Intégration d'applications

Java EE

Pascal Bleuyard

Pascal.Bleuyard@fr.michelin.com

28 septembre 2011 /ISIMA

Plan du cours

Remerciements

- Julien Ponge [1]
- Marie Agier [2]
- Philippe Lacomme [3]

[1] http ://julien.ponge.info
[2] http ://www.isima.fr/~ agier/
[3] http ://www.isima.fr/~ lacomme/

Références

- The Java EE 6 Tutorial[4] (anglais, 588 pages)
- Introduction à Java EE5 [5] (français,339 pages)

[4] http ://java.sun.com/javaee/6/docs/tutor ial/doc/
[5] http ://tahe.developpez.com/java/javaee/

Objectifs

A l'issue de ce cours vous apprendrez à :

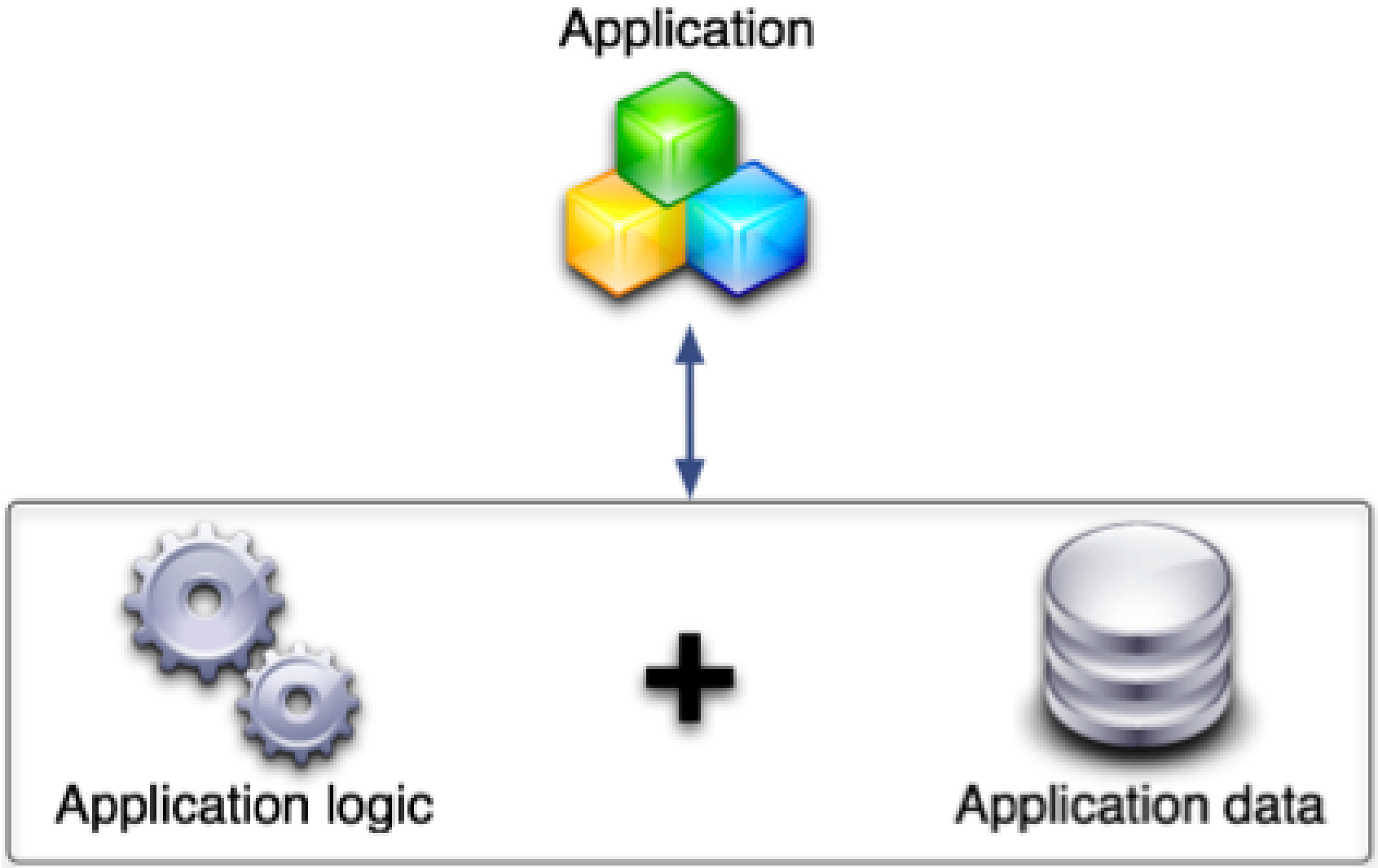
- concevoir une application Java EE(NetBeans)
  - vocabulaire
  - architecture distribuée
  - création
  - déploiement
- Faire les bons choix:
  - composants
  - *design patterns* (motifs de conception)

Plan du cours

Application d’entreprise

Exemples?

