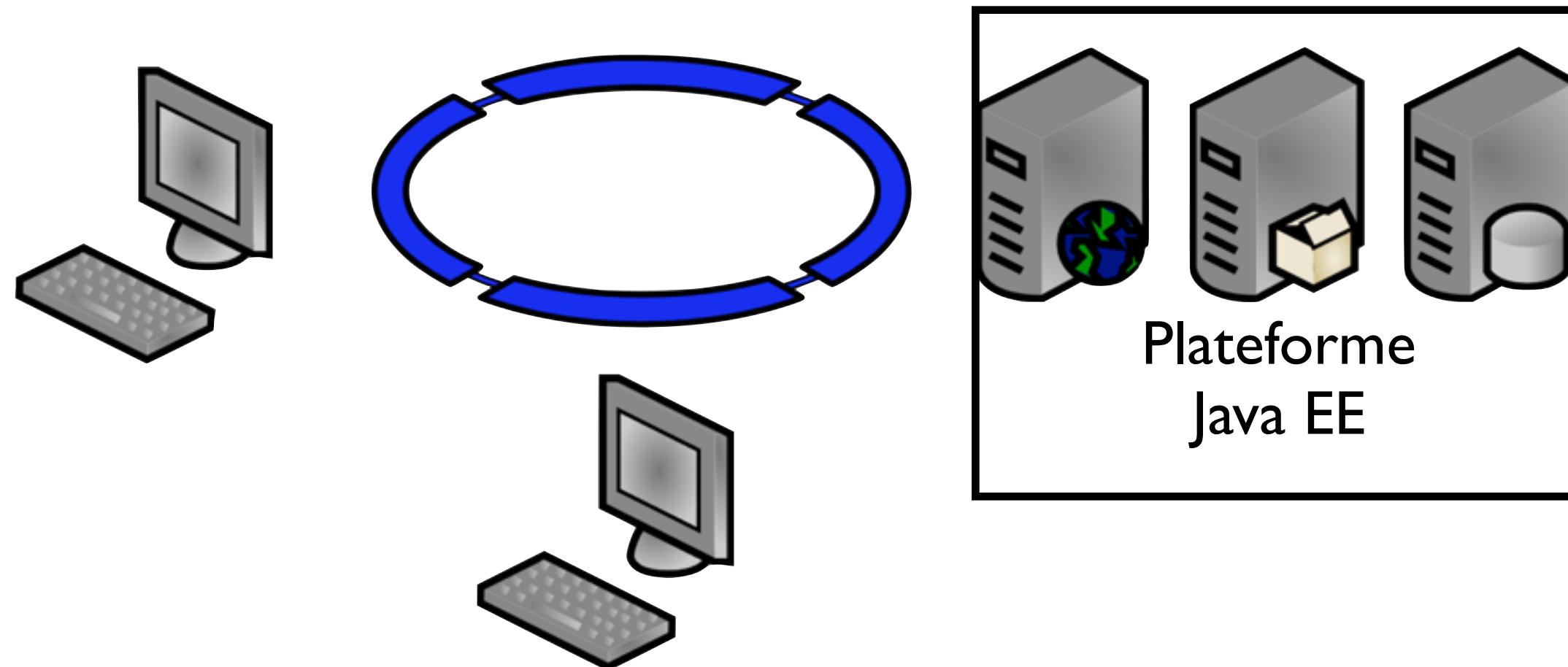
Un peu d'histoire



Un peu d'histoire

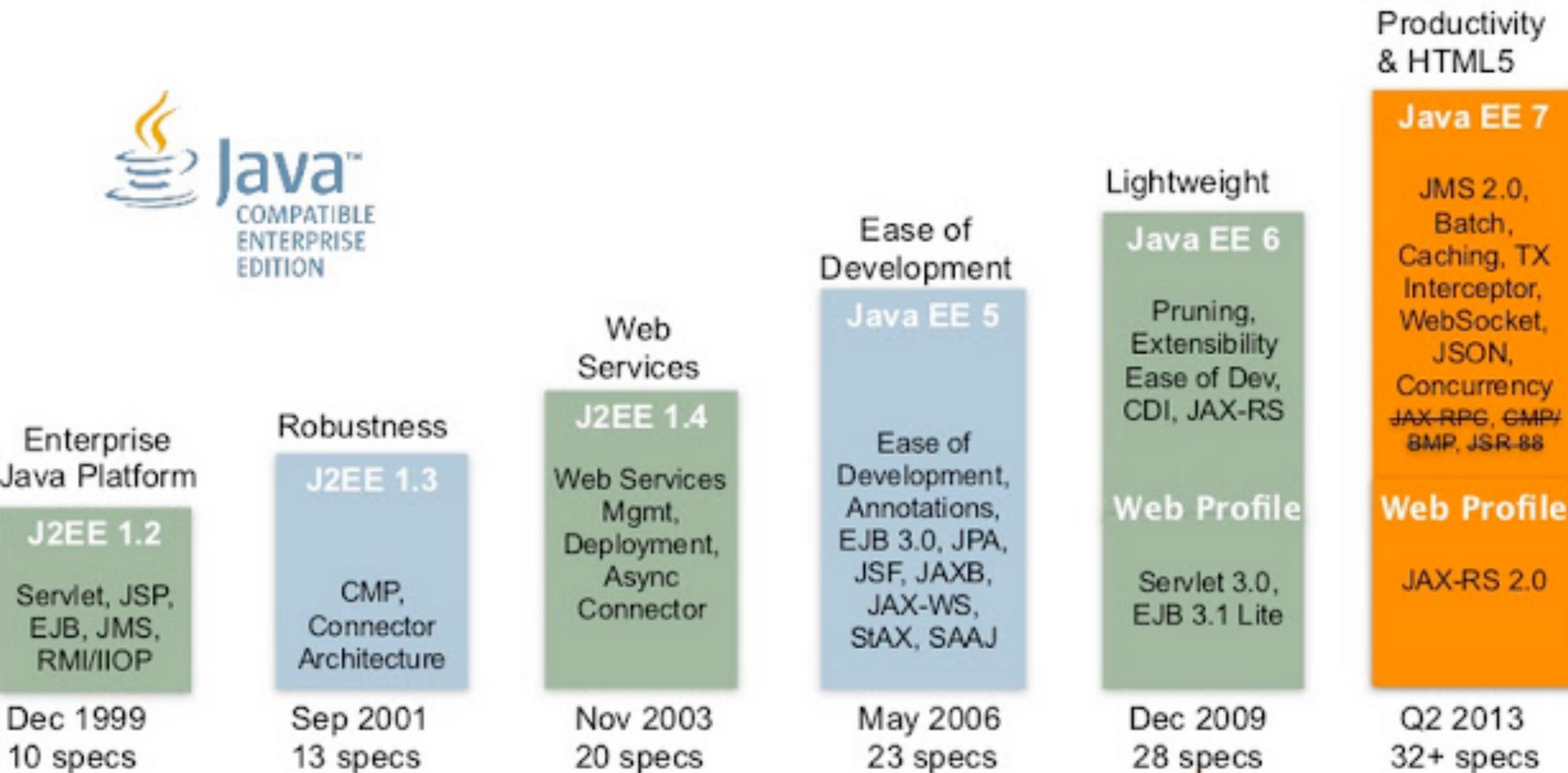


Un peu d'histoire





Un peu d'histoire





And the winner is ...

Portable
Extensions

JSP 2.2

JSF 2.2

JAX-RS
2.0

EL 3.0

Servlet 3.1

Common
Annotations
1.1

Interceptors 1.1

CDI 1.1

Managed Beans 1.0

EJB 3.2

Connector
1.6

JPA 2.1

JTA 1.2

JMS 2.0

Bean Validation 1.1

Concurrency Utilities
(JSR 236)

Batch Applications
(JSR 352)

Java API for JSON
(JSR 353)

Java API for WebSocket
(JSR 356)



Java EE 7

- Plus riche
- Plus simple
- Plus léger
- Extensible
- Modèle de programmation plus cohérent

Cloud WTF ???



SERLI



Profils

- Possibilité de créer des ensembles cohérents de technologies
- Java EE, un profil parmi d'autres
- Web Profile
- D'autres profils à venir (peut être ... ou pas)
 - Minimal
 - Portail



Les données



- 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)

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

```
@Entity  
public class Person implements Serializable {  
    @Id private Long id;  
    private String name;  
    // getters, setters  
}
```



- 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`



- 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

```
@Entity
public class Person implements Serializable {
    @Id private Long id;
    private String name;
    @ManyToOne private Address address;
    // getters, setters
}
```

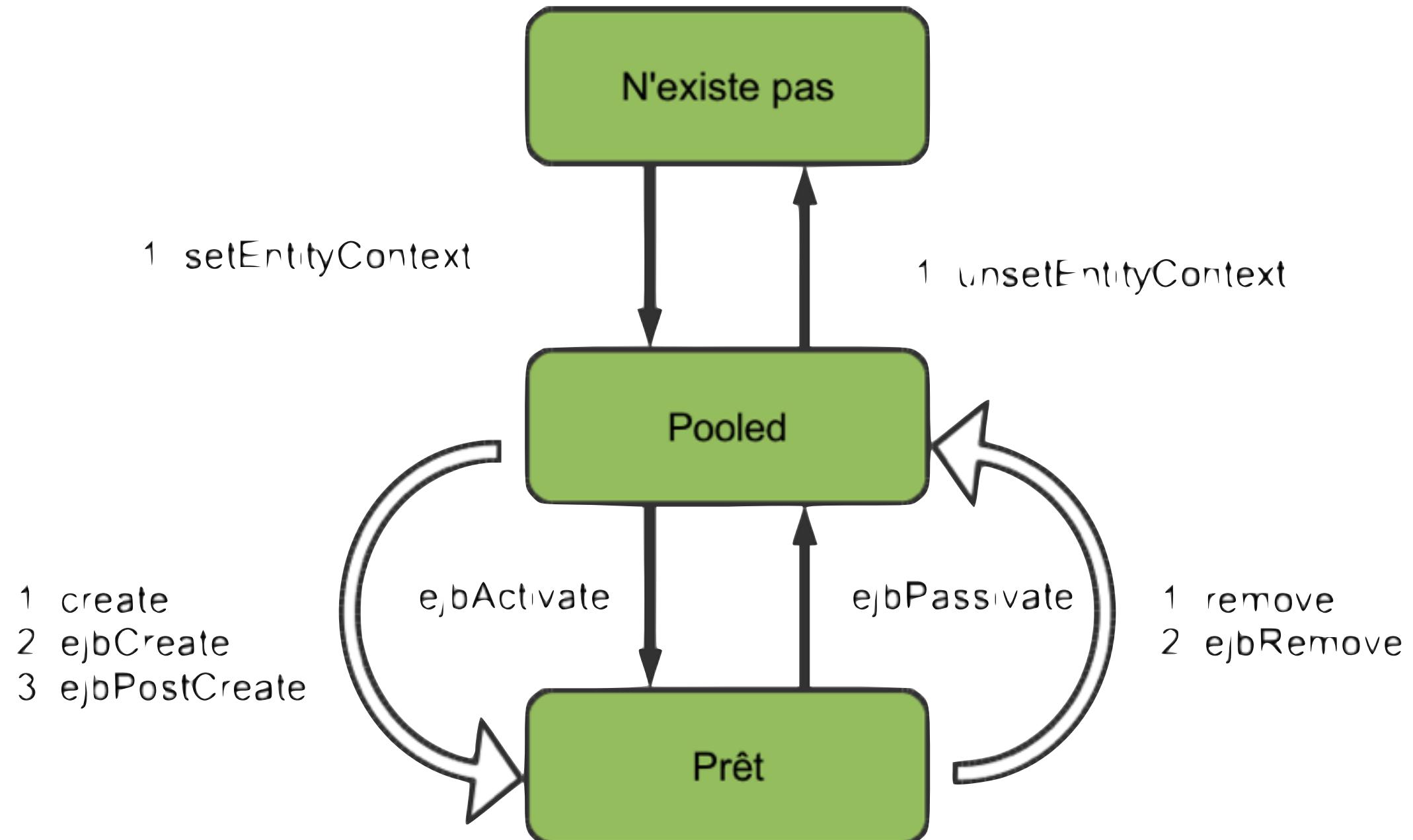
```
@Entity
public class Address implements Serializable {
    @Id @GeneratedValue(strategy = GeneratorType.AUTO) private Long id;
    private String value;
    @OneToMany private List<Person> persons;
    // getters, setters
}
```



Cycle de vie

- Il existe des callbacks de cycle de vie 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

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="TodoPU" transaction-type="JTA">
        <jta-data-source>jdbc/_default</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="javax.persistence.schema-generation.database.action"
                     value="drop-and-create"/>
        </properties>
    </persistence-unit>
</persistence>
```



Entity Manager

```
@PersistenceContext EntityManager em;

public void createPerson() {
    Address address = new Address();
    address.setValue("Technoforum 17071 La ROCHELLE");
    Person person = new Person();
    person.setName("John Doe");
    person.setAddress(address);
    em.persist(address);
    em.persist(person);
    em.flush();
}
```



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 méta-modèles
 - Chaque entité X possède un méta-modèle nommé class X_
 - Génération par Annotation Processor
 - Peut être utilisé sans méta-modèle



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 }
```



JPA

I DON'T ALWAYS DO
JPA
BUT WHEN I DO, IT'S LIKE
MAGIC



Bean Validation

- 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



Transactions

- Une transaction est une série d'opérations qui apparaissent comme 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

- Java Transaction API
- 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)



```
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 ete 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     }
```

- 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



```
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     ①  public TransfersBean setMonnaie(int monnaie) { ... }
32
33     ①  public TransfersBean setNumCompte(int numCompte) { ... }
34
35     ①  public TransfersBean setSomme(double somme) { ... }
36 }
```

