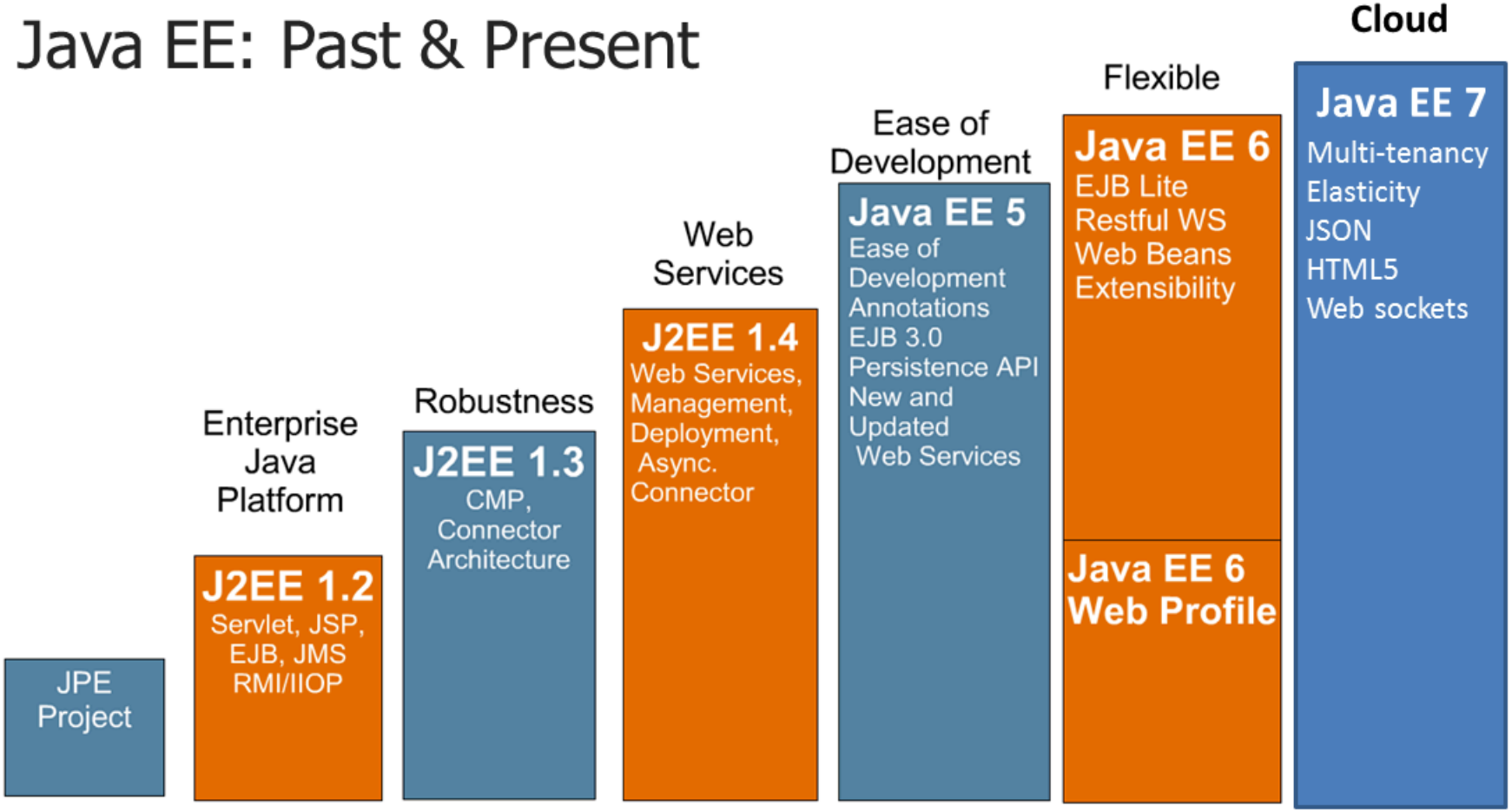
CDI & EJB

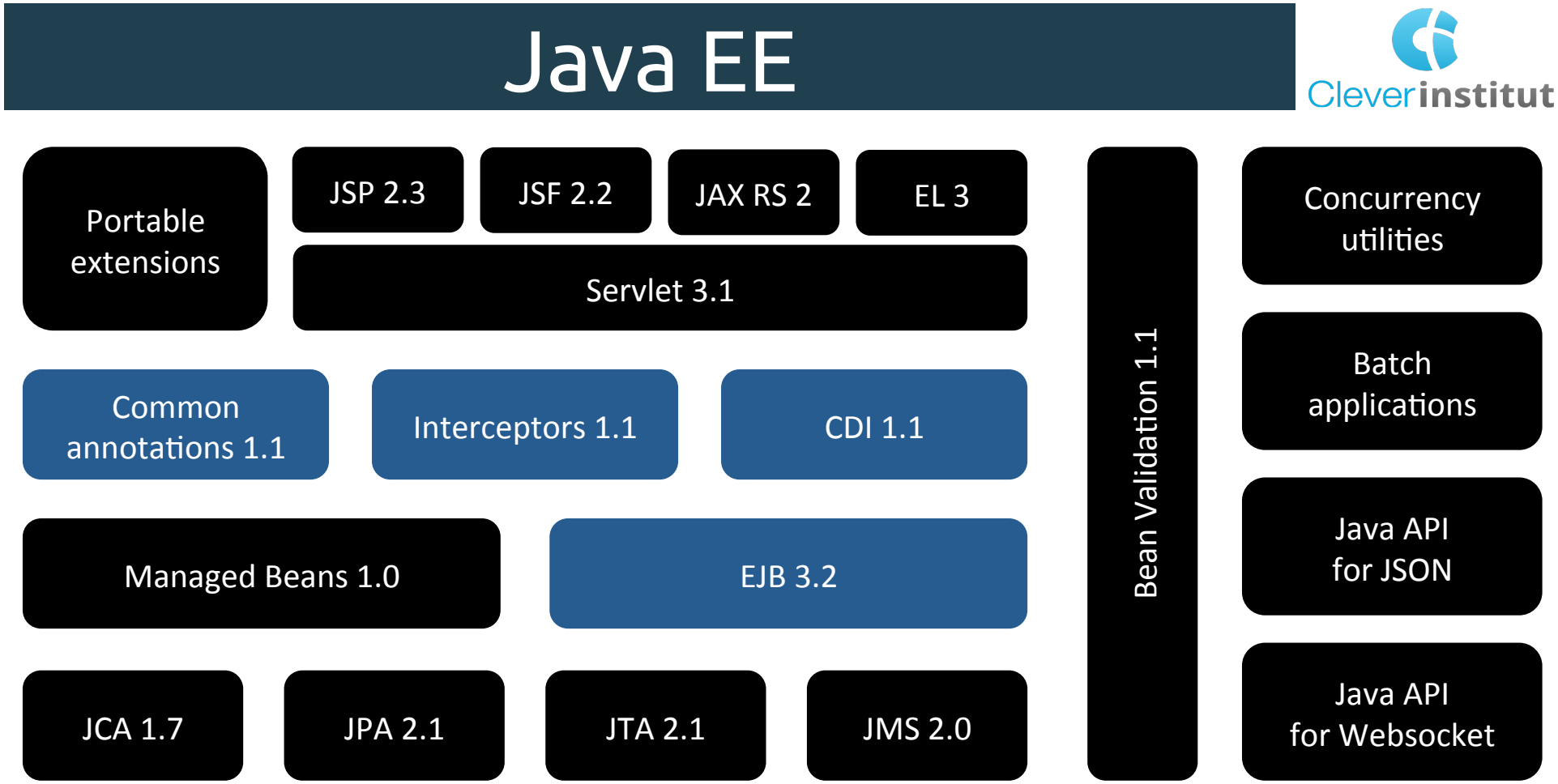
Animé par Rossi Oddet

Java EE

Historique

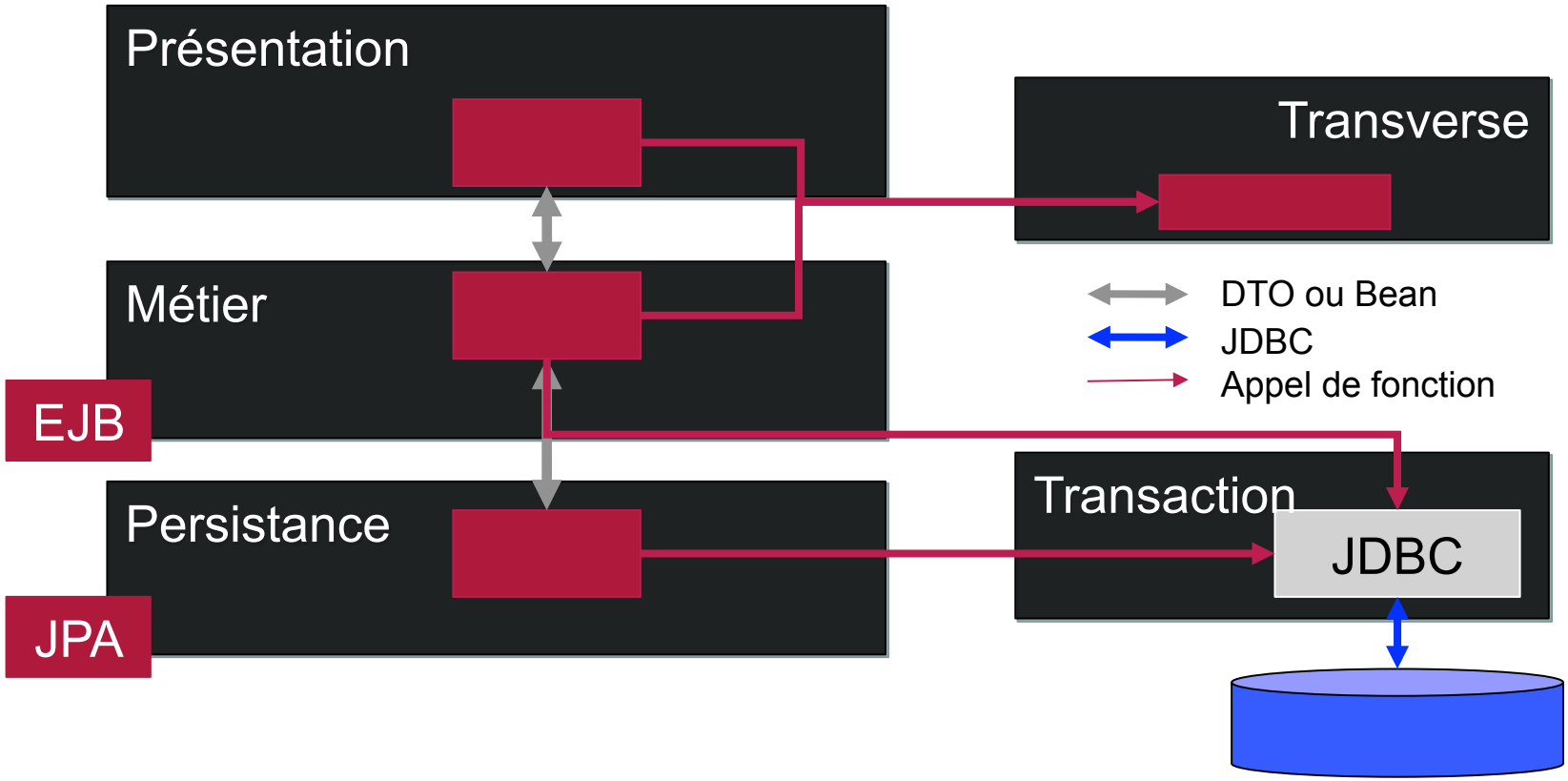

Java EE: Past & Present
JPE Project
Enterprise Java Platform
J2EE 1.2
Servlet, JSP, EJB, JMS RMI/IIOP
Robustness
J2EE 1.3
CMP, Connector Architecture
Web Services
J2EE 1.4
Web Services, Management, Deployment, Async. Connector
Ease of Development
Java EE 5
Ease of Development Annotations EJB 3.0 Persistence API New and Updated Web Services
Flexible
Java EE 6
EJB Lite Restful WS Web Beans Extensibility
Java EE 6 Web Profile
Java EE 7
Multi-tenancy Elasticity JSON HTML5 Web sockets