Application d’entreprise

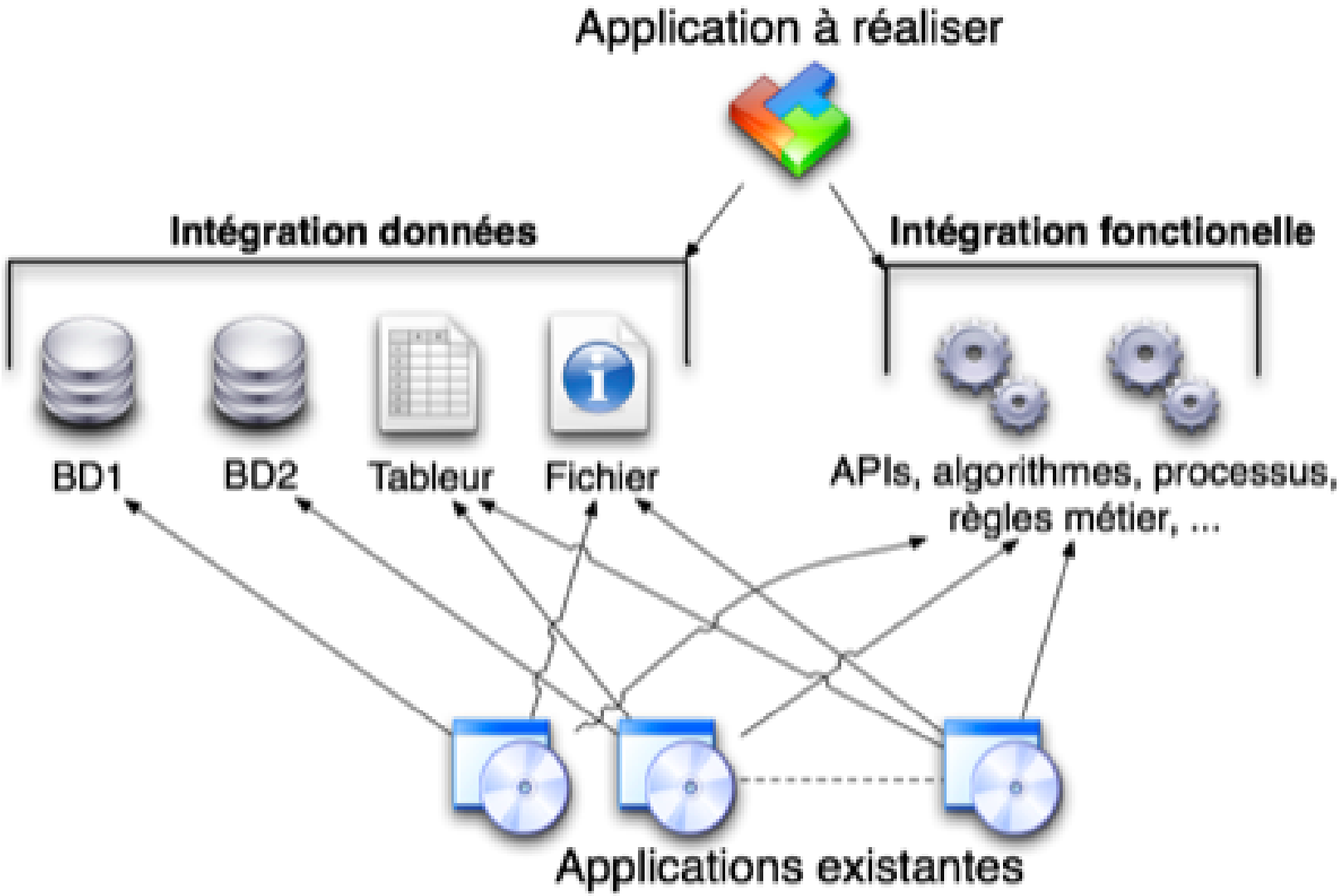


Des composants hétérogènes et dispersés

Systèmes d'exploitation

Langages de programmation
+
plateformes logicielles

Intégration



En résumé

Le système d'information de l'entreprise est constitué d'applications:

- hétérogènes
  - matériels
  - systèmes d'exploitation
  - langages
- réparties
  - propres
  - partenaires (clients, fournisseurs)
- protocoles de communication différents:
  - fichiers
  - appels de services (SOAP)
  - messages (Jabber/XMPP)
  - objets distants

Solution : middleware

On doit procéder à un *mapping* (mise en correspondance) des infor mations échangées!

Ce mapping est confié à un *middleware* (connecteur, intergiciel).

> *Un middleware est un logiciel qui per met de connecter des composants logiciels et des applications.*

Objectifs:

- communication et interopérabilité entre applications distribuées hétérogènes
- Complexité masquée par de nombreuses facilités (*forward engineering*)

Infrastructures: **développement** et **runtime**

Avantages

- Définition obligatoire d'une architecture (flux entre applications)
- Conception modulaire(découplage)
- *Frameworks* (cadres applicatifs) fiables et robustes
  - Spécifications figées dans le temps
  - composants réutilisables
- Maintenable (migration)
- Evolutif
- Extensible (encapsulation)

Inconvénients

- Coût initial élevé
- Compétences pluri-disciplinaires nécessaires, à forte valeur ajoutée:
  - Développement d'applications
  - Bases de données
  - Architecture des systèmes d'information
  - Systèmes distribués

Plan du cours

Introduction

Problématique

Architecture distribuées

Middleware: implémentations Java EE

Environnement d’exécution Java EE

Système d'information

Les systèmes d'information sont intrinsèquement **distribués** :

- postes de travail
- serveurs (applications, bases de données, fichiers)
- terminaux mobiles

Leur architecture a évolué au cours des années:

- matériel
- logiciel
- pratique

Enjeux

- Accès concurrents
- Gestion de connexion réseau
- Connexion aux bases de données
- Interface de gestion
- Équilibrage de charge
- Tolérance aux pannes
- Extensibilité et performance

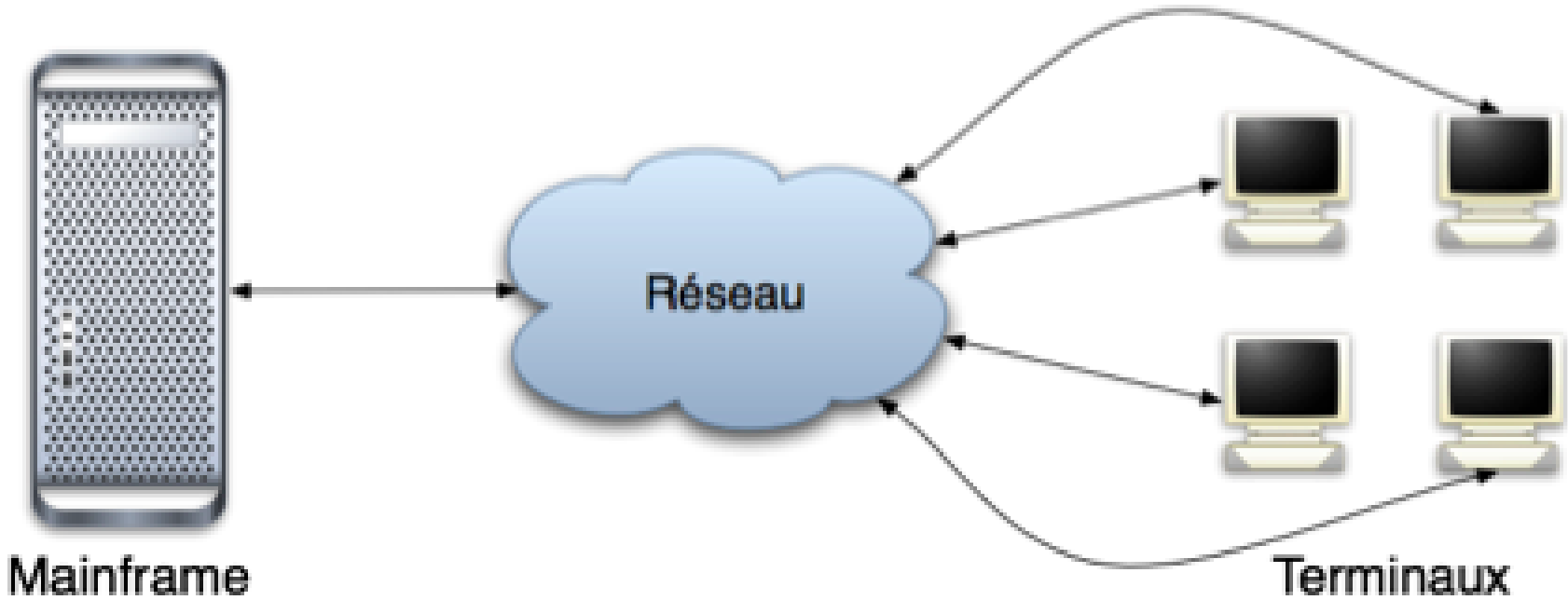
Niveaux logique d'une application

On sépare traditionnellement les fonctionnalités d'une application en *tiers* (étages, couches, niveaux logiques):