Composants métier





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

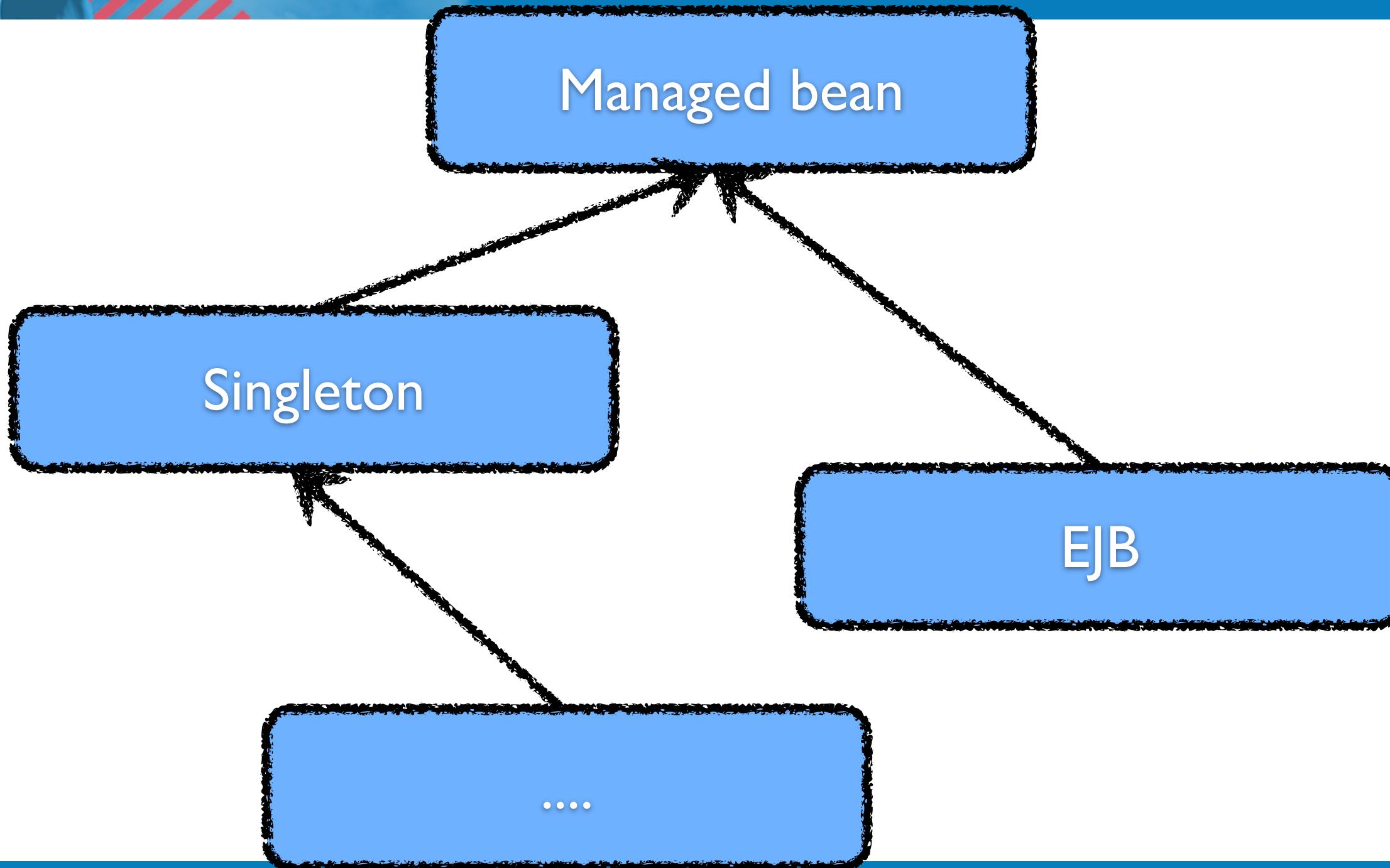


Managed beans

- 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



Les managed beans Managed beans

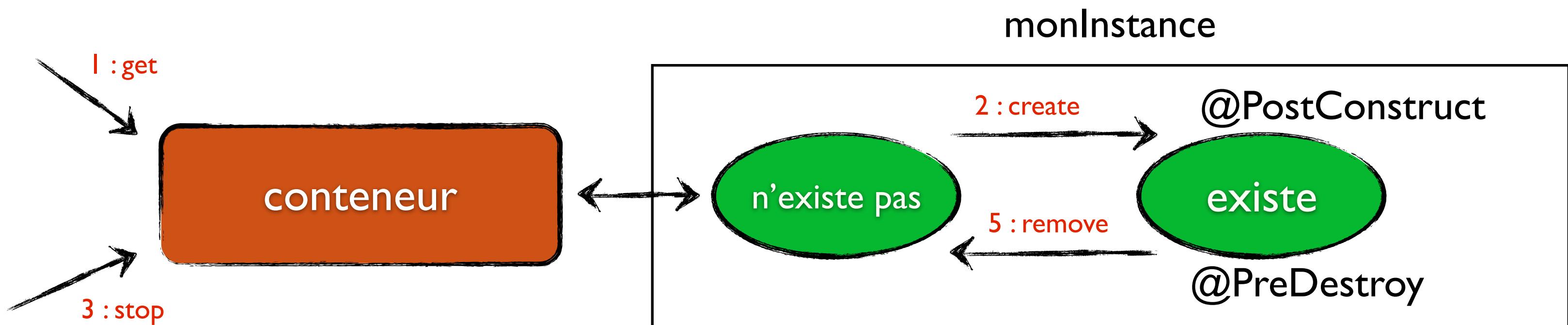




Les managed beans Managed beans

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

Les managed beans Managed beans



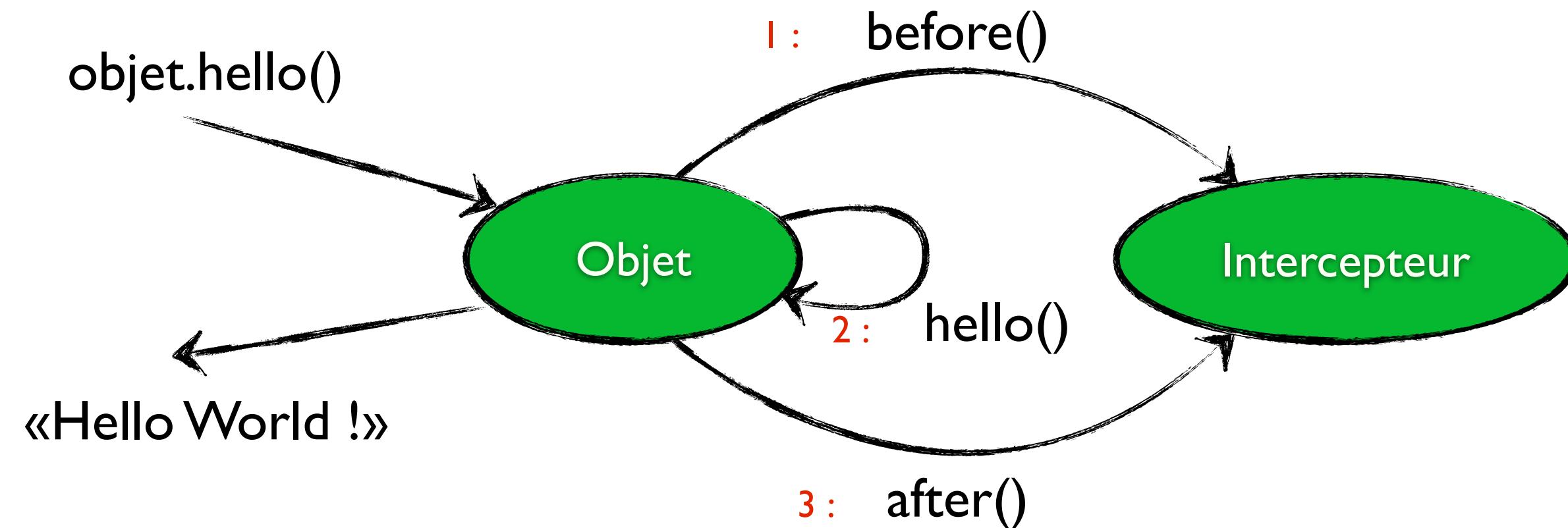


Les managed beans Managed beans

- 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



Les managed beans Managed beans





CDI

- Context and Dependency Injection
- Ensemble de services permettant d'améliorer le 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

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

- 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,

- 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

- 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 {}
```

- 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`

- 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)`

- 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

- 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

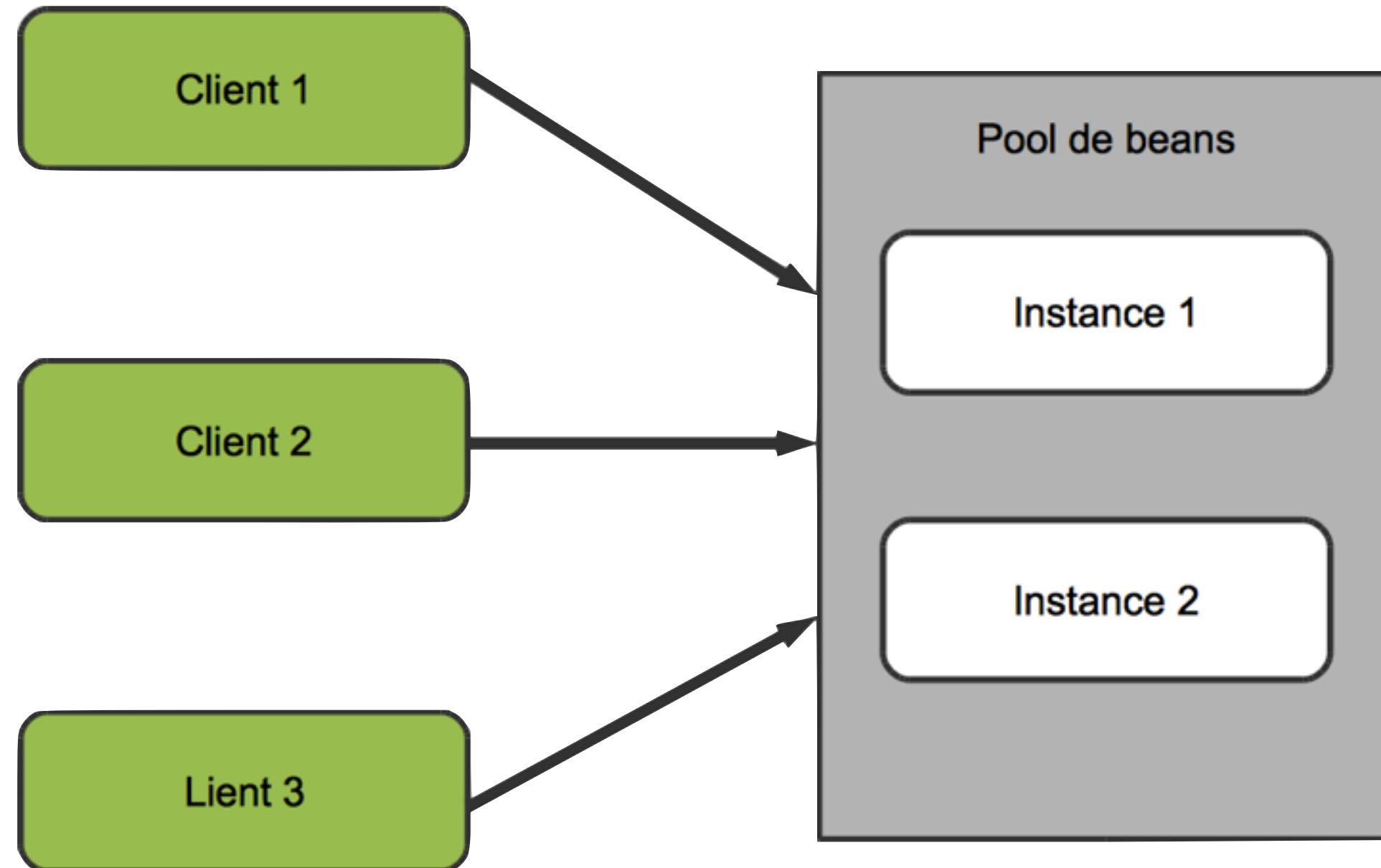
- Conçut pour encapsuler la logique métier
- EJB les plus utilisés
- Deux types
 - Stateless
 - Statefull



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

```
@Stateless  
public class HelloBean {  
    public void sayHello() {  
        System.out.println("Hello World !");  
    }  
}
```



Stateless

```
public class OtherBean {  
    @EJB HelloBean helloBean;  
    public void doSomething() {  
        helloBean.sayHello();  
    }  
}
```

