

# Java Enterprise Edition

## Chapitre 1: Introduction et motivations

Sébastien Chèvre, 2020

# Who am i?

## Chèvre Sébastien



seb.chevre@gmail.com



@sebChevre



sebchevre



sebChevre

- ***Softcom S.A.***
- ***Digital Project Manager/Software Architect***
- ***Expert CFC JU, Travaux de Bachelor***
- ***Orateur conférences IT***

# Organisation du module JEE



# Objectifs pédagogiques du module

## Java Entreprise Edition - JEE

- ❑ Décrire l'architecture de base du framework **JavaEE /Spring**. **(M)**
- ❑ Expliquer le rôle de chaque couche **verticale du framework** (JVM, Serveurs d'application, Framework). **(M, A)**
- ❑ Expliquer le rôle de chaque couche **horizontale du framework** (Présentation, Métier, Données). **(M)**
- ❑ **Développer une application Web sécurisée et testée** en se basant sur *Spring boot, Spring MVC, Spring Data / JPA, Spring Testing* et *Spring Security* **(A)**
- ❑ Développer une application simple mettant en œuvre **une communication asynchrone** en utilisant la technologie **JMS** (*Java Messaging Services*) et/ou *Spring Integration* **(A)**
- ❑ Développer une application simple mettant en œuvre une architecture micro-services **(A)**
- ❑ **Concevoir et développer** (sous forme de projet) une application métier de complexité moyenne en utilisant les Framework *Spring boot, Spring MVC, Spring Security, Spring Data* et une base de donnée *mySQL* **(R)**

# Organisation du module

## Java Entreprise Edition - JEE

### ■ Module orienté pratique et théorie

- ☐ Présentations théoriques suivies de mise en pratique
- ☐ Nécessité d'approfondissement personnel
- ☐ Projet et rendus notés
- ☐ Max 30-60' de travail / semaine
- ☐ 10 min de questions/réponses cours précédent

### ■ Évaluation

- ☐ Projet Pratique
- ☐ 2 contrôles écrits :
  - **Semestre d'automne → courant décembre 2019**
  - **Semestre de printemps → courant avril 2020**

# Organisation du module

## Suivi des travaux

### ■ Aspects et exercices pratiques sur GitHub Wiki

- <https://github.com/sebChevre/cours-JEE-2020-2021/wiki>

### ■ Suivi des travaux sur GitHub project

- <https://github.com/sebChevre/cours-JEE-2020-2021/projects/1>

### ■ Fichiers divers sur le repository

- Répertoire **/beershop**  
*contient des ressources diverses sur le projet que nous allons faire évoluer*
- Répertoire **/util**  
*Utilitaires divers (classes Java, exemples, etc...)*

## Plan du module – chapitres Java Entreprise Edition - JEE

- **Chapitre 01 : Introduction et motivations**
- Chapitre 02 : Machine Virtuelle Java
- Chapitre 03 : Java EE
- Chapitre 04 : *Spring Boot & Spring MVC*
- Chapitre 05 : *ORM & DAO avec Spring Data & JPA*
- Chapitre 06 : Sécurité avec *Spring Security*
- Chapitre 07 : Tests avec *Spring Testing*
- Chapitre 08 : SOA & Micro-services avec *Spring*
- Chapitre 09 : *JMS (Java Messaging Service)*
- **Projet**

# Objectifs du chapitre

## Chapitre 01 : Introduction et motivations

A la fin de ce chapitre, les étudiants peuvent:

- Expliquer les objectifs du cours **Java Enterprise Edition**
- Comprendre les notions de bases en matière **d'architecture logicielle**
- Comprendre les notions de **SOA/WOA**
- Comprendre les implémentations **SOAP/REST** et leurs différences
- Comprendre la notion de **service**
- Définir les différents termes et notions relatifs à **JavaEE** et **Spring**
- Expliquer dans quels contexte les technologie **JavaEE** et **Spring** sont pertinentes



# Plan du chapitre 1

## Chapitre 01 : Introduction et motivations

- **Introduction et motivation**
  - Architectures d'entreprise
  - Architectures Distribuées, SOA, WOA
  - Technologies Web
  - La technologie **Java EE** et **Spring**

# JEE

## Sémantique des diapositives



À savoir théoriquement (TE)