- présentation des données (affichage, dialogue avec l'utilisateur)
- Traitement métier des données(règles de gestion et logique applicative)
- accès aux données persistantes

1-tier : console passive(années 70–80)

- Applications hébergées sur un *mainframe* (unité central)
- Connexion par un terminal



1-tier : analyse

- Monolithique (centralisé)
  - déploiement relativement facile
  - point de défaillance unique
  - Montée en charge délicate
- Technologies propriétaires
- Logique applicative créée *ex nihilo* (de rien)
- Langage de niche (Cobol, For tran)
- Interfaces utilisateurs textuelles
- **Toujours en production!**

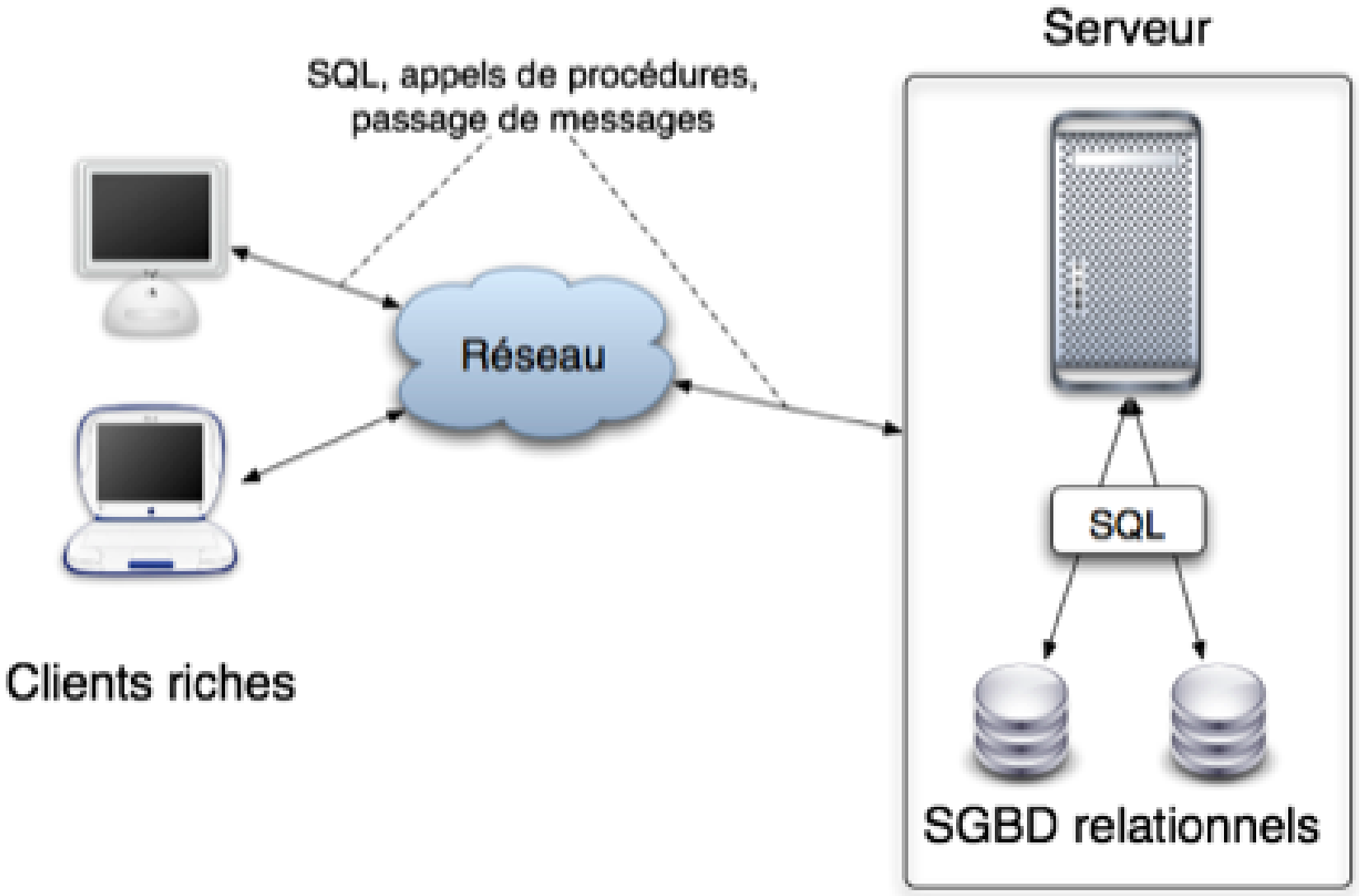
1-tier : intégration

Travail à fournir très coûteux:

- **schéma de données absent?**
- technologies de gestion des données (fichiers, SGBD) propriétaires ? obsolètes?
- logique applicative à décrypter?
- **langages arrivés à péremption?**
- technologies non-maintenues?
- extensibilité?

Solutions envisageables :extraction des données,émulation de terminal

2-tier : client-serveur (fin des années 80–90)



2-tier : analyse

1 tier *client*, 1 tier *serveur*. Les SGBD se généralisent avec cette architecture.

Le poste de travail exécute l'interface de l'application. Le traitement est déporté sur un serveur centralisé (cas général).

La communication se fait par envoi de messages et/ou appels de procédures à distance.

Il existe quelques variantes:

- Traitements légers côtés clients (logique applicative découpée)
- serveur uniquement pour les données
- mode *offline* (hors ligne)

2-tier : intégration

En général, le SGBDR est accessible. On emploie de plus souvent des couches d'abstraction(JDBC,ODBC).

Il est possible d'utiliser et mettre à jour les données depuis les bases existantes.