```
public class OtherBean {  
    @Inject HelloBean helloBean;  
    public void doSomething() {  
        helloBean.sayHello();  
    }  
}
```

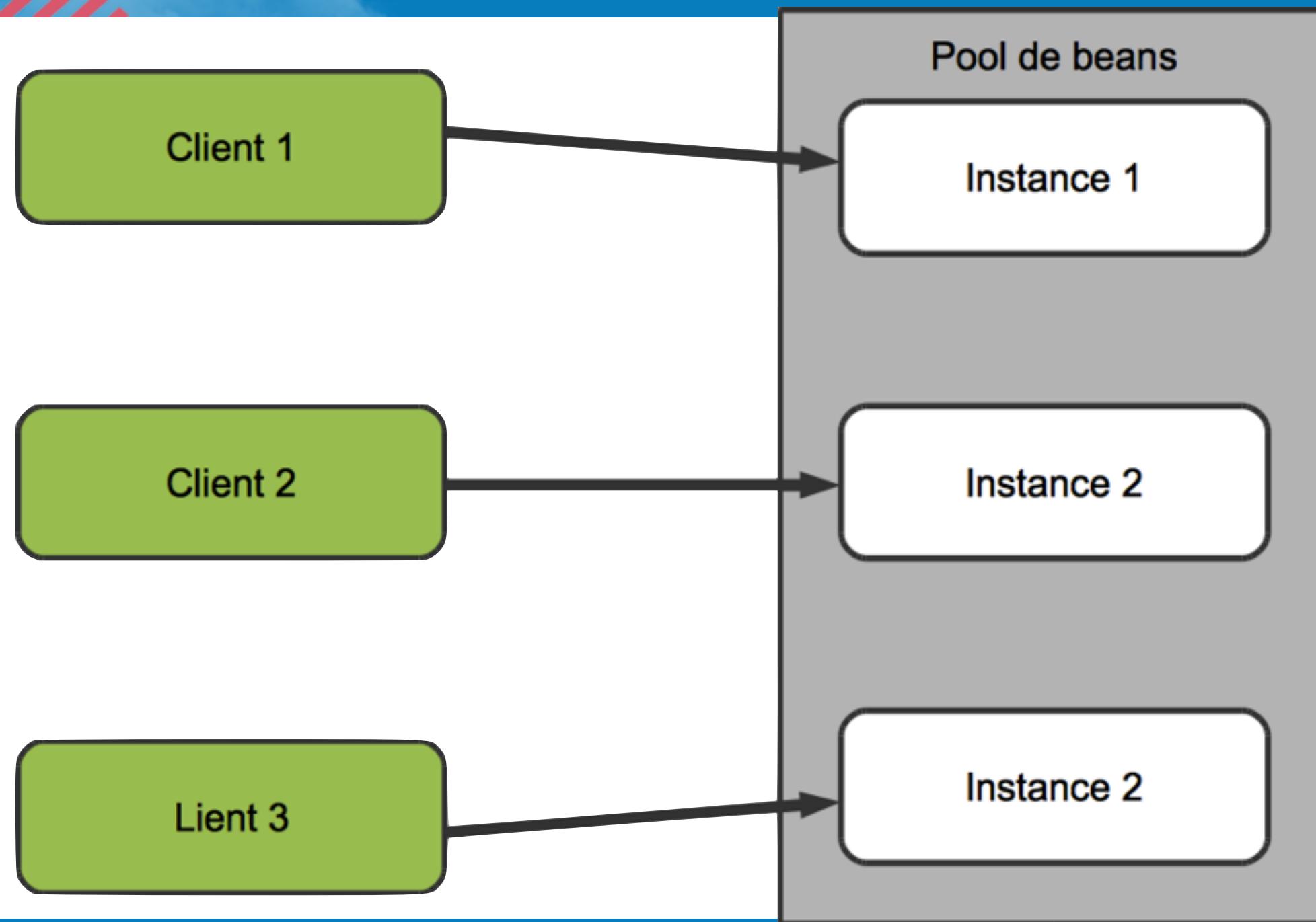
Stateful



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

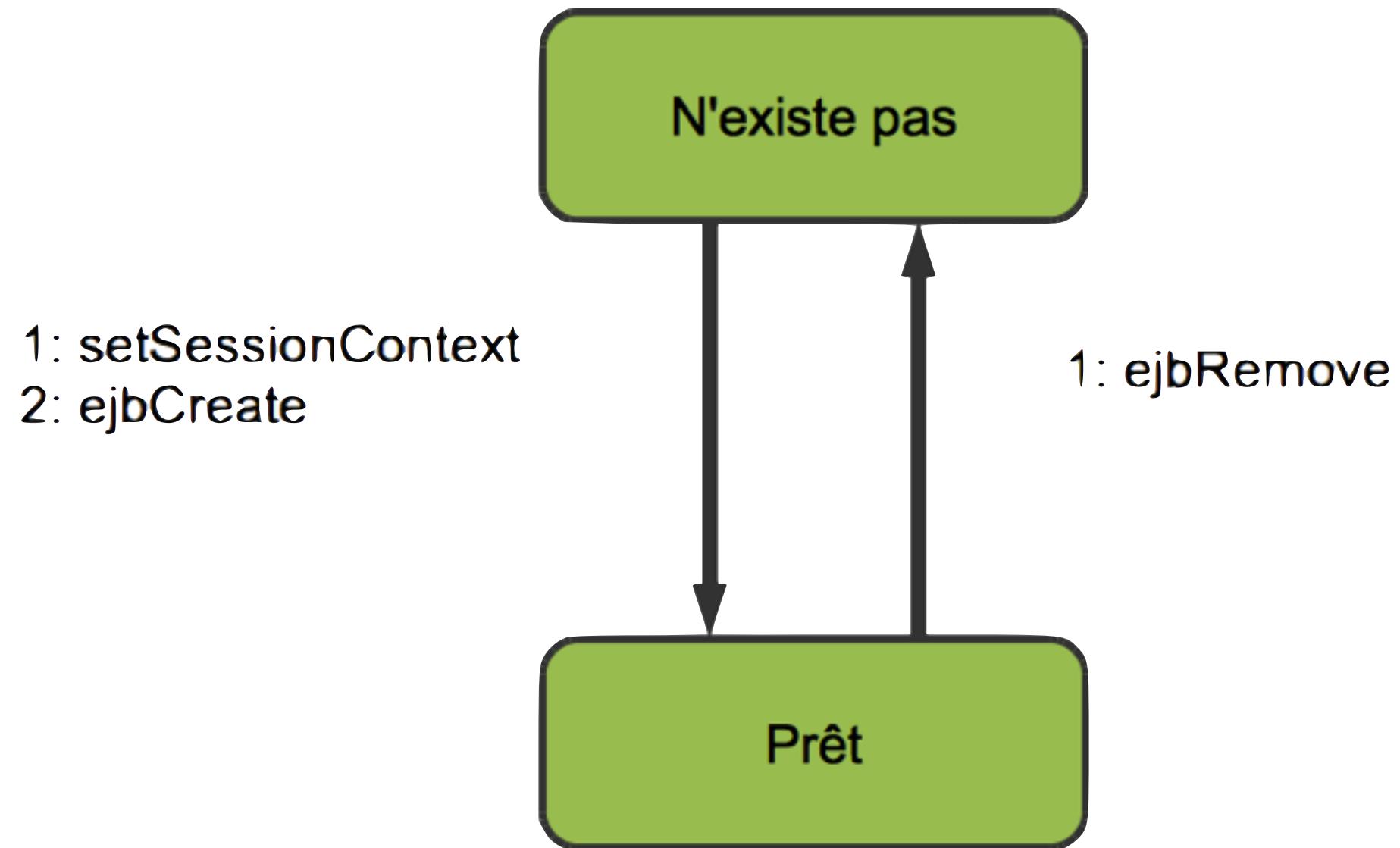
```
@Statefull
public class HelloBean {
    private String name;
    public void sayHello() {
        System.out.println("Hello " + name + " !");
    }
    public void setName(String name) {
        this.name = name;
    }
}
```



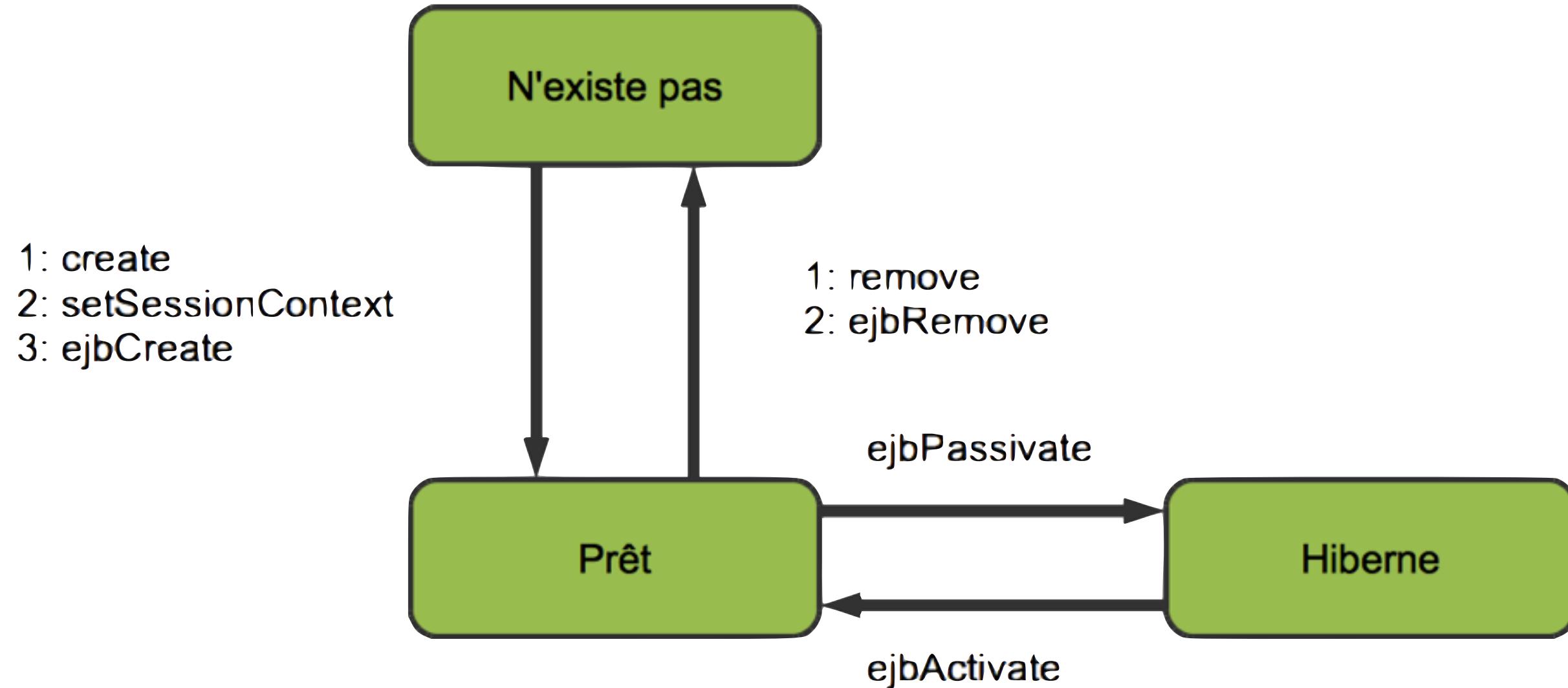
Cycle de vie

- 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



- 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

- 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



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



JMS

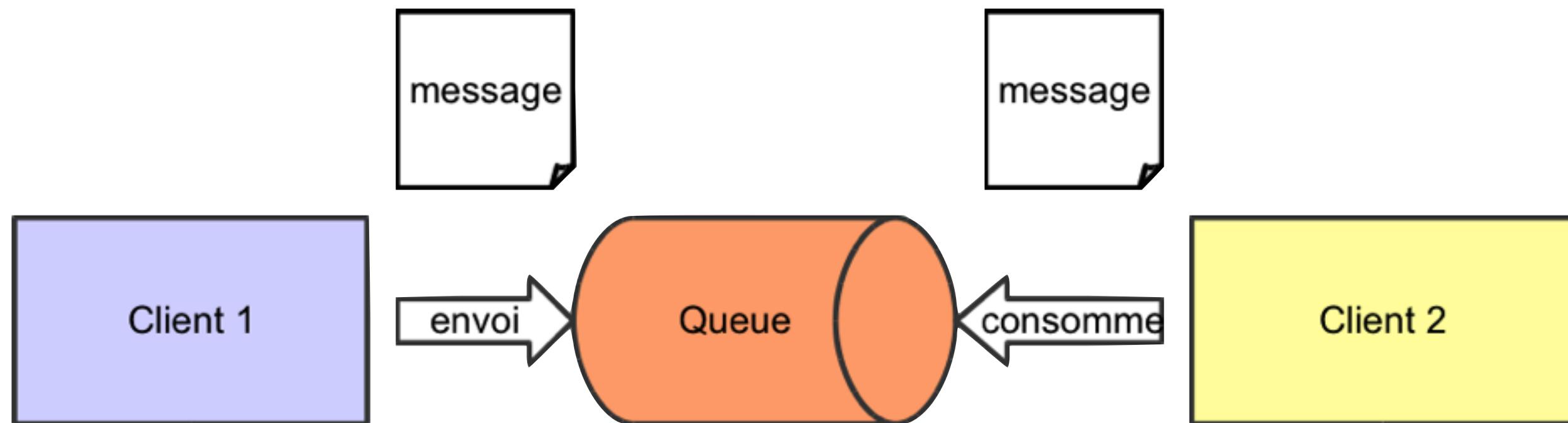
- Java Message System
- Spécification permettant la manipulation de messages distribués
- Programmation faible couplage
- Consommation synchrone ou asynchrone
- Distribution fiable
 - Un message est garanti d'être délivré une et seulement une seul fois



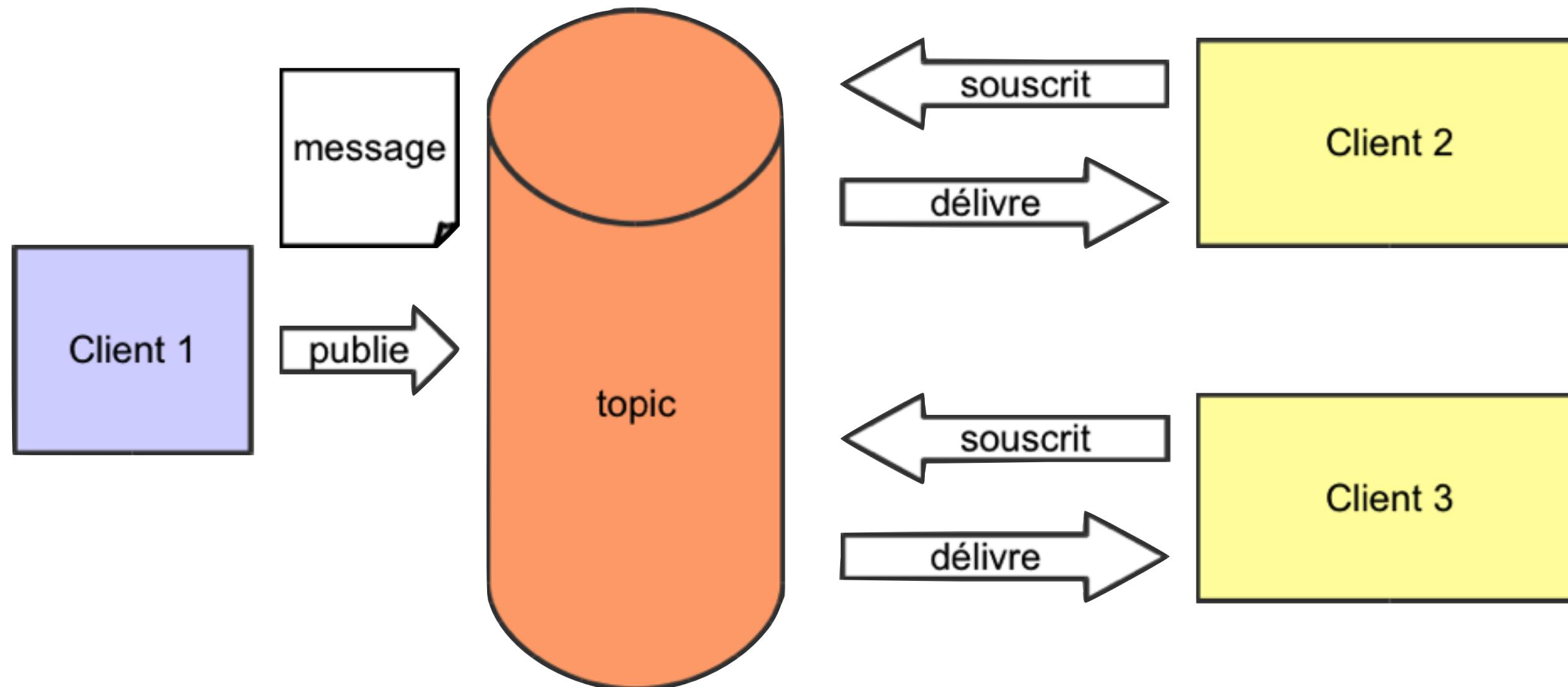
Types de topologie de communication

- 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

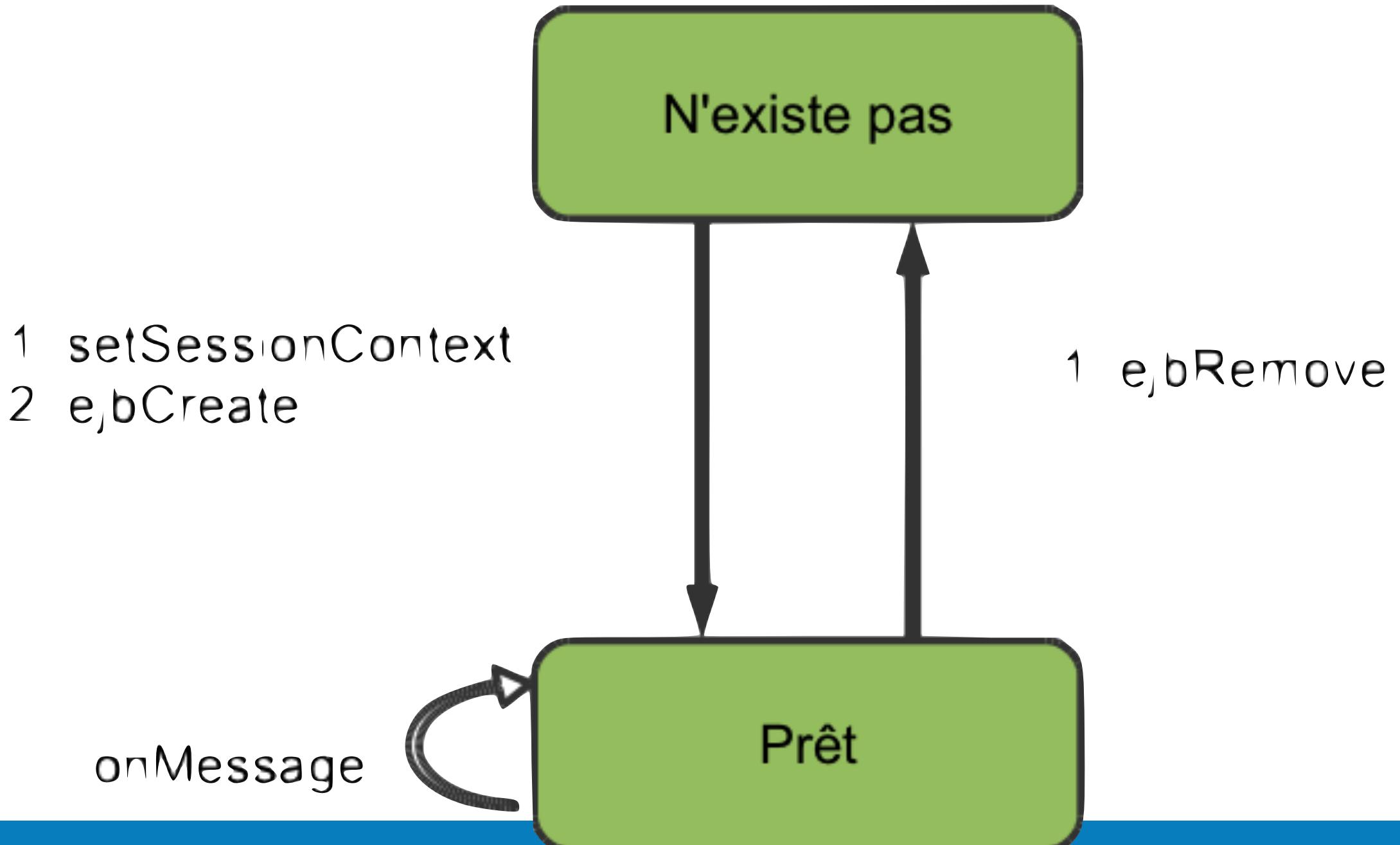