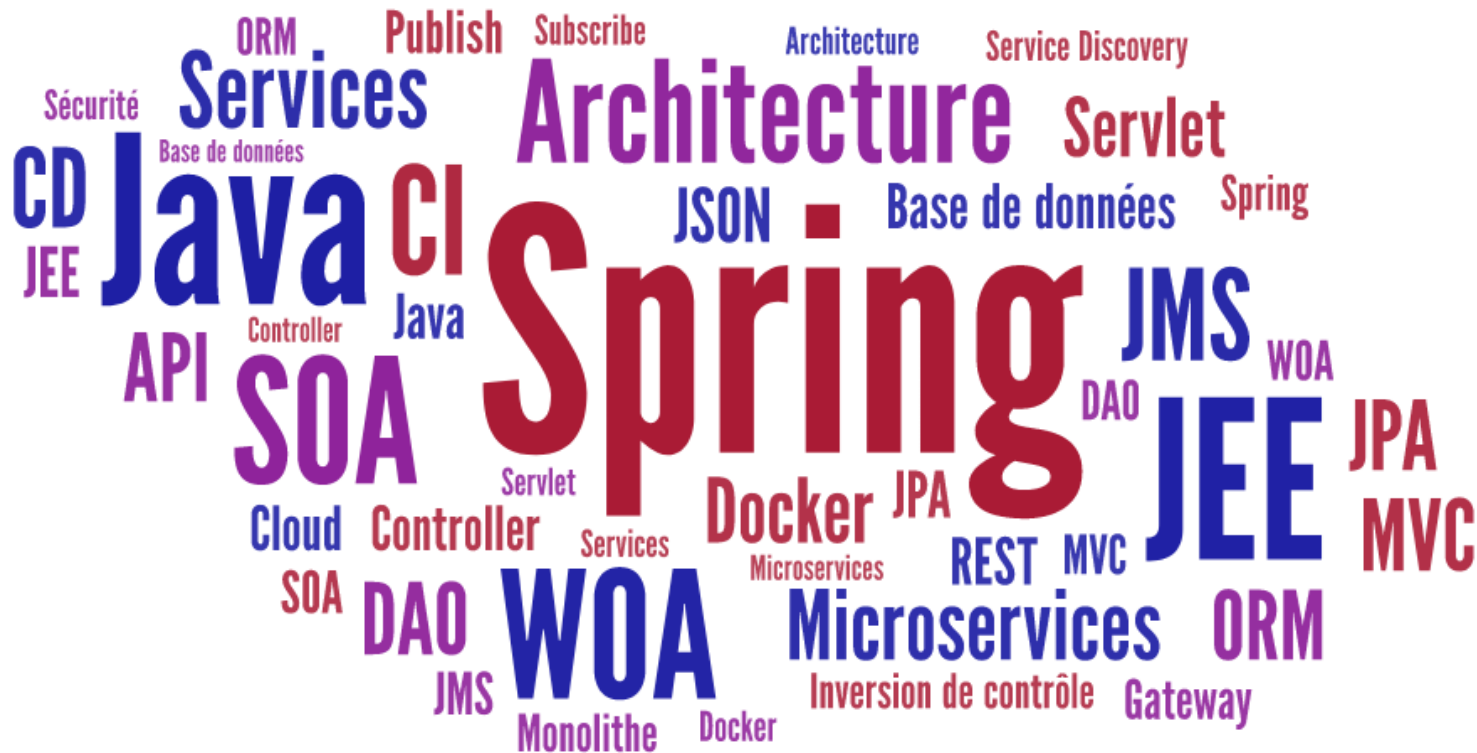


Sensibilisation, illustration de concepts, exemples

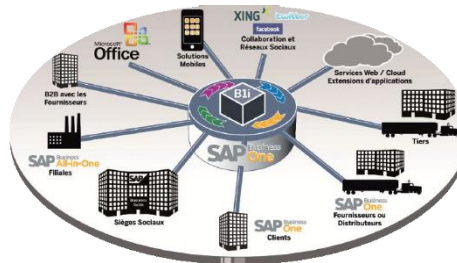


Aspects pratiques

## Réflexion commune et vos attentes



# Applications d'entreprise et architecture



# Applications d'entreprises

## Besoins propres



### ■ Besoins de normalisation

- Intégrables, communicantes / distribuées, adaptables, maintenables, portables

### ■ Besoins d'abstraction

- Portables, maintenables, extensibles, intégrables / distribuées, adaptables

### ■ Besoins de communication

- Intégrables, sécurisée, distribuées

### ■ Besoins de composants

- Maintenables, sûres, extensibles, adaptables, portables, disponibles / distribuées