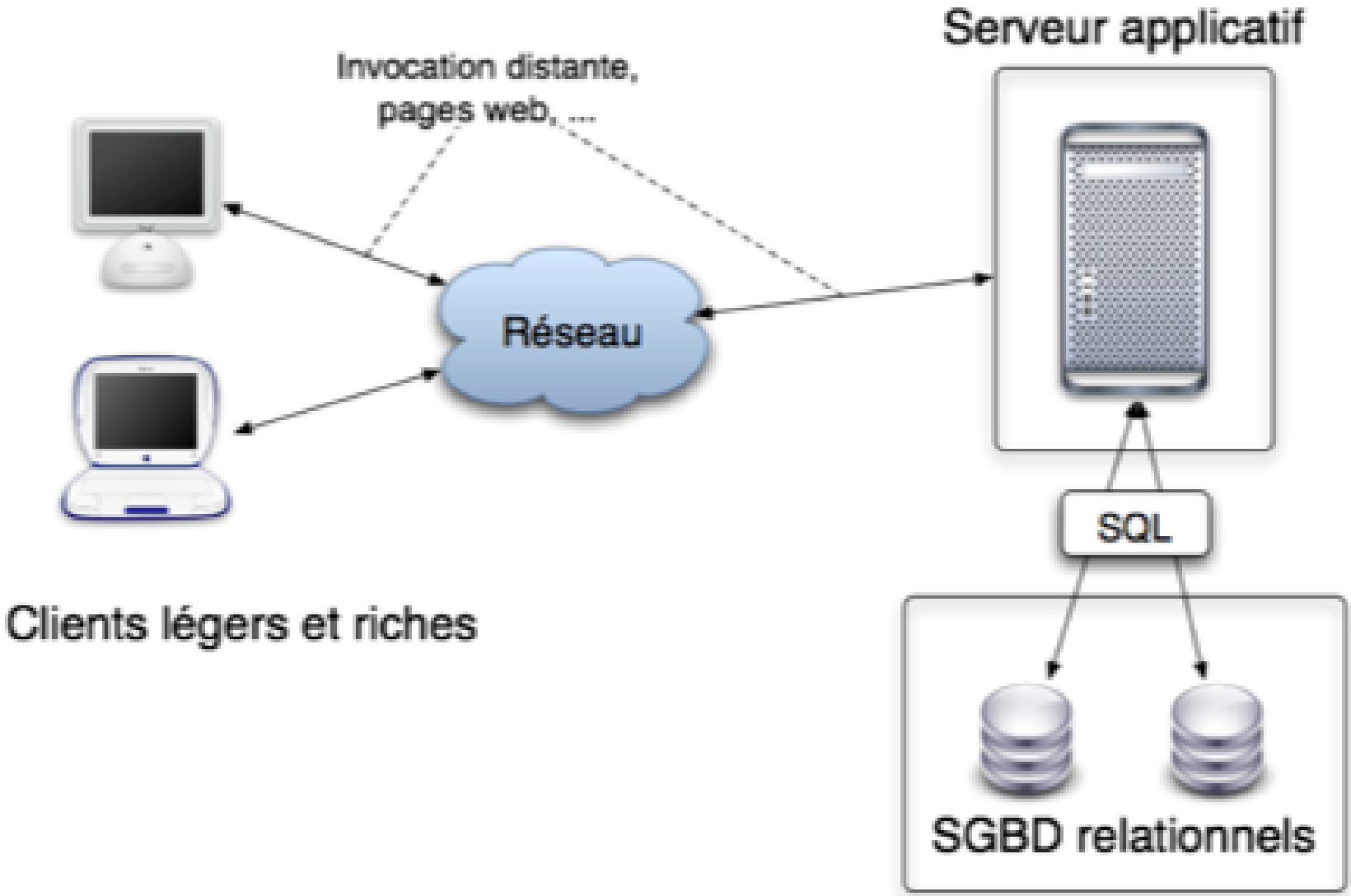
La logique applicative est accessible via une technologie hôte (Corba, RPC spécifique, messaging).

Il faut mettre en place des adaptateurs spécifiques pour connecter des systèmes hétérogènes sur les interfaces logiques.

Types de client

- Léger : traitement sur le serveur
- Lourd : traitement partiel par le client
- Riche : traitement *semi-fini* par le serveur, finalisé par le client

3-tier : application web (fin des années 90–2000)