```
45  public void send() {
46      try {
47          TextMessage m = session.createTextMessage();
48          m.setText("just another message");
49          producer.send(m);
50      } catch (JMSException ex) {
51          Logger.getLogger(JMSExemple.class.getName()).log(Level.SEVERE, null, ex);
52      }
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 (JMSException ex) {
63          Logger.getLogger(JMSExemple.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  }
```



```
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      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 (JMSException ex) {
25                  System.out.println(ex);
26              }
27          }
28          System.out.println(txt);
29      }
30  }
```

Cycle de vie





EJB Lite

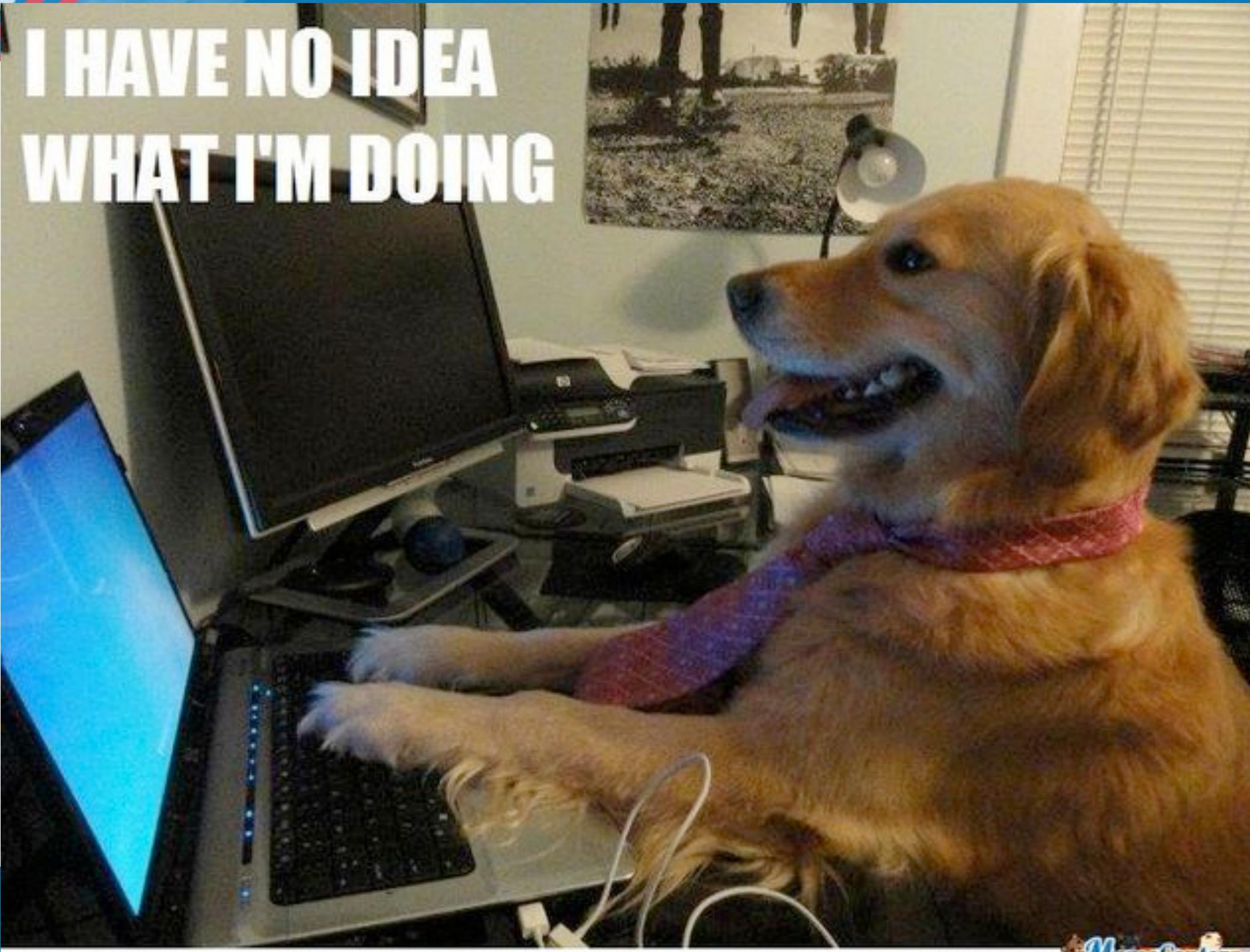
- Sous ensemble de l'api EJB
- Utilisé dans le web profile
- Fonctionnalités
 - local session beans
 - persistance + JTA
 - sécurité
 - injection de dépendance
 - intercepteurs

Couche UI/Web





Mais c'est quoi le web ???



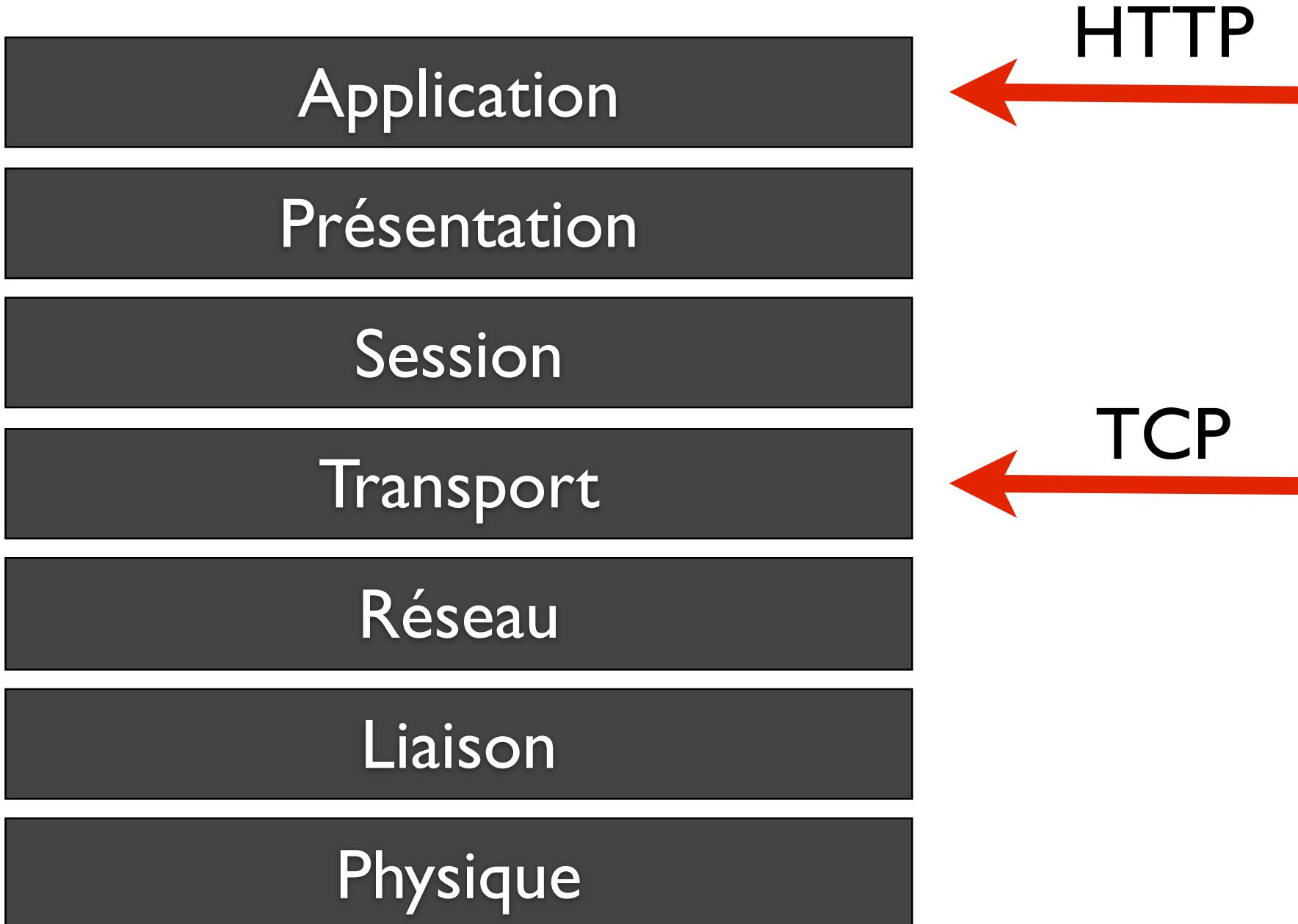
HTTP



- Protocole de communication client/serveur développé pour le web
- Protocole de la couche application
- est censé fonctionner sur n'importe quel type de connexion fiable
- dans les faits, basé sur TCP



HTTP





- Par défaut utilise le port 80
 - port 443 pour le mode sécurisé (https)
- Au niveau du client, le plus connu est le navigateur
 - outils ligne de commande : curl, wget, etc ..
 - librairies
 - download managers, aspirateurs, etc ...



Un peu d'histoire

- HTTP a été inventé par Tim Berners-Lee au CERN
 - avec entre autre, le concept d'adresse web, HTML
 - création du web
- Apparemment inventé dans un bureau sur la partie française du CERN ;-)
- Depuis, standard IETF (RFC 1945, 2068, 2616)

HTTP



- Quatres méthodes / verbes disponibles en HTTP
 - GET
 - POST
 - PUT
 - DELETE

HTTP



- Autres méthodes moins fréquentes
 - HEAD
 - OPTIONS
 - CONNECT
 - TRACE



HTTP's anatomy

- Requête HTTP

- Ligne de commande (Commande, URL, Version)

- En-tête de requête

- [Ligne vide]

- Corps de requête

- Réponse HTTP

- Ligne de statut (Version, Code-réponse)

- En-tête de réponse

- [Ligne vide]

- Corps de réponse



HTTP's anatomy