**Et surtout : Comment réduire les temps et les coûts de développement et d'évolutions d'une application ?**

# Applications d'entreprises

## Cas réel



### ■ Cas réel: application de gestion

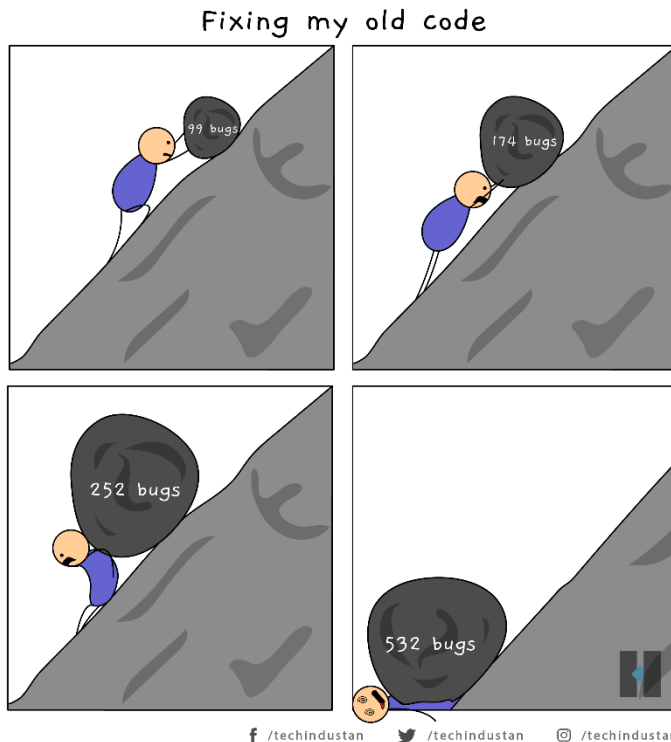
- ☐ 25 ans de vie
- ☐ Basée sur un framework «maison», sous perfusion
- ☐ Monolithe
- ☐ Intégration avec systèmes externes (ESB, WebServices, etc...)
- ☐ 2 – 4 versions /année
- ☐ > 3M de ligne de code
- ☐ > 1M CHF de paiement de rentes/mois
- ☐ Technologies et pratiques depuis 25 ans!
- ☐ Basé sur JEE
- ☐ Maintenu par des «nouveaux développeurs»
- ☐ Représentation mentale impossible



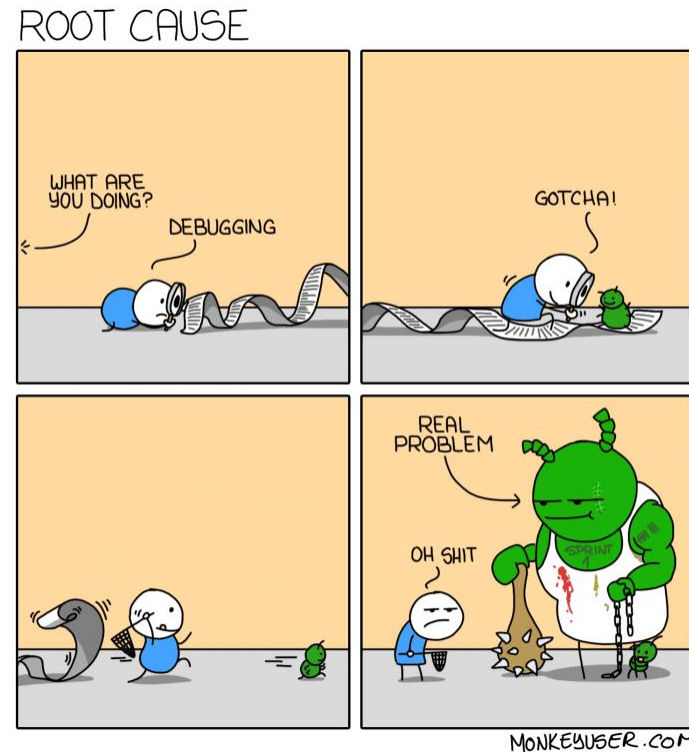
# Applications d'entreprises Bugs



## ■ Pour éviter cela



## ... ou celà



# Architecture Logicielle

## Définition



- Une **architecture logicielle** consiste à définir pour un système informatique **les éléments qui le composent, les liens qui les relient, les interfaces de ces éléments et l'organisation qui les gère.**
- Une architecture logicielle d'un programme ou d'un système d'information **est la structure ou les structures de ce système incluant les composants logiciels, leurs interfaces visibles et les relations qui les relient**



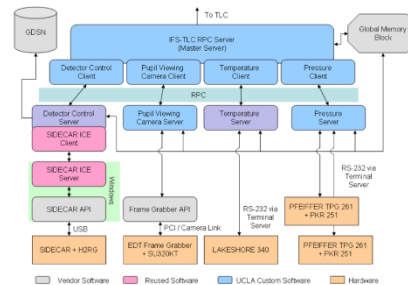
# Architecture Logicielle



- Les pratiques de développement et d'architecture d'un logiciel ne peuvent se baser que sur une seule constante:

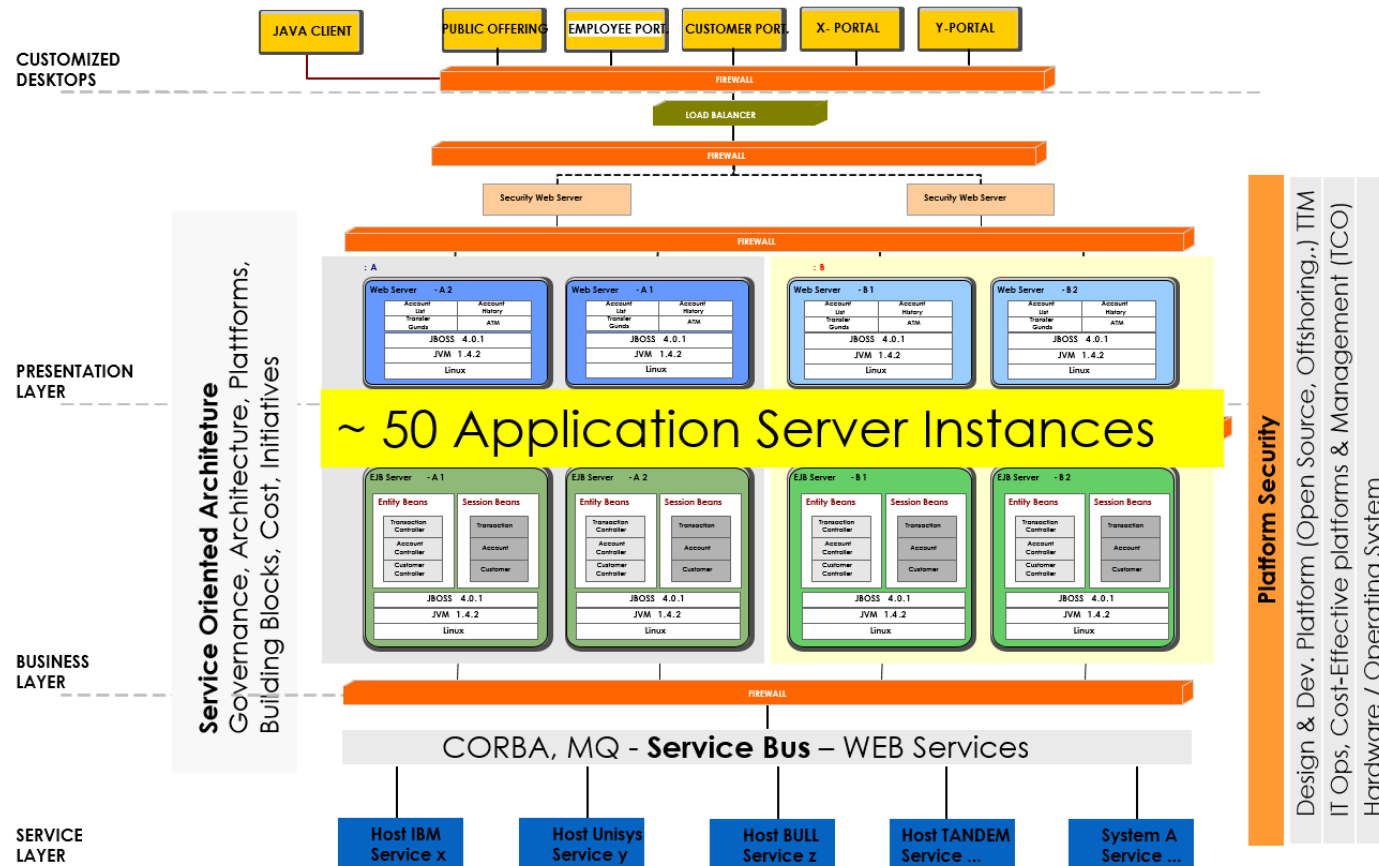
## ***Le changement !***

- Les pratiques d'architecture consistent donc à rendre une application *maintenable*, à des **coûts raisonnable**, et dans le respect **des besoins fonctionnels**