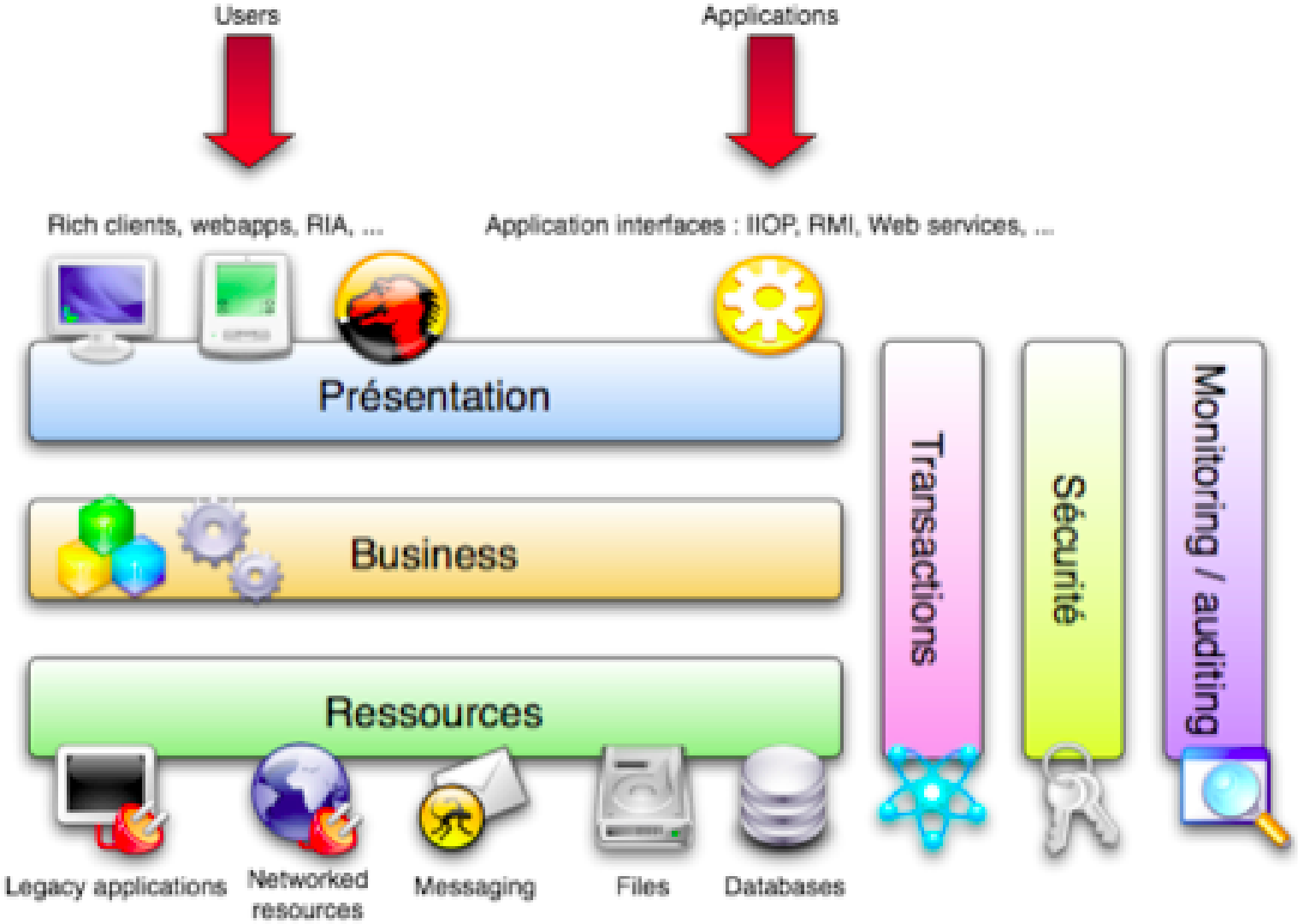
3-tier : analyse

1tier présentation, 1tier logique applicative,1 tier données.

Ces architectures se généralisent avec le web. On voit apparaître des *clients légers*(navigateurs web).

Serveurs applicatifs : JavaEE, .Net, LAMP

N-tier : serveur applicatif



N-tier : analyse

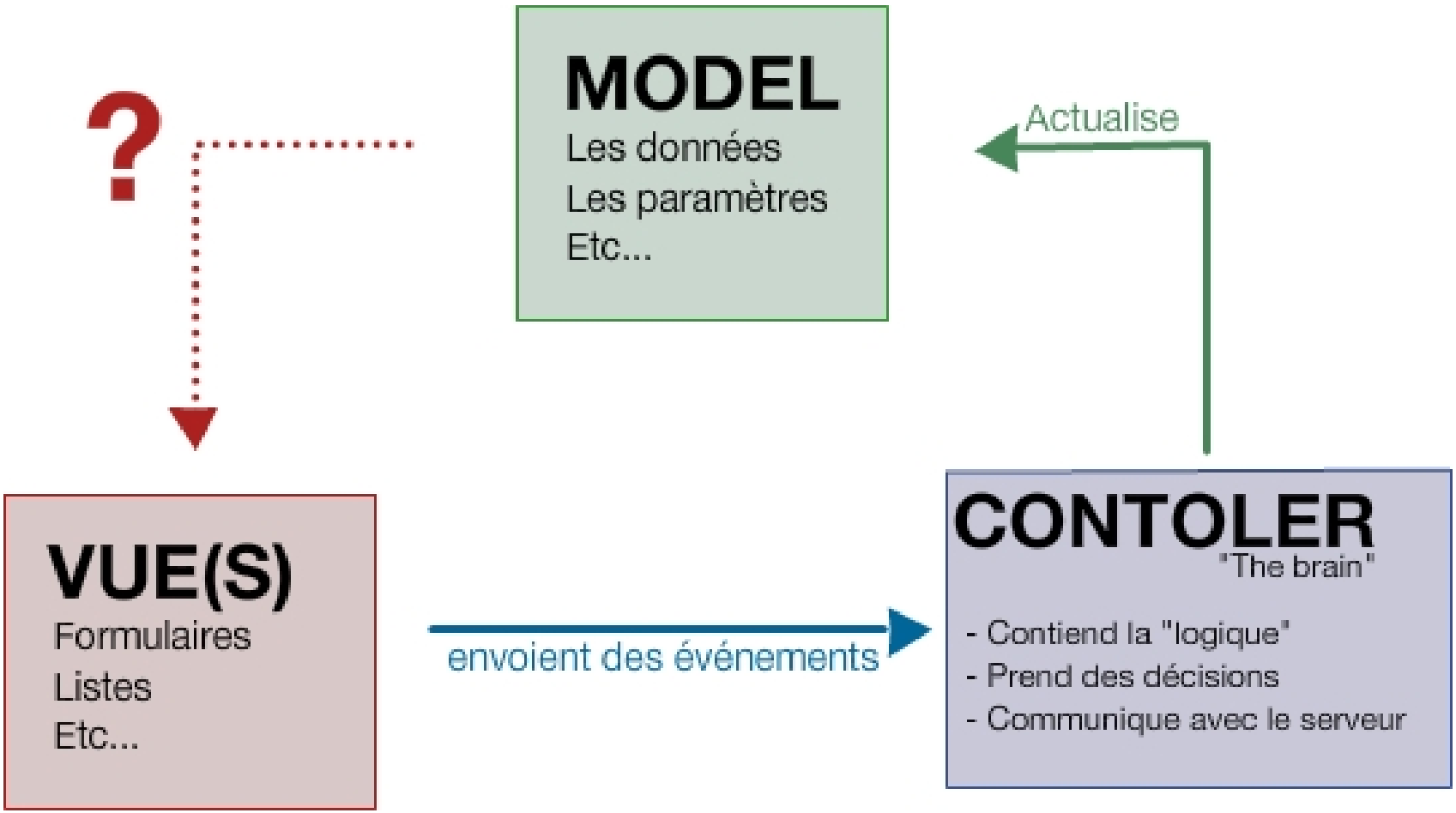
Niveau de granularité plus fin et couches transversales.
Exemples:

- 1 tier web, 1 tier application, 1 tier metier,1 tier données
- SGBDR en clusters et système de fichiers
- logique applicative distribuée avec communication par messager ie asynchrone
- clients web, mobiles et bureau

Découpage par niveaux

- Avantages
  - Facile à maintenir et faire évoluer
  - Modulaire
  - Concepts industriels (infrastructure)
- Inconvénients
  - Expérience nécessaire
  - Conception obligatoire avant développement (prototypage)
  - **Pas vraiment orienté objet**
    - Entités autonomes, ne contiennent pas de logique

Modèle-Vue-Contrôleur



Plan du cours

Introduction

Problématique

Architecture distribuées

Middleware: implémentations Java EE

Environnement d’exécution Java EE

Information

Visite du site de Ladoux à destination des écoles d'ingénieur courant octobre.