```
GET /page.html HTTP/1.0
Host: example.com
Referer: http://example.com/
User-Agent: Firefox/Gecko 11.0
```

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Server: Apache/0.8.4
Content-Type: text/html
Content-Length: 59
```

```
<html><body><p>Hello World!</p></body></
html>
```



Paramètres HTTP

- Comment passer des paramètres à une ressource HTTP
 - GET : passer les paramètres dans l'URL
 - <http://site.com/index?a=1&b=2&c=3>
 - POST, PUT : passer les paramètres dans le corps de la requête

```
POST /index HTTP/1.0
Host: site.com
User-Agent: Firefox/Gecko 11.0
```

a=1&b=2&c=3



Paramètres HTTP

- Possibilité de passer autre chose dans le corps de la requête

POST /index HTTP/1.0

Host: site.com

User-Agent: Firefox/Gecko 11.0

```
<root><a>1</a><b>2</b><c>3</c></  
root>
```

POST /index HTTP/1.0

Host: site.com

User-Agent: Firefox/Gecko 11.0

{a: 1, b: 2,c: 3}



Paramètres HTTP

- Il est également possible de passer des paramètres qui font partie intégrante de l'URL
- <http://myblog.com/posts/1>
- <http://myblog.com/posts/1/comments>
- <http://myblog.com/posts/1/comments/30>
- <http://myblog.com/authors/mathieu>

- Style d'architecture apparut avec l'invention du web
- Fournit la définition de «ressources»
- localisation de quelque chose n'importe où dans le monde (coord. GPS)
- représentées par les URLs
- une ressource possède une ou plusieurs représentations
 - html, xml, json, texte, image, etc ...



- URL => nom
 - universel et unique
 - contrairement aux machines
- Méthodes HTTP => verbes
 - notion de polymorphisme des verbes
 - on applique le même verbe sur les différents noms
 - contrairement à d'autres protocoles où chaque ressource expose ses propres méthodes



- Sur une ressource particulière
 - GET => récupérer une représentation de la ressource
 - POST => mise à jour d'une ressource (valeurs dans le corps de la requête)
 - PUT => création d'une ressource (valeurs dans le corps de la requête)
 - DELETE => suppression d'une ressource



- Le sens des URLs

- <http://www.myapp.com/mycompany/user/>
- <http://www.myapp.com/blog/post/123>
- <http://www.myapp.com/blog/post/123/comment/12>
- Bookmarkable, lisible, échangeable, etc ...



REST

<http://tomayko.com/writings/rest-to-my-wife>



Connecté vs. déconnecté

- HTTP est un protocole connecté
 - ouverture d'une socket vers le serveur
 - écriture de la requête dans la socket
 - lecture de la réponse depuis la socket
 - fermeture de la socket
- Le serveur traite une connexion, puis l'oublie
 - traite chaque requête unitairement
 - protocole HTTP



Stateless

- Suivant le style d'architecture REST, le serveur n'est pas censé conserver d'état de l'application en court
- Stateless
- Mais dans une application, il est nécessaire de conserver un état
 - contenu du panier
 - valeurs d'un formulaire
 - etc ...

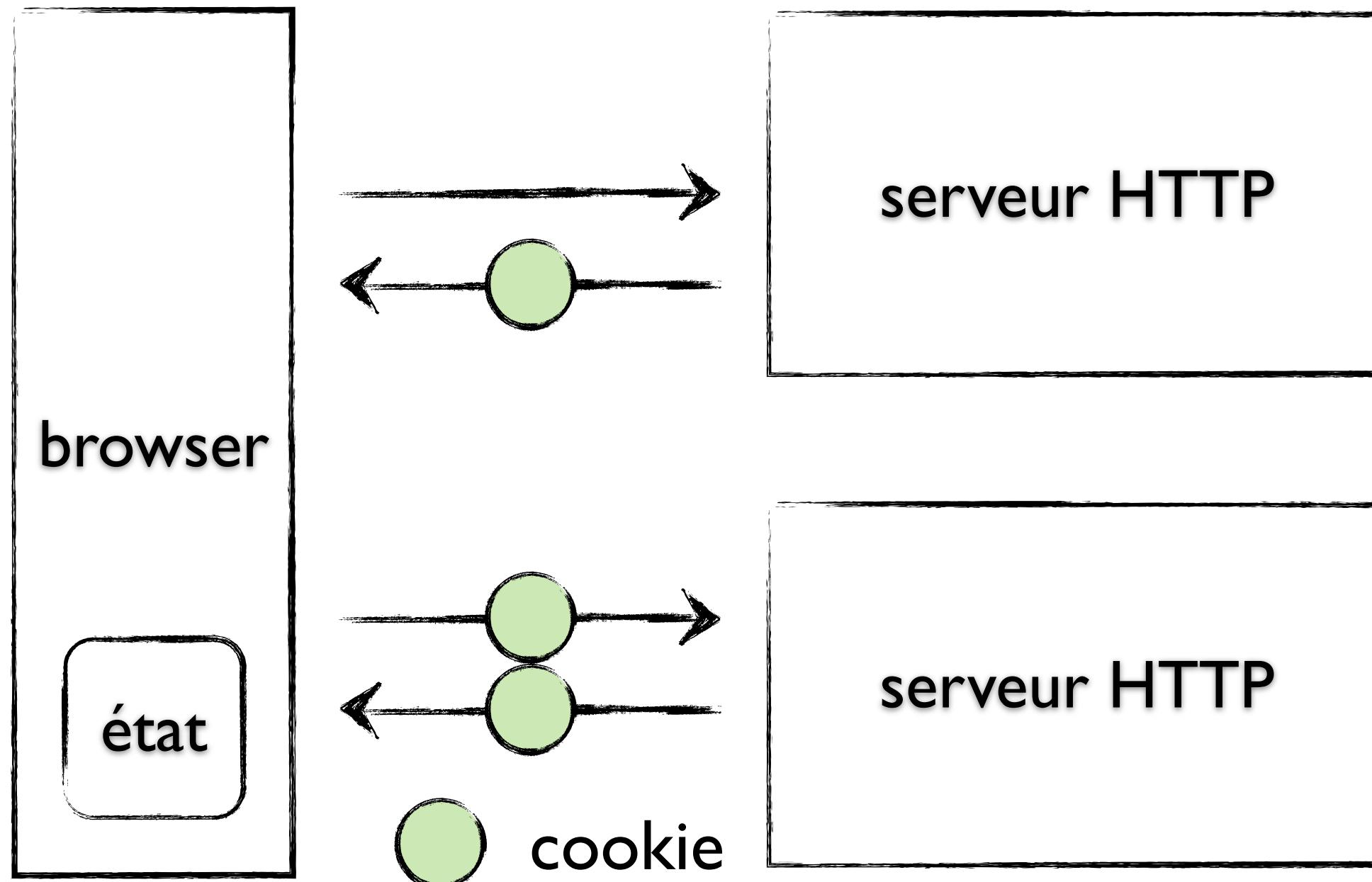


Cookies



SERLi

Cookies





Cookies

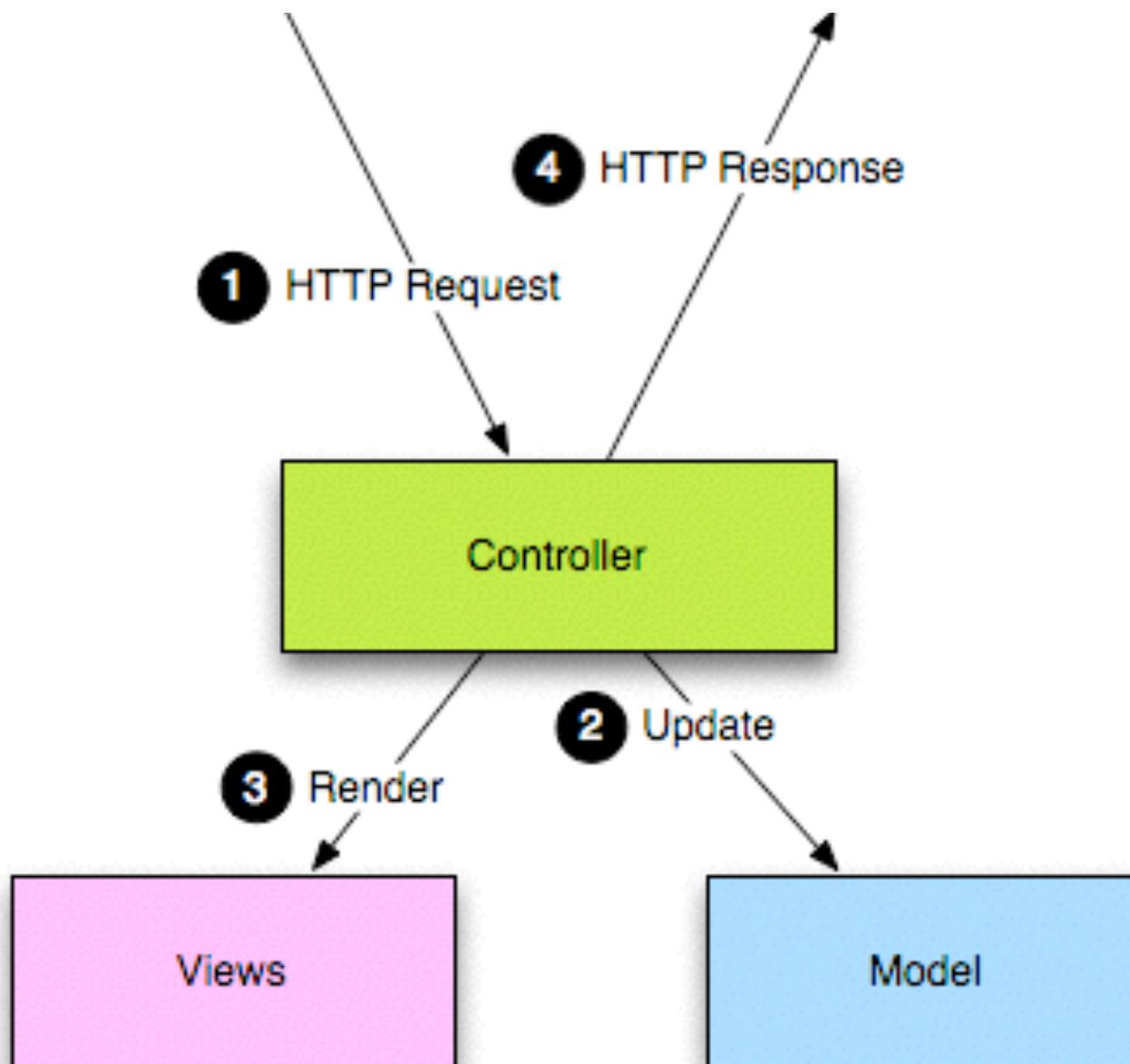
- Suite de données nommée envoyée par le serveur puis renvoyée par le navigateur à chaque requête suivant certaines conditions
- Stocké au niveau du client
 - pas plus de 4096 octets
 - Durée de validité
 - Session (onglet courant)
 - Heures, jours, mois, années ...



Cookies

- Très utile pour gérer l'état de l'application
- Un cookie contient peu d'informations
 - stocker des ids dans les cookies
 - récupérer les données au moment opportun
 - stocker directement les petites données

MVC





- Le routeur
 - Sorte de super contrôleur capable de faire correspondre un pattern d'URL et un type de méthode HTTP à une action

```
(GET, /index) => Application.index()
(GET, /add/{id}) => Base.add(id)
```



- Le contrôleur
 - Unité de code (souvent une classe) comprenant une ou plusieurs actions



- Des actions
 - Contenues dans les contrôleurs, les actions correspondent à une URL et une méthode HTTP
 - Une action est déclenchée par une méthode HTTP, exécute du code métier et retournent une vue (où une représentation de la ressource)

- Des modèles
 - Objets de tout types, reliés à une base de données, un fichier, ou autre
 - Représentent les données à afficher dans une vue (ou une représentation)

- Des vues / représentations
 - Templates textuels
 - Binaires
 - Données structurées ou semi-structurée
 - XML
 - JSON
 - YAML
 - autre

HTML



- Langage de données conçu pour représenter des pages web
- Permet d'écrire de l'hyperTEXte (hyperliens)
- Permet d'inclure des ressources multimédia
- Souvent utilisé avec
 - des feuilles de style
 - un langage de programmation



- Langage à balise dérivé du SGML
- La version 4 contient 91 éléments
 - Structure du document, sémantique
 - Listes, tables, liens, inclusion multimedias
 - Formulaires, scripts, cadres, regroupements
 - Style, présentation du texte
 - etc ...