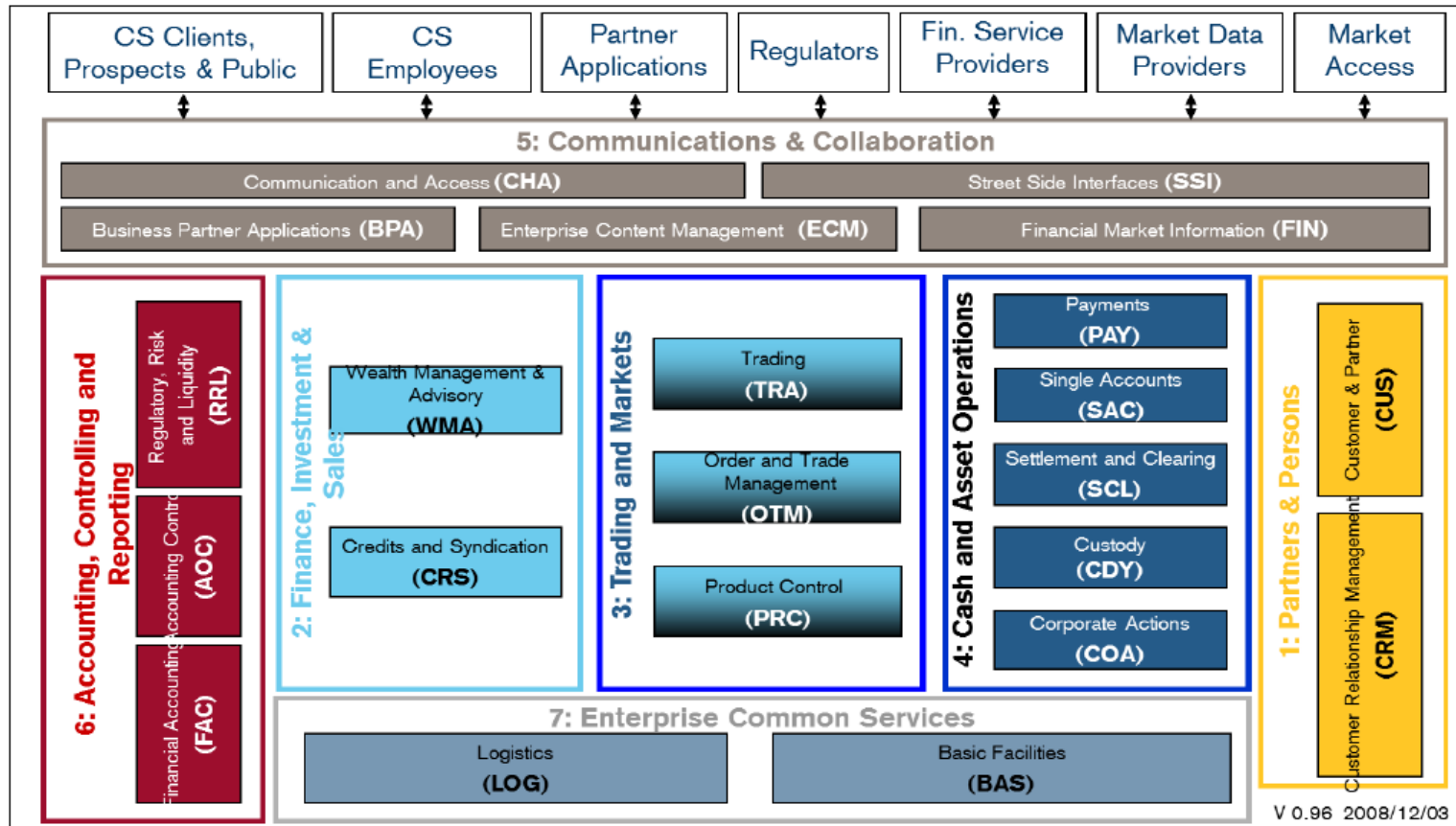
# Exemple d'architecture

## Vue système globale



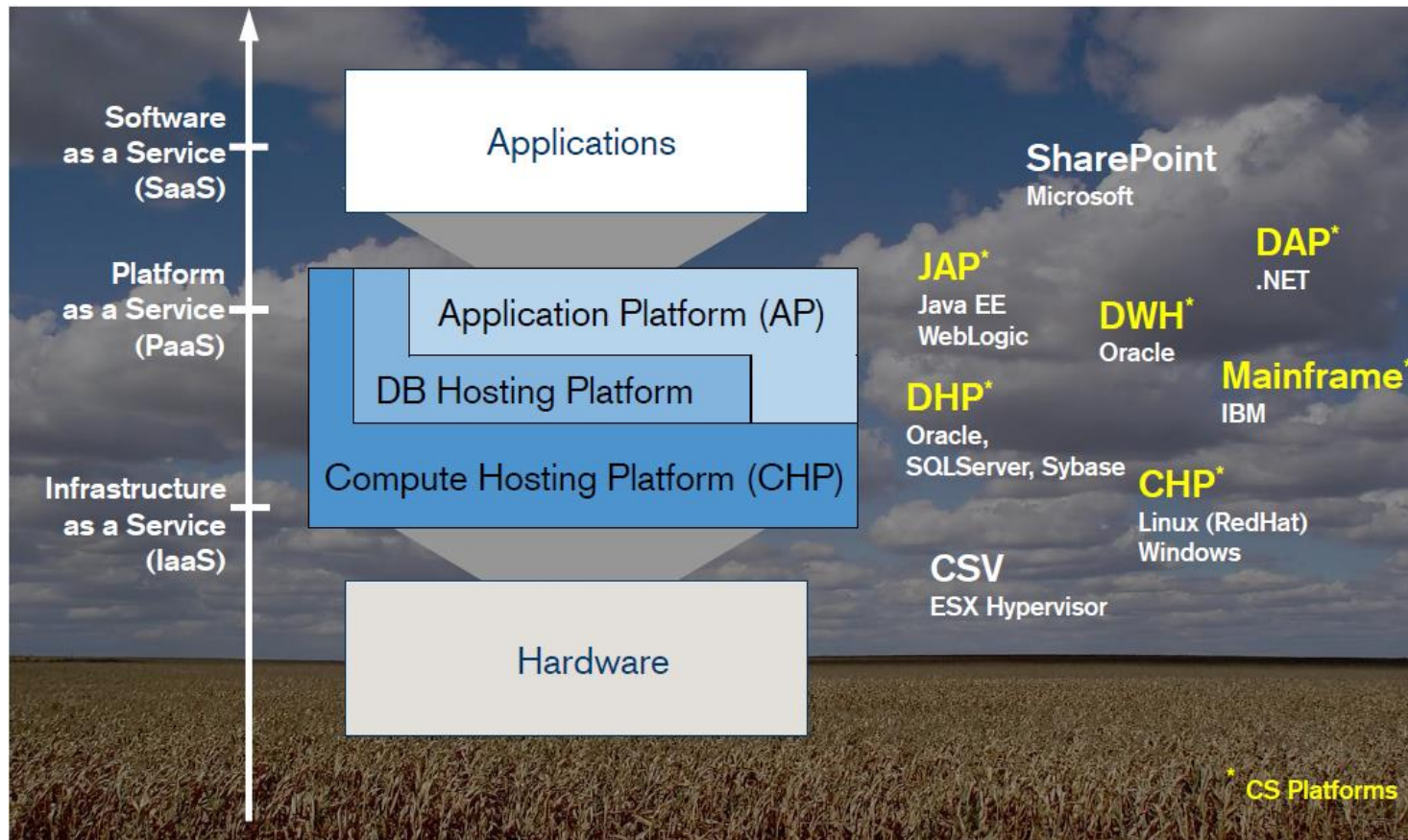
# Exemple d'architecture

## Architecture d'entreprise - vue métier



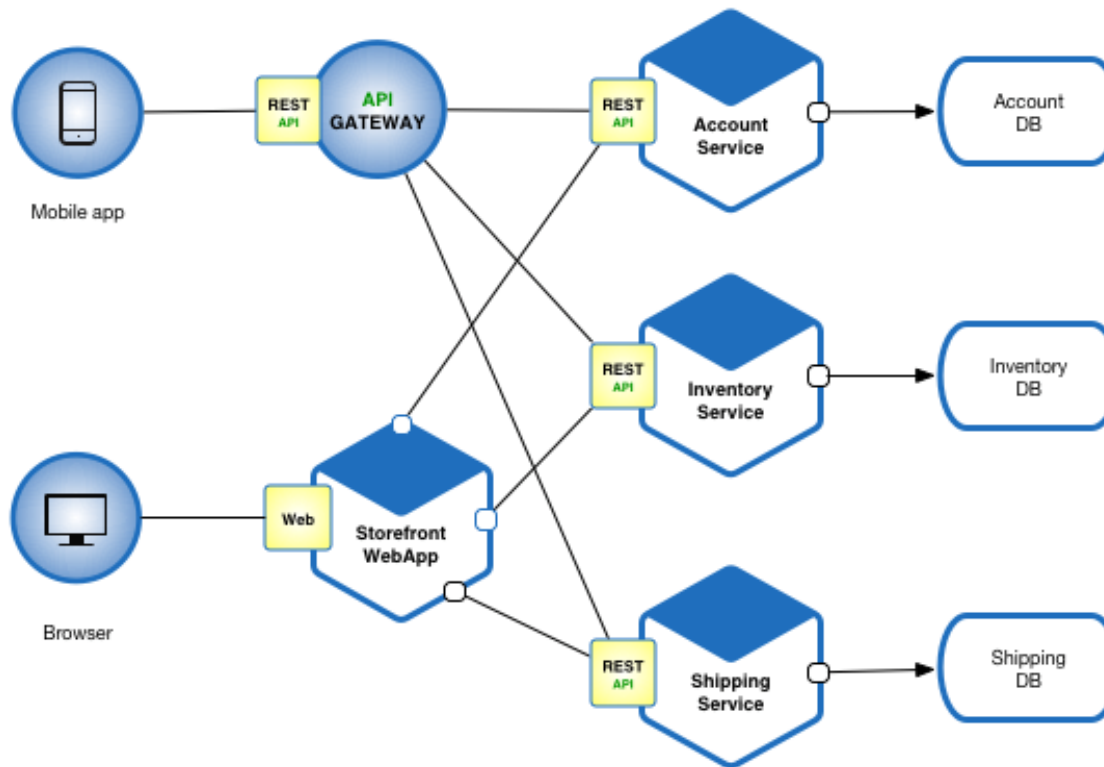
# Exemple d'architecture

## Architecture d'entreprise – vue plateforme



# Exemple d'architecture

## Architecture applicative - microservices

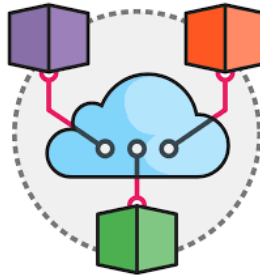


## Exemple d'architecture Exercice



- En groupe de 4-5 personnes, esquissez, en 10 minutes, l'architecture d'un système e-banking