Rappels

- Applications d'entreprise naturellement distribuées
- Architecture multi-couche
  - présentation (client)
  - métier (web,application)
  - Accès aux données
  - transverses (transactions, sécurité)
- Middleware
  - communication et interopérabilité entre applications distribuées hétérogènes
  - Complexité masquée par de nombreuses facilités
  - conception (développement) et déploiement (runtime)

Middlewares

- Encapsulent de nombreuses fonctionnalités
- Offrent un cadre applicatif pour:
  - créer de nouvelles applications
  - recycler les anciennes (réduction des coûts)

Attention : ticket d'entré souvent élevé!

Questions

Comment permettre le recyclage/découplage des applications distribuées?

Quels problèmes cela pose-t-il?

Serveur d'applications

- Propose un framework pour la conception et le déploiement d'applications multicouches distribuées
- Héberge plusieurs applications
- Offre une large gamme d'API(BDD, **composants**, *etc.*)
- Spécifications :
  - Ouvertes : **Java EE** (Sun/Oracle, 40%)
  - Fermées : .NET (Microsoft, 7%)
- **N'est pas un serveur web!**

Question

Qu’est-ce qu’un serveur web?

Quelles sont les différences entre serveur web et ser veur d’applications?

Différences entre un serveur web et un serveur d’applications

Un serveur web ne traite que des requêtes HTTP (statique, pas de session, *etc.*).

Un serveur d’applications fournit des méthodes qu’un client peut appeler à travers différents protocoles.

Différences entre Java SE et JavaEE

La plateforme Java est à la fois:

- un **langage** de programmation orienté objet
- un environnement pour l'exécution d'applications

Plateformes Java

- Java Platform, Standard Edition (Java SE) : coeur
- Java Platform, Enter prise Edition (Java EE) : distribué
- Java Platform, Micro Edition (Java ME) : mobile
- JavaFX : script

Elles consistent toutes en une **machine virtuelle** (VM) et une **interface de programmation** (API).

Java EE

Plateforme fortement orientée serveur pour le développement et l'exécution:

- d'applications distribuées
- d'applications web

Composée de 2 par ties:

- Spécifications pour l'exécution de composants Java (serveur d'applications)
- API

Java EE : Avantages

- Architecture basée sur les composants
  - Découpage de l'application
  - séparation des rôles
- Interfaçage avec le système d'information existant
- Choix des outils
  - Développement :NetBeans, Eclipse
  - Serveurs d'applications, libres ou commerciaux
- Flexibilité dans l'architecture de l'application (couches) en combinant les composants par:
  - besoins
  - compétences

Java EE6

6ème évolution de la spécification Java Entreprise Edition (autrefois J2EE).

Spécifications pour la version 7 en cours : présentation le 13 octobre par Lava Jug

Plusieurs implémentations concurrentes:

- libres: **GlassFish**, JBoss, *etc.*
- propriétaires : IBM WebSphere, *etc.*

Implémentation minimale par Oracle/Sun : Java EE SDK

Compatibilité donnée par un TDK (*Test Compliance Kit*).

Evolutions pilotées par un processus ouvert : Java Community Process [6]

[6] http ://jcp.org

Java EE6

2 grandes familles d'outils:

- **composants**
  - métier
  - web
- **services**
  - infrastructure (JDBC, JNDI, *etc.*)
  - communication (JAAS,WS)

Java EE vise à rendre le développement de la **couche métier** plus facile, robuste et sécurisé.

Java EE 6 :Composants

- Métier
  - Composants spécifiques pour le traitement des données et l'interfaçage avec les bases de données: **JavaBean**, **EJB** (Entreprise JavaBean)
- Web
  - présentation (interface utilisateur et événements)
  - pages HTML uniquement
  - différentes technologies (code plus performant, robuste, facile à maintenir): **servlets**, **JSP**, **JSF**, **facelets**

API Java EE 6 par couche : Accès aux données

Couche contenant les sources de données (ERP, SGBD, mainframes, applications anciennes).

Généralement hébergée sur une machine différente du serveur Java EE.

Accessible par les composants de la couche métier.

API Java EE 6 par couche : Accès aux données

- **Java Database Connectivity** API (JDBC)
- **Java Persistence API** (JPA)
- Java EE Connector Architecture (JCA)
- Java Transaction API (JTA)

API Java EE 6 par couche : Application (Métier)

Fournit la logique métier d'une application (finance, commerce, *etc.*).

- Services de nommage et annuaires (JNDI)
- Composants **Enterprise JavaBeans** (EJB)
- Ser vices web (JAX-RS RESTful et JAW-WS)
- Entités **Java Persistence API** (JPA)

API Java EE 6 par couche : Web (Métier)

Couche gérant les intéractions entre les couches client et application.

Génère du contenu dynamique pour le client.

Gère les événements de l'interface utilisateur et renvoie les résultats appropriés de la couche application.

Contrôle le flux de pages chez le client.

Maintient l'état des données pour une session utilisateur.

Effectue des opérations basique et conserve des données temporaires.

API Java EE 6 par couche : Web (Métier)

- **Servlets** : objets (classes), traitent dynamiquement les requêtes et construisent les réponses (pages HTML)
- **JavaServer Pages** (JSP) : documents textes, compilés en servlets
- JavaServer Faces (JSF) : formulaires (validation, *etc.*)
- Facelets: applications JSF, utilisent XHTML au lieu de JSP
- Expression Language (EL) : *tags*(étiquettes) standards, Utilisés par les JSP et facelets, référencent les composants Java EE
- Java Server Pages Standard Tag Library : tags, encapsulent les fonctionnalités communes des JSP
- Composants JavaBeans : objets, conservent temporairement les donées des pages d'une application

API Java EE 6 par couche : Client (Présentation)

Couche composée des applications client accèdant à un serveur Java EE.

Généralement hébergée et exécutée sur une machine différente du serveur Java EE.

Les clients envoient des requêtes au serveur Java EE. Le serveur traite les requêtes et renvoie une réponse au client.

Pas nécessairement une application Java!

- navigateur web
- application autonomes
- autres serveurs

API Java EE 6 par services

- Nommage et annuaires: JDNI
- SGBD : JDBC,JPA
- Composants métiers: EJB
- Transactions :JTA
- Messages : JMS
- Web services/XML : JAXB, JAX-WS, JAX-RS
- Web : servlets, JSP, JSF, facelets
- Authentification/sécurité : JAAS
- Instrumentation : JMX

Plan du cours

Introduction

Problématique

Architecture distr ibuées

Middleware: implémentations Java EE

Environnement d’exécution Java EE

Serveurs Java EE

Implémentent les API de la plateforme Java EE et fournissent les services Java EE.