HTML

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
  <head>
    <title>Exemple de HTML</title>
  </head>
  <body>
    Ceci est une phrase avec un
      <a href="cible.html">hyperlien</a>.
    <p>
      Ceci est un paragraphe sans lien.
    </p>
  </body>
</html>
```

HTML



- Base
 - <div>, <p>
 -
 - <table>
 - <form><input>
 - <a>
 - <script>, <style>

- Langage permettant de décrire la présentation de documents HTML et XML
- Séparer la structure de la présentation
 - permet de supporter plusieurs environnement depuis une même source
 - présentation unifiée
 - conception en parallèle



CSS

```
a {  
    color:#FF0000;  
}  
  
.btn {  
    background:#000000;  
    height: 30px;  
    color:#FFFFFF;  
}  
  
#bouton23 {  
    background:#123456;  
    height: 30px;  
    color:#543211;  
}
```



JavaScript

- Langage de programmation orienté script
- Crée pour surfer sur la vague du succès de Java et pour contrer les API propriétaires de Microsoft
- Langage orienté objets à prototype
- Langage avec une incursion dans la programmation fonctionnelle
- Indispensable de nos jours si on fait du web
 - Typescript
 - Coffescript
 - etc ...



Manipulation du DOM

- Javascript est capable de parcourir et manipuler l'arbre représentant la page HTML
 - à ce niveau cet arbre est complètement dynamique
 - Javascript est donc capable de modifier l'arbre DOM en temps réel
- Utilisation simplifiée via des frameworks



Manipulation du DOM

```
<div id="valeur">Hello World ! </div>
<button id="goodbye" href="#"
        Goodbye</button>
```

```
$( '#goodbye' ).click( function(e) {
    $( '#valeur' )
        .html( «Goodbye World !» );
} );
```



Templating

- Comme Javascript permet de manipuler le DOM, il est possible d'ajouter des éléments complexes dynamiquement à la page
- Au lieu de faire ca par concaténation de chaînes de caractères
 - librairies de templating en JavaScript
 - Mustache.js



JavaScript UI

- En plus de gérer le côté dynamique de la page, Javascript est capable de gérer des composants graphiques
 - en manipulant le DOM ;-)
- Il existe énormément de librairies permettant de gérer des composants graphiques riches
 - JQuery UI, Dojo, Prototype
 - YUI, MooTools, AllowUI



JavaScript MVC

- MVC côté client
- Lorsque l'UI devient très complexe
 - plus simple à gérer
- Databinding bi-directionnel
 - très très pratique, pas besoin de tout faire à la main
- Ember.js, Batman.js, Knockout.js, Backbone.js, Cappucino, ...
- Angular, React, Mithril, Vue.js,



AJAX ?



DET<



XMLHttpRequest

- La star qui se cache derrière Ajax
- API présente dans Javascript permettant de faire des requêtes HTTP de manière asynchrone
 - appel non bloquant dans Javascript
 - Callback via une fonction
- En le combinant avec la manipulation du DOM
 - Comportement dynamique proche des application de bureau

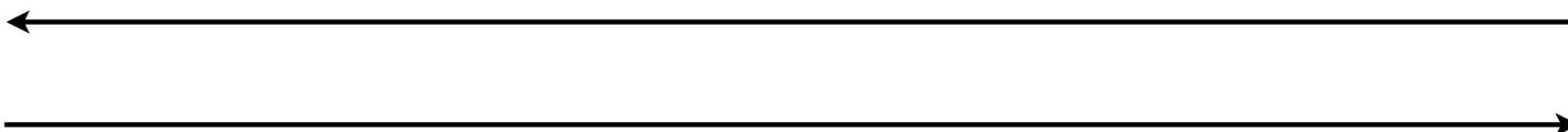


Polling

Serveur

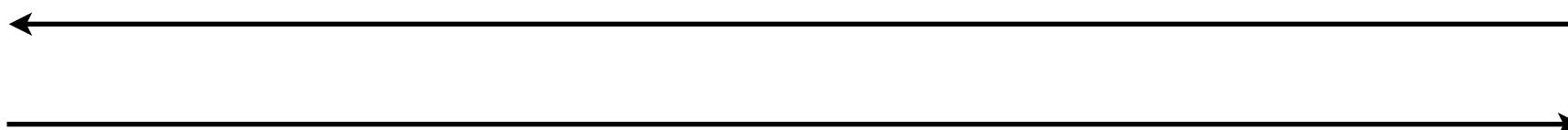
Client

Something new ?



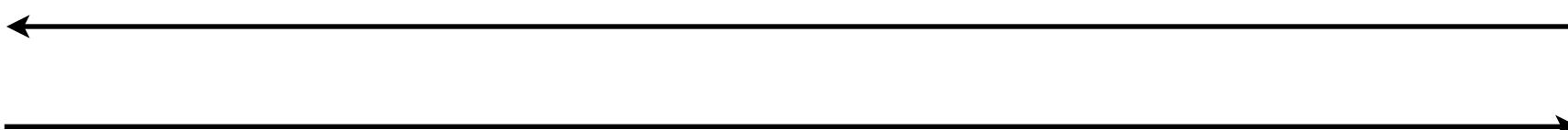
Yes/No

Something new ?



Yes/No

Something new ?



Yes/No

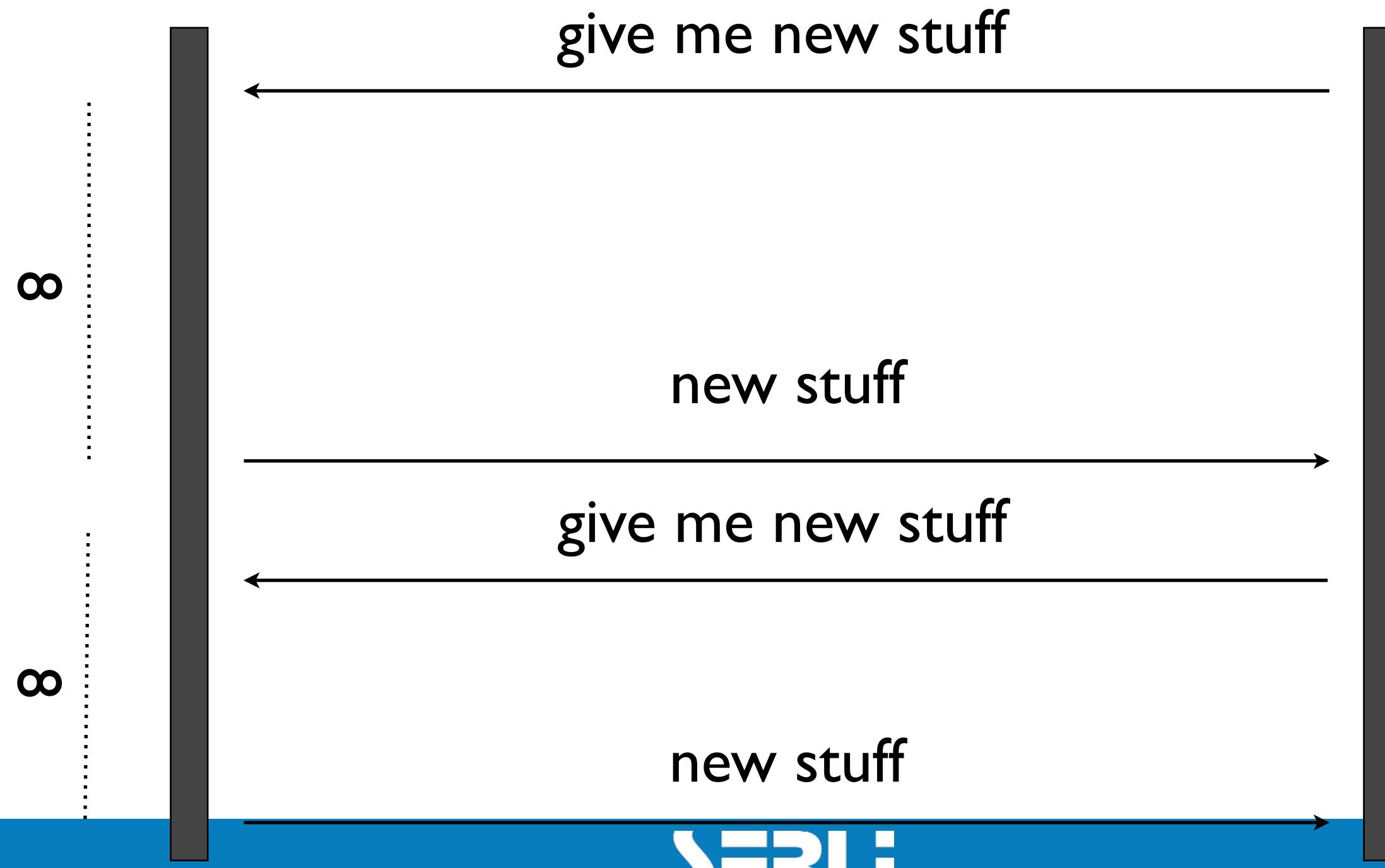
SERLI

Long polling



Serveur

Client



Http Streaming



Serveur

Client





Client riche ?

- Combo HTML/CSS/Javascript considéré comme client riche en face de
 - Flash / Flex
 - JavaFX
 - Silverlight



Le futur

- Les standards du web évoluent
 - certes, lentement
- HTML5
 - nouvelles balises, nouvelles API
 - stockage, temps réel, 2D, 3D
- CSS 3
 - nouvelles capacités
 - transitions



HTML5

**HTML5 = HTML + CSS + JS
+ SVG + WebGL**



- <http://davidwalsh.name/canvas-demos>
- <http://codepen.io/soulwire/pen/Ffvlo>
- <https://developer.mozilla.org/fr/demos/tag/tech:html5>
- <http://ancelin.org/cardboard>



Storage

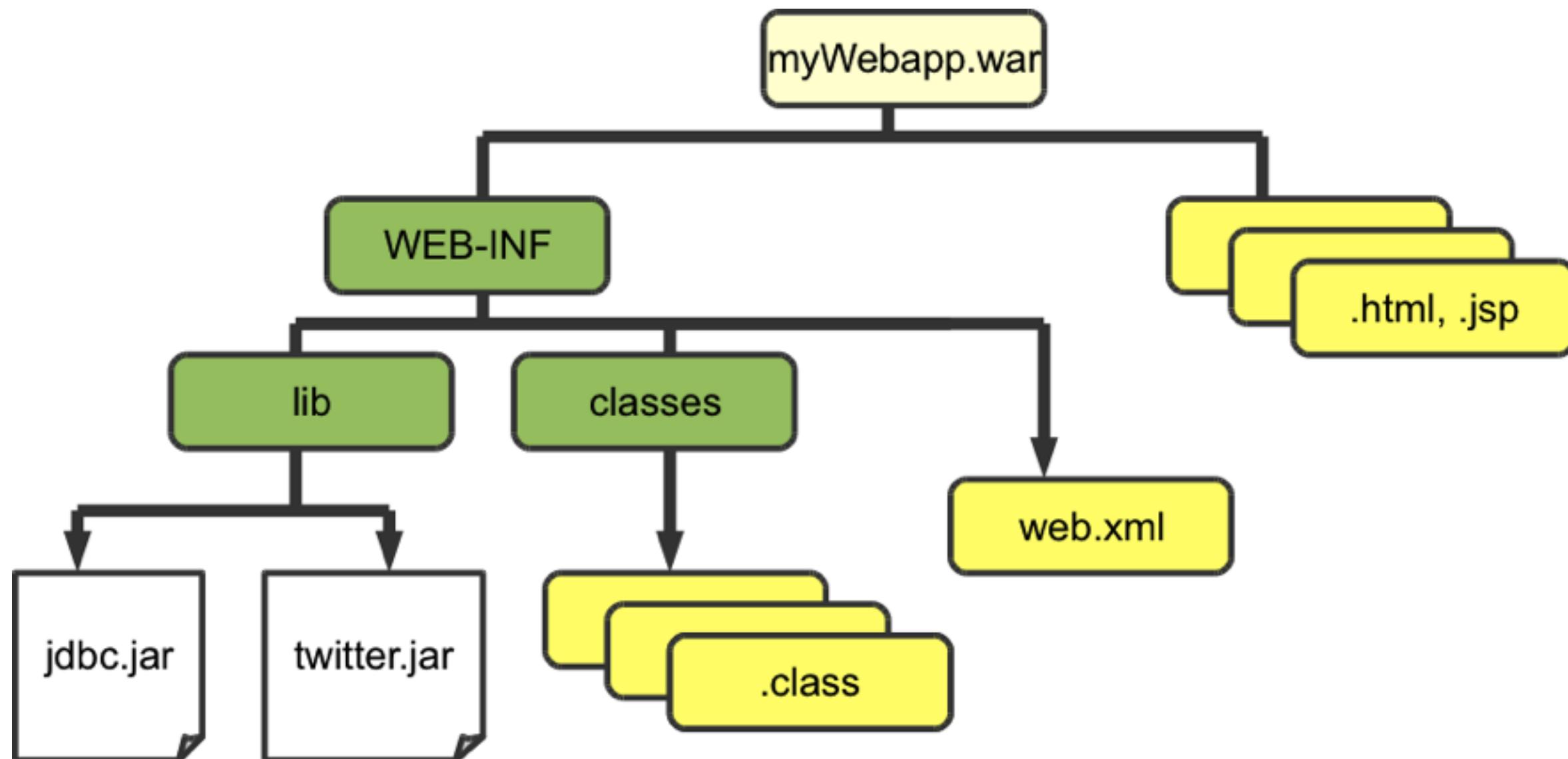
- Session storage
- Local storage
- Indexed storage



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

- 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

- 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 }
```