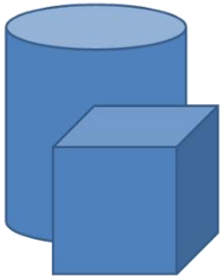
# Architectures réparties/distribuées





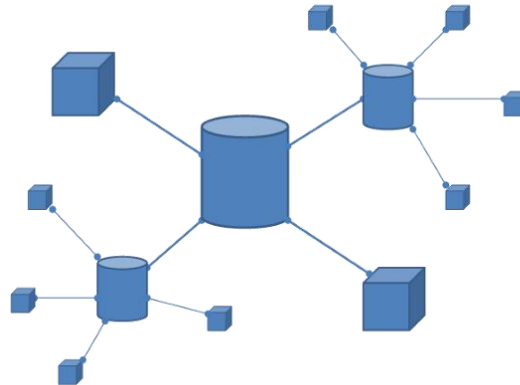
# Architecture

## Organisation globale



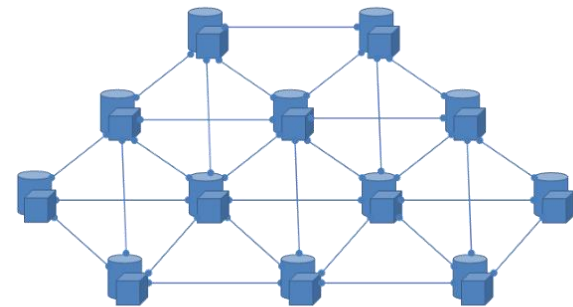
### Centralisé

un nœud s'occupe de tout



### Distribuée

la charge est répartie aux  
sous nœuds