Appelés serveurs d'applications car ils permettent de distribuer les données aux clients de la même manière que les serveurs web distribuent les pages aux navigateurs.

Les serveurs Java EE hébergent plusieurs types de composants qui correspondent aux couches d'une application.

Les clients n'accèdent pas directement aux composants!

Le serveur Java EE fournit les services de ces composants sous forme de **conteneurs**.

Conteneurs Java EE

Un serveur Java EE héberge différentes ressources:

- fournies par les applications (composants avec logique applicative, web, *etc.*)
- administrées par le serveur (connections SGBDR, politiques de sécurité, *etc.*)

Les conteneurs permettent de gérer ces ressources:

- Cycle de vie
- Service transverses
- Accès à d’autres ressources

Conteneurs Java EE

Permettent de découpler les applications de leur configuration.

Interface entre les composants et les fonctionnalités bas niveau offertes par la plateforme.

Fonctionnalités différentes pour chaque type de composant.

Le serveur permet aux composants de travailler ensemble dans une application d'entreprise.

Exemple : accès à une BDD **sans connaître les détails de connection.**

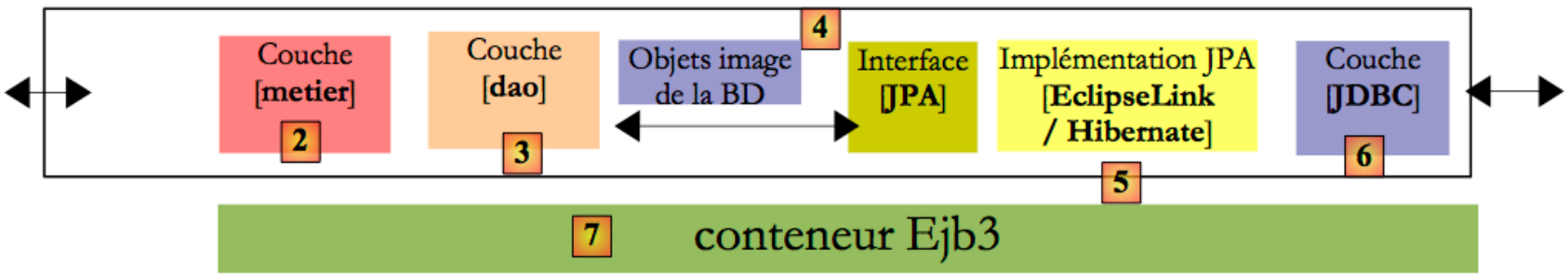
Conteneurs Java EE

Conteneur web : interface entre composant (servlet, JSP ou JSF) et les composants du serveur web, gère le cycle de vie du composant, distribue les requêtes et fournit les informations de **contexte**

Conteneur d'applications client : interface entre l'application client et le serveur Java EE, s'exécute sur la machine client

Conteneur EJB : interface entre les EJB et le serveur Java EE, fonctionne sur le serveur et gère l'exécution des beans

Infrastructure Java EE : En résumé



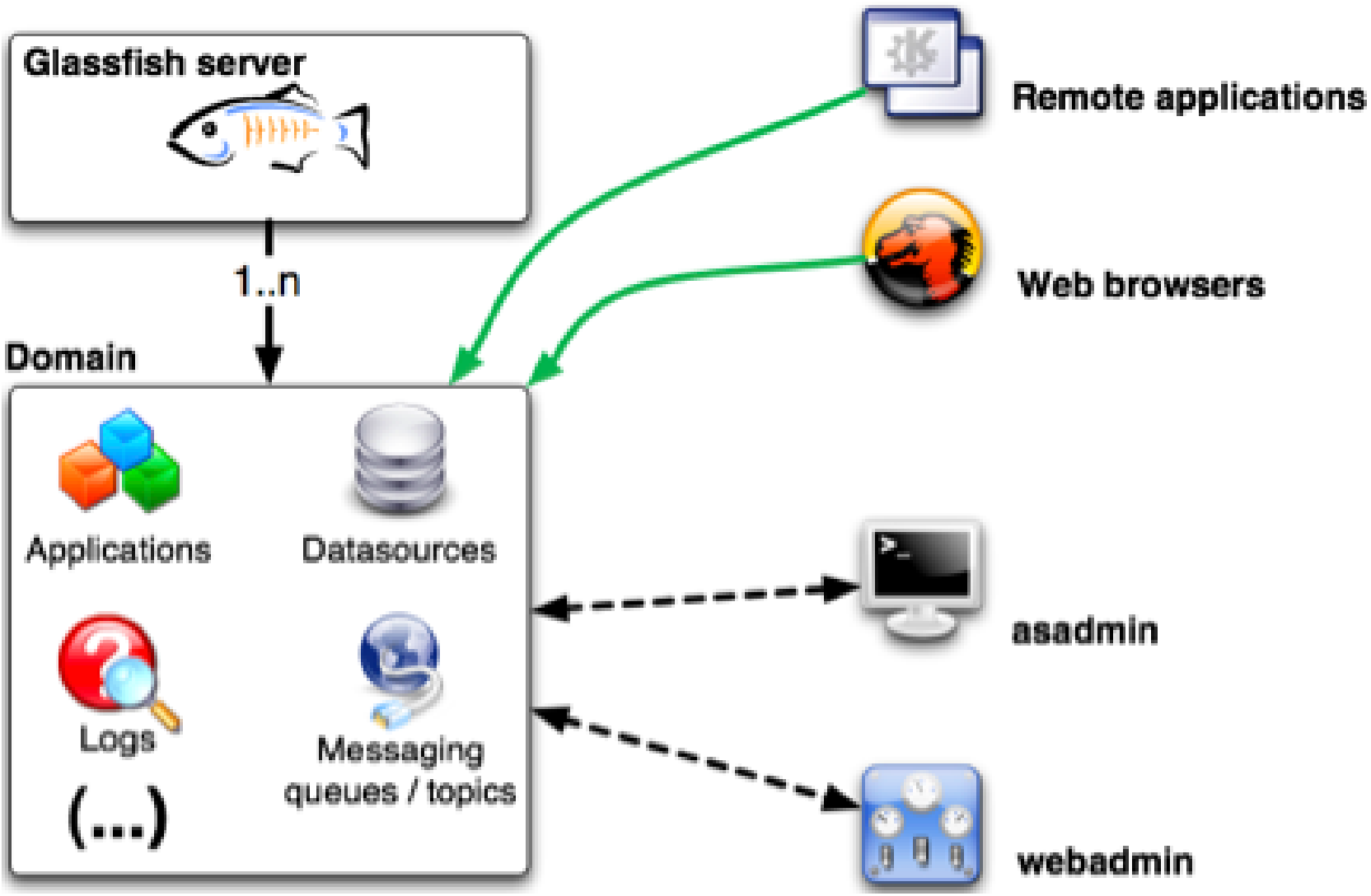
GlassFish

Implémentation Java EE 6libre [7] de Oracle/Sun.