Java EE


Portable extensions
JSP 2.3
JSF 2.2
JAX RS 2
EL 3
Servlet 3.1
Common annotations 1.1
Interceptors 1.1
CDI 1.1
Managed Beans 1.0
EJB 3.2
JCA 1.7
JPA 2.1
JTA 2.1
JMS 2.0
Bean Validation 1.1
Concurrency utilities
Batch applications
Java API for JSON
Java API for Websocket

Architectures orientées services

Cleverinstitut



CDI

CDI ?

- Context and Dependency Injection
  - Couplage faible avec un typage fort
  - Injection de dépendance avec types
  - Intercepteurs
  - Notifications d'événements
  - Extensions
- Implémentation de référence : Weld
- CDI injecte des "beans"

Bean ?

Cleverinstitut

- Un objet issu d'une classe qui a au moins un des éléments suivants :
  - un constructeur sans paramètres
  - un constructeur annoté avec l'annotation @Inject
  - un producteur de beans

Exemple d'injection

Cleverinstitut

```
public class PizzaService {

    public List<Pizza> findAllPizze(){
        ...
    }
}
```

Injection par type
Qualifier = @Default

```
@WebServlet("/pizze")
public class PizzaWeb extends HttpServlet {

    @Inject private PizzaService pizzaService;
```

Gestion des ambiguïtés


```
public class CommandeService {}

public class CommandeEntrepriseService
                extends CommandeService {}

public class CommandeParticulierService
                extends CommandeService {}

public class PizzaService {

    @Inject CommandeService commandeService;
```

QUEL TYPE INJECTE ?

CommandeService ?
CommandeEntrepriseService ?
CommandeParticulierService ?

@Qualifier


```java
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE, METHOD, PARAMETER, FIELD})
public @interface Entreprise {}

public class CommandeService {}

@Entreprise
public class CommandeEntrepriseService extends CommandeService {}

public class CommandeParticulierService extends CommandeService {}

public class PizzaService {

    @Inject @Entreprise CommandeService service;
```

@Named


```
public class CommandeService {}

@Named
public class CommandeEntrepriseService extends CommandeService {}

public class CommandeParticulierService extends CommandeService {}

public class PizzaService {

    @Inject @Named("commandeEntrepriseService") CommandeService service;
```

@PostConstruct, @PreDestroy


```
@Named
public class CommandeEntrepriseService extends CommandeService {

    @PostConstruct
    public void onPostConstruct() {
        ...
    }

    @PreDestroy
    public void onPreDestroy() {
        ...
    }

}
```

Scope d'un bean CDI

- @RequestScoped
  - Le bean existe le temps d'une requête HTTP
- @SessionScoped
  - Le bean existe le temps d'une session serveur
- @ApplicationScoped
  - Le bean existe tout au long de la vie de l'application
- @Dependent
  - Scope par défaut. Le bean prend le même cycle de vie que le bean dans lequel il est injecté
- @ConversationScoped
  - Le bean vit le temps d'une conversation dans une application JSF.

Intercepteurs


```java
public class LogInterceptor {

	@AroundInvoke
	public Object log(InvocationContext context) throws Exception {

		// traitement avant l'invocation de la méthode

		Object b =  context.proceed();

		// traitement après l'invocation de la méthode

		return  b;
	}
}

@Interceptors(LogInterceptor.class)
public class PizzaService {
```

EJB

Rôle d'un EJB

- Propose d'automatiser ce qui a trait au système :
  - Transactions
  - Sécurité
  - Evolutivité
  - Concurrence
  - Communications
  - Gestion des ressources
  - Persistance
  - Gestion des erreurs
  - ...