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



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



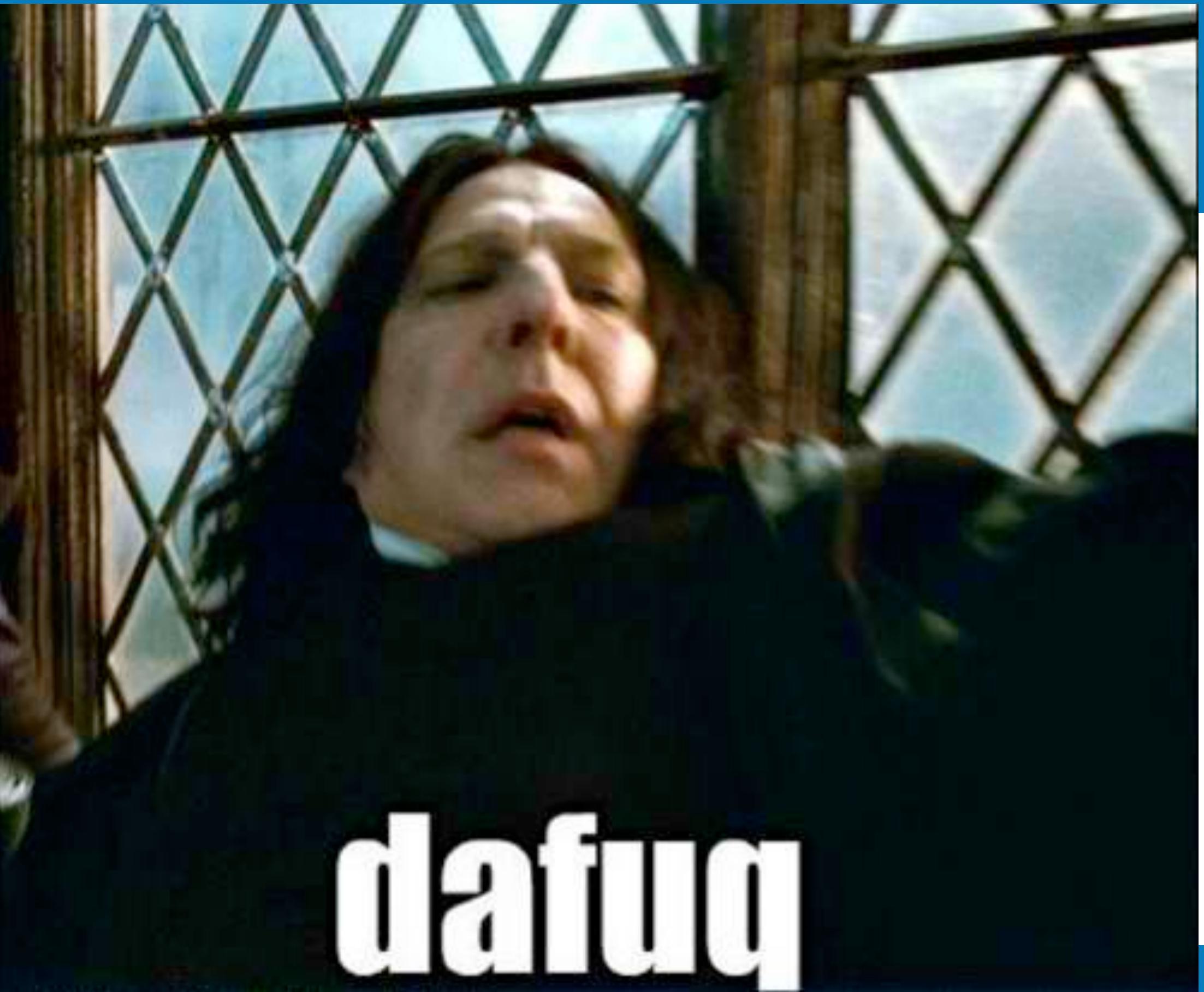
Suivi des utilisateurs

- 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)



dafuq

- 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 !!!



- 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 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 côté serveur
 - La création de nouveaux composants graphiques
 - De générer autre chose que du HTML (XUL, XML, WML, etc ...)



Contrôleur

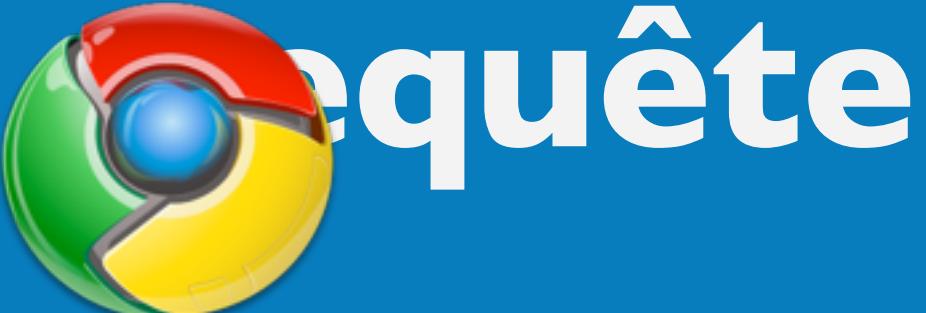
- Managed bean avec un nom
- Propriétés accessibles via getters et setters
 - la vue les utilise pour afficher / mettre à jour les données
- Méthodes retournant une clé de navigation
 - nom du fichier
 - phase de la machine à états

- 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}" />



Fonctionnalités

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



requête

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

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

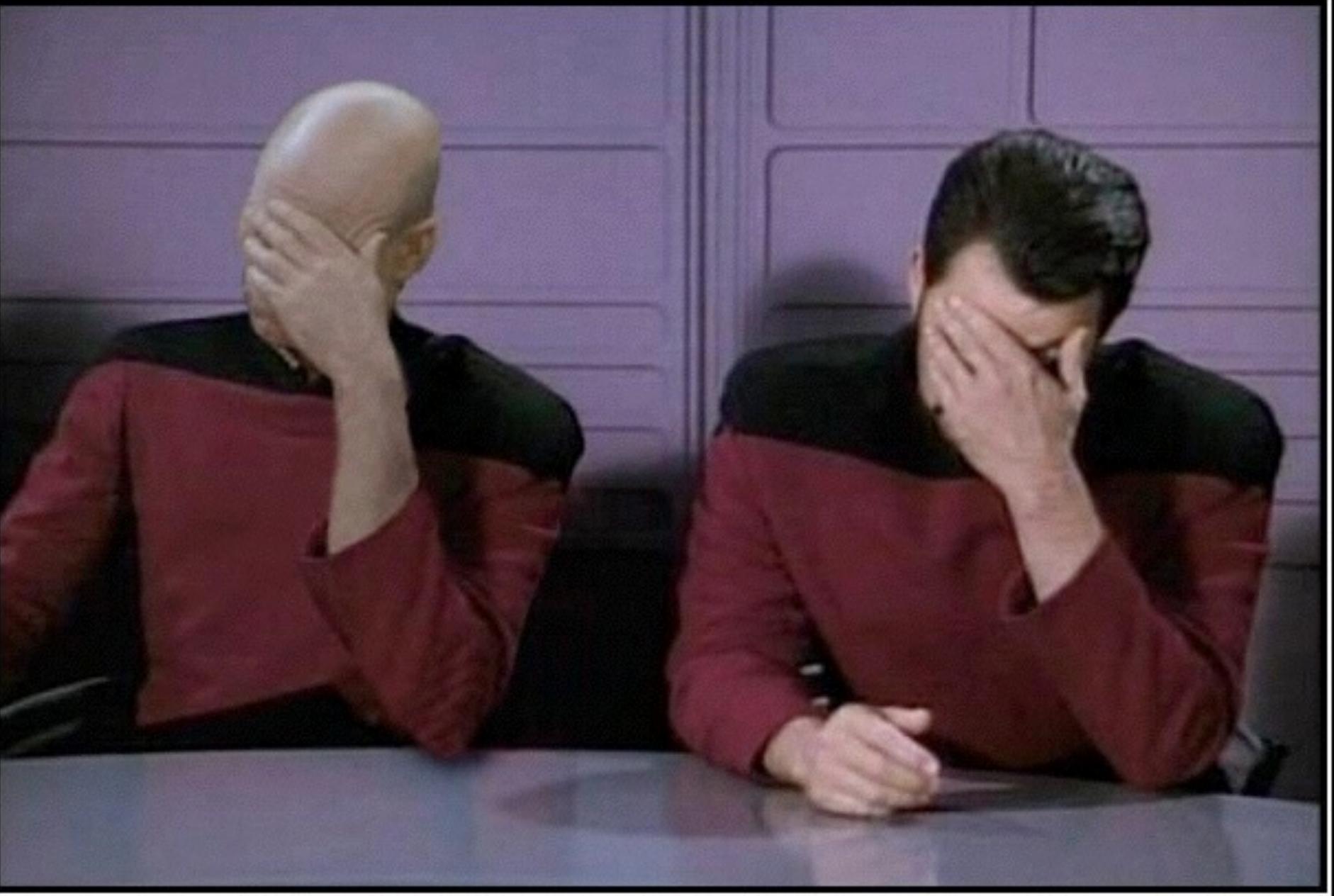
```
    public String search() {  
        items = itemService.search(keyword);  
        return "results.xhtml";  
    }  
    // getters, setters  
}
```

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



search.xhtml

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



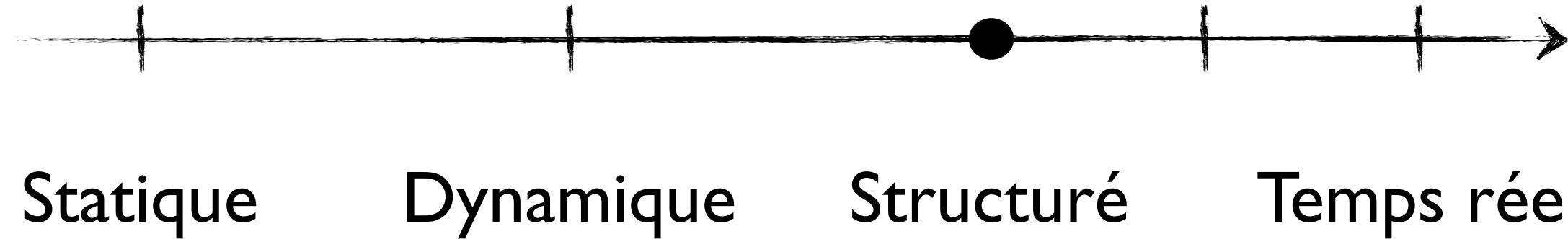
DOUBLE FACEPALM

FOR WHEN ONE FACEPALM DOESN'T CUT IT



Nouvelle ère

- Programmation réactive avec HTTP





HTTP connecté ?

- Programmation réactive avec HTTP
- Depuis HTTP 1.1 une connexion HTTP peut avoir l'option 'Keep-Alive'
- plusieurs requêtes/réponses pour une même connexion
 - streaming
- HTML 5 va profiter au maximum de cette capacité



Server Sent Events

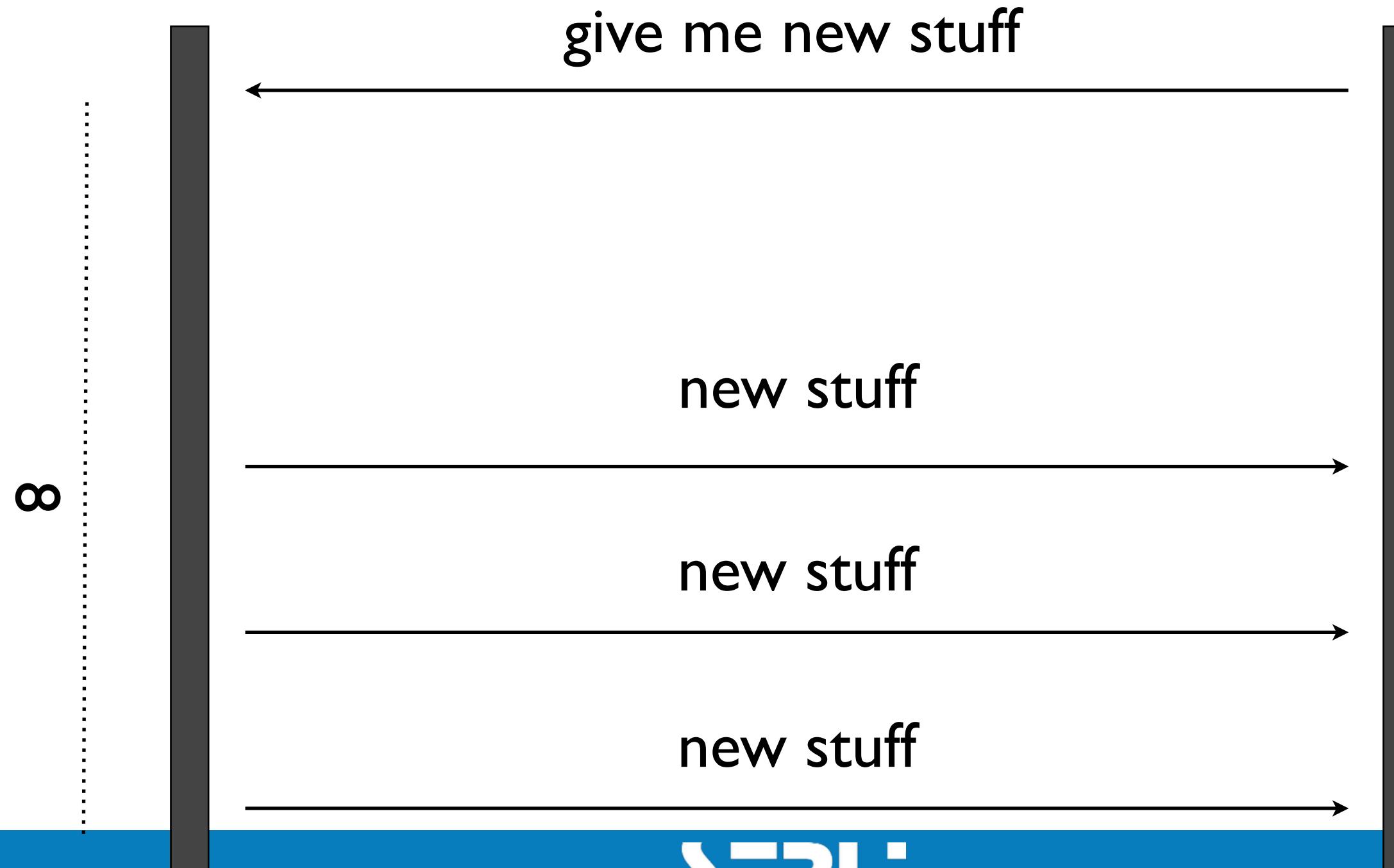
- Standardisation du concept de push serveur mais en mieux
- Le navigateur ouvre une connexion persistante et enregistre des callback Javascript
- Le serveur envoie des informations quand bon lui semble ce qui actionne les callbacks sur le navigateur