Version actuelle :3.1.1

[7] http ://glassfish.java.net/

GlassFish


Glassfish server
1..n
Domain
Applications
Datasources
Logs
(...)
Messaging
queues / topics
Remote applications
Web browsers
asadmin
webadmin

Déploiement d'une application dans un conteneur

Il faut fournir :

- l'application avec tous les composants (classes compilées, ressources, *etc.*) regroupée dans une archive/module (format propre à chaque conteneur)
- des **descripteurs** (XML) de déploiement (configuration pour certains modules)

Assemblage et déploiement d'une application Java EE

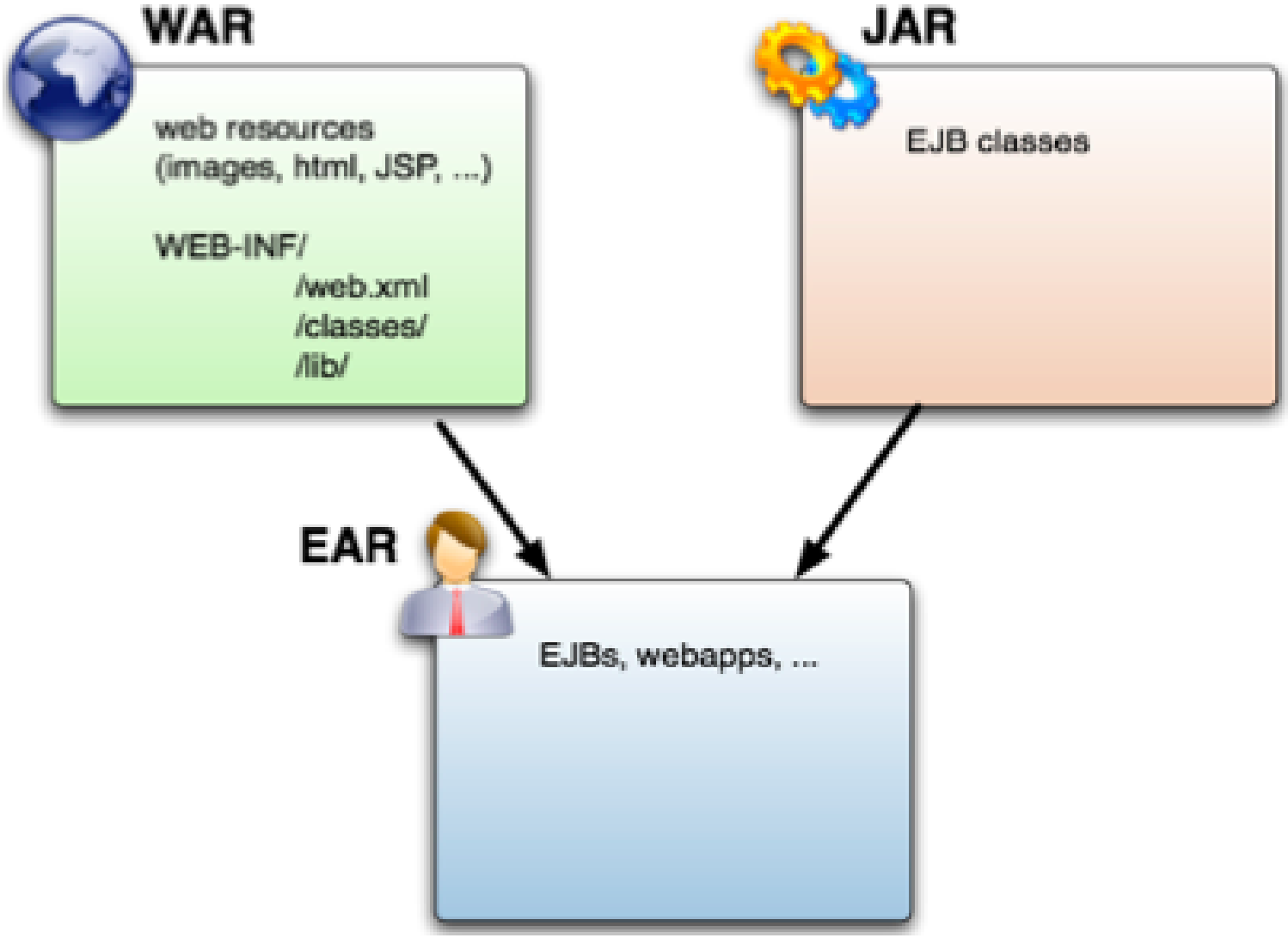
Plusieurs types d'archives (PkZIP):

- web : WAR (ressources, servlets, JSP)
- Application client : JAR (ressources, classes, bibliothèques, images)
- EJB : JAR(EJB, classes)
- *Ressource Adapters*: RAR
- *Enter prise Applications* : EAR (modules)

Simple : commande, upload par une interface web ou simple dépôt dans un répertoire (domaine).

Serveurs d'application extraient chaque module du fichier EAR et les déploient séparément un par un.

Déploiement Java EE : En résumé



Simplification des développements

Java EE réputé lourd et complexe à cause des nombreuses entités à développer (classes, interfaces, fichiers de configuration, *etc.*).

Utilisation intensive des **annotations** pour:

- simplifier les développements
- réduire le temps nécessaire à leur réalisation

Introduites dans Java EE 5 (autrefois que pour Javadoc).

Méta-données exploitées par le conteneur. Remplace entités à définir ou règles à respecter.

Simplification des développements : Annotations

Java EE propose des annotations pour de nombreux rôles:

- définition et utilisation des services web
- définition des EJB
- mapping objets/XML
- **mapping objet/relationnel**
- **injection de dépendances**
- précision sur les infor mations de déploiement

Utilisation pas obligatoire : configuration par descripteur de déploiement.

Simplification des développements : Annotations

- Commence par le caractère @ suivi par le nom de l'annotation
- Éventuellement suivi d'une liste de paramètres
- possède une valeur par défaut
- précède par convention les entités qu'elle caractérise
- correspondent à une classe particulière
- influence sur la façon dont certains outils vont exécuter le code (mais pas les traitements)