3 types d'EJB

Cleverinstitut

- Session
  - composant métier réutilisable
  - Stateless
    - 1 instance par appel
  - Stateful
    - 1 instance par session
- Entity
  - JPA
- Message-Driven
  - JMS

Configuration ejb-jar.xml

- Facultatif
- Dans META-INF
- Priorité aux annotations

EJB Session Stateless

Cleverinstitut

- Stateless
  - 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

```
@Stateless
public class PersonService {
```

EJB Session Stateful

Cleverinstitut

- @Stateful
  - L'état de l'EJB est sauvegardé durant toute la session
  - Une instance de l'EJB est créée, cette instance reste disponible pour les futurs appels de ce client uniquement. Cette instance sera détruite a la fin de la session (timeout ou appel à une méthode portant l'annotation @Remove)

```
@Stateful
public class PersonService {
```

Injecter un Entity Manager


```java
@Stateless
public class PizzaService {

    @PersistenceContext(unitName="pizza-db") private EntityManager em;
```

Tâche planifiée


```
@Stateless
public class PizzaService {

    @Schedule(second="*/30", minute="*", hour="*")
    public void insererPizza() {
        ....
    }
```

Asynchronisme (1)


```
@Stateless
public class PizzaService {

    @Asynchronous
    public void insererPizza() {
        ....
    }
```

Asynchronisme (2)

Cleverinstitut


```
@Stateless
public class PizzaService {

    @Asynchronous
    public Future<List<Pizza>> findAllPizzas() {
        ...
        return new AsyncResult<List<Pizza>>(pizze);
    }
```

Transactions

ACID

Cleverinstitut

- Atomicité
  - Plusieurs opérations => toutes les opérations sont effectuées ou aucune.
- Cohérence
  - Après une transaction, la ressource doit se trouver dans un état cohérent
- Isolation
  - Pas de dépendance avec d'autres transactions qui peuvent avoir lieu en même temps
- Durabilité
  - Le résultat d'une transaction est durable (persisté).

Transactions

Cleverinstitut

- Les transactions EJB peuvent être gérées par le container ou programmatiquement dans l'EJB Session ou Message-Driven
- CMT
  - Transaction par défaut
  - Le container démarre et valide la transaction
- BMT
  - L'implémentation de l'EJB doit démarrer et valider une transaction fournie par le Container
  - Les transactions dans servlets sont gérées comme des BMT

Exemple de CMT

Cleverinstitut

```
@Stateless
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class Bean implements LocalBean{

    @TransactionAttribute(TransactionAttributeType.REQUIRED))
    public void myMethod() {
    }
}
```

Type d'attributs de transaction

Cleverinstitut

Attribut	Si une transaction existe	Si pas de transaction
REQUIRED	Utiliser la transaction existante	Démarrer une nouvelle transaction
REQUIRES_NEW	Suspendre la transaction en cours. Démarrer une nouvelle transaction.	Démarrer une nouvelle transaction.
MANDATORY	Utiliser la transaction existante	Lancer une exception
NEVER	Lancer une exception	Traiter sans transaction
NOT_SUPPORTED	Suspendre la transaction en cours.	Traiter sans transaction
SUPPORTS	Utiliser la transaction existante	Traiter sans transaction

Rollback Applicatif


```
@ApplicationException(rollback = true)
public class BusinessException extends Exception  {


}
```

Rollback Applicatif

Cleverinstitut

```
@Resource
private SessionContext sessionContext;

private void save() {

        // Du code
        // Du code
        // Du code

        sessionContext.setRollbackOnly();

}
```

Exemple de BMT

Cleverinstitut

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class Beanimplement implements LocalBean {
    @Resource private SessionContext sessionContext;

    public void myMethod(ReminderForm reminder) {
        UserTransaction utx = sessionContext.getUserTransaction();
        try {
            utx.begin();
           //implementation
            utx.commit();
        } catch(Exception e) {
            utx.setRollbackOnly();
        }
    }
}
```