Server Sent Events

Serveur

Client





Web Sockets

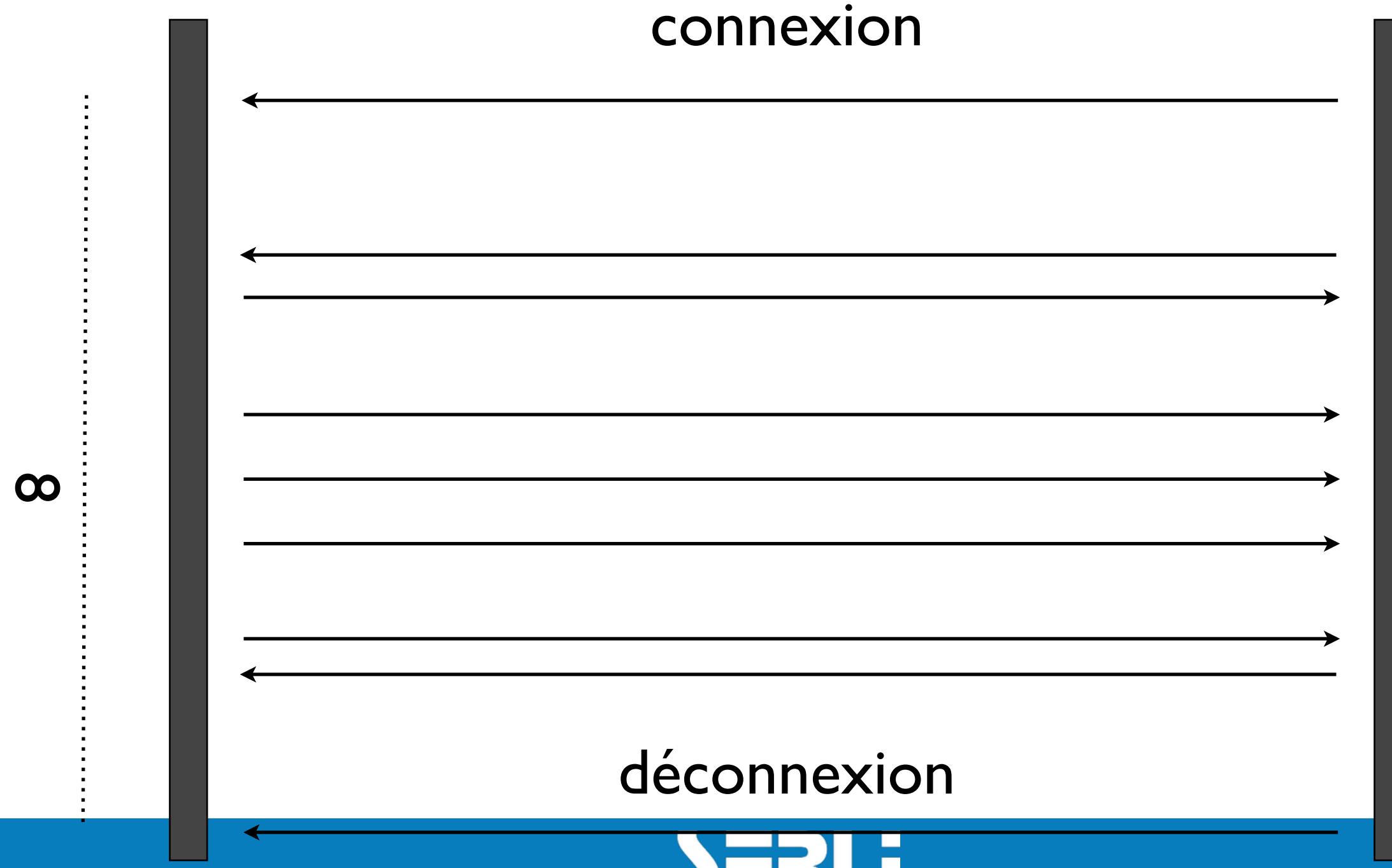
- Ouvre une socket bi-directionnelle entre le client et le serveur
- API Javascript réactive à base de callback
- Privilégier l'envoie d'informations structurées pour ne pas repartir sur les problèmes de formats

Web Sockets



Serveur

Client



WebSockets



```
@ServerEndpoint("/chat")
public class ChatServer {
    Set<Session> peers = ...  
  

    @OnOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }  
  

    @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }
    ...
}
```

WebSockets



```
...  
@OnMessage  
public void message(String message, Session client)  
    throws IOException {  
    for (Session session : peers) {  
        if (!session.equals(client)) {  
            session.getRemote().sendObject(message);  
        }  
    }  
}  
}
```



- <http://fr.lichess.org/world-map>
- <http://fr.lichess.org/games>

Intéropérabilité





Web Services

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



- Java Architecture for Xml Binding
- 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



```
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>
```



JSON

SERLi

JSON



- API apparue dans Java EE 7
- Permet de manipuler des structures de données semblables aux objets JavaScript
 - semi-structurés
 - Schéma dynamique
 - Simple



JSON

```
JsonArray value =  
    Json.createArrayBuilder()  
        .add(Json.createObjectBuilder()  
            .add("type", "home")  
            .add("number", "212 555-1234")  
        )  
        .add(Json.createObjectBuilder()  
            .add("type", "fax")  
            .add("number", "646 555-4567")  
        )  
.build();  
  
[  
  {  
    "type": "home",  
    "number": "212 555-1234"  
  },  
  {  
    "type": "fax",  
    "number": "646 555-4567"  
  }]  
]
```



- 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



```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@Path("api")
public class App extends Application {

}
```



GET /api/hello

```
@Path("hello")
public class HellBean {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String Hello() {
        return "Hello World!";
    }
}
```



GET /api/tasks

```
@Stateless
@Path("tasks")
public class TodoServices {
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Task> allTasks() {
        return Task.findAll();
    }
}
```



GET /api/tasks/2

```
@Stateless
@Path("tasks")
public class TodoServices {
    @Path("{id}") @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Task getTask(@PathParam("id") Long id) {
        Task task = Task.findById(id);
        if (task == null) {
            throw new NotFoundException("Entity with id "
                + id + " not found.");
        }
        return task;
    }
}
```



GET /api/tasks?id=2

```
@Stateless
@Path("tasks")
public class TodoServices {
    @Path("{id}") @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Task getTask(@QueryParam("id") Long id) {
        Task task = Task.findById(id);
        if (task == null) {
            throw new NotFoundException("Entity with id "
                + id + " not found.");
        }
        return task;
    }
}
```



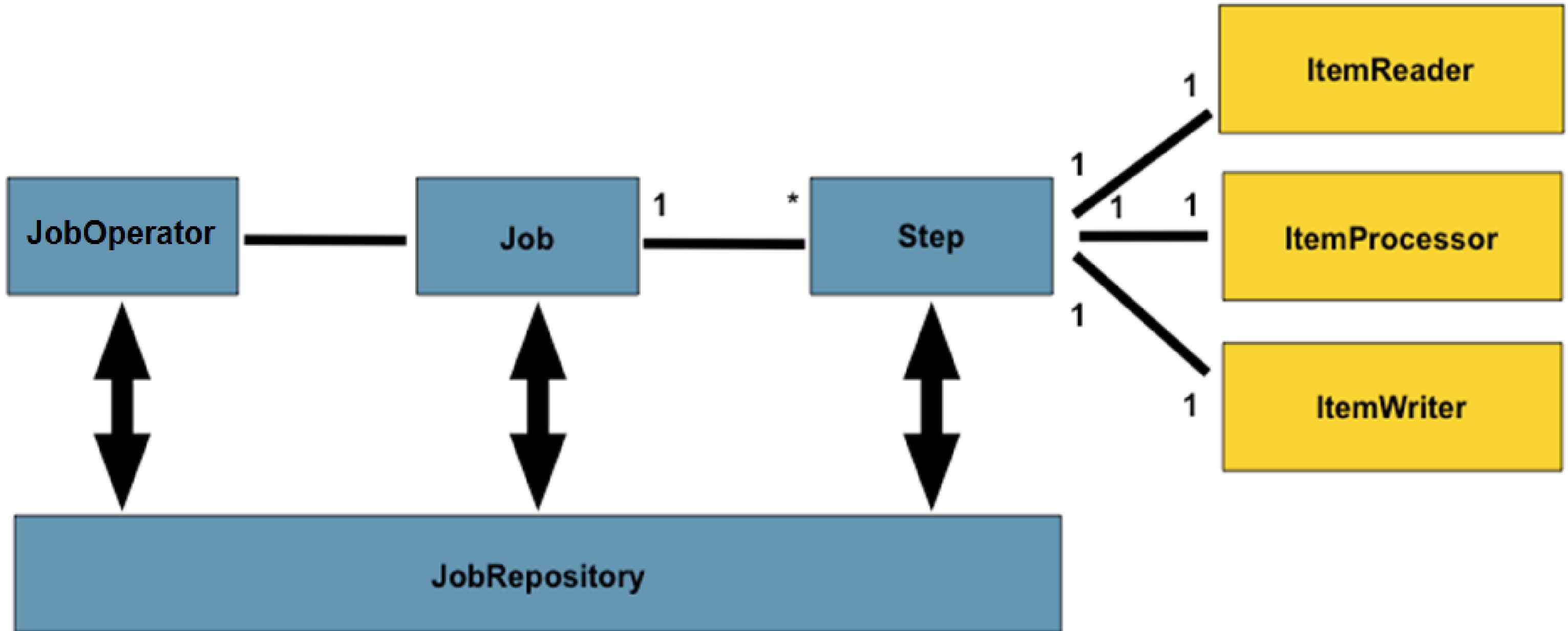
JAX-RS client API

```
Client client = ClientBuilder.newClient();
JsonObject task = client.target("../api/tasks")
    .queryParam("id", 2)
    .request()
    .get(JsonObject.class);
```



Autres

Batch



Batch



```
<step id="sendStatements">
  <chunk reader="accountReader"
    processor="accountProcessor"
    writer="emailWriter"
    item-count="10"/>
</step>
```

```
@Named("accountReader")
... implements ItemReader... {
public Account readItem() {
  // read account using JPA
}

@Named("accountProcessor")
... implements ItemProcessor... {
Public Statement processItems(Account account) {
  // read Account, return Statement
```

```
@Named("emailWriter")
... implements ItemWriter... {
public void writeItems(List<Statements> statements) {
  // use JavaMail to send email
```

Serveur d'application





Serveur d'applications

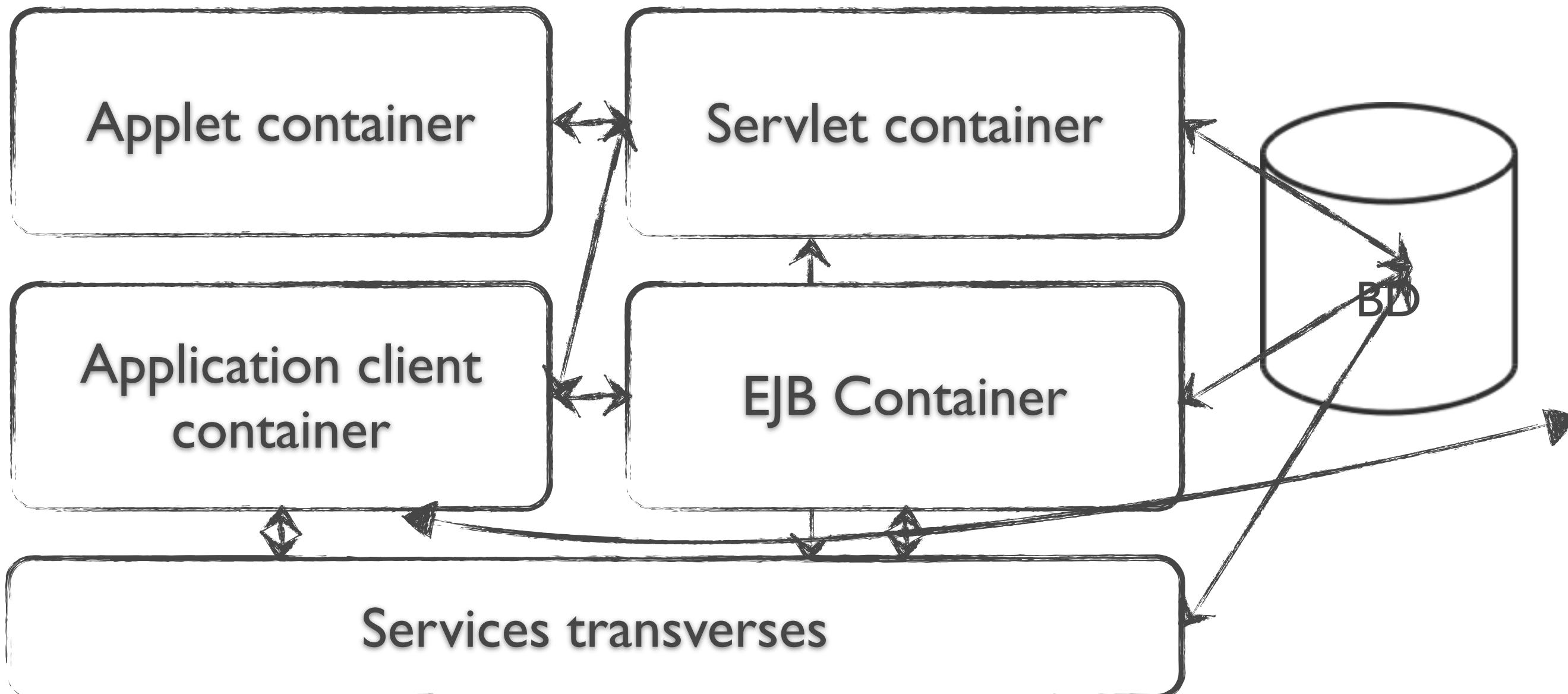
- Expose la logique et les processus métier et les propose à des applications tiers
- Centralise les applications
- Assure la persistance des données au cours et entre différentes transactions
- Assure la persistance des données partagées
- Arbitre l'accès aux ressources (clients concurrents)



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
- ...

Démo



SERLi

TP



This is the end ...

des questions ?