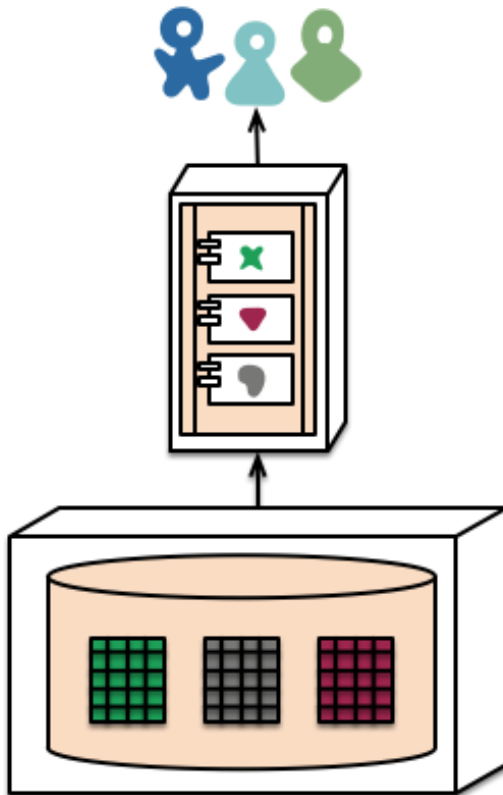


### Décentralisée

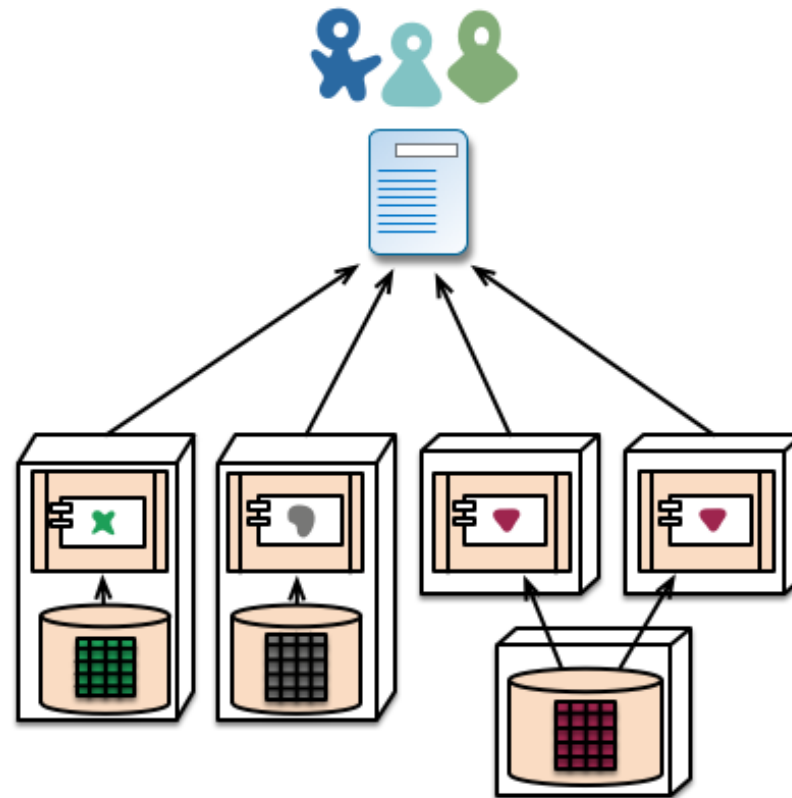
les nœuds sont connectés  
entre eux

# Architecture

## Monolithe vs distribuée



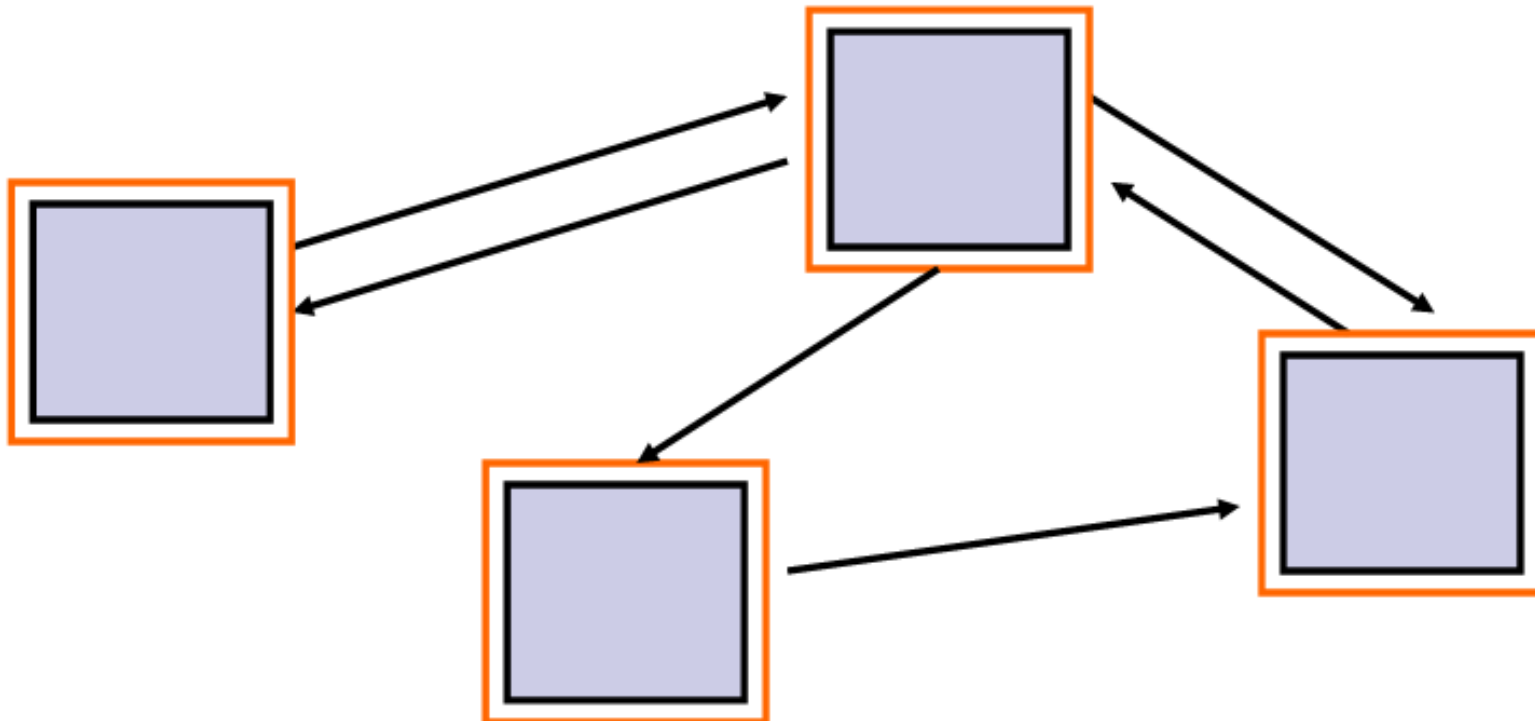
**Monolithe**



**Distribuée**

# Architecture

## Architecture basé objet (RMI, CORBA, RPC,...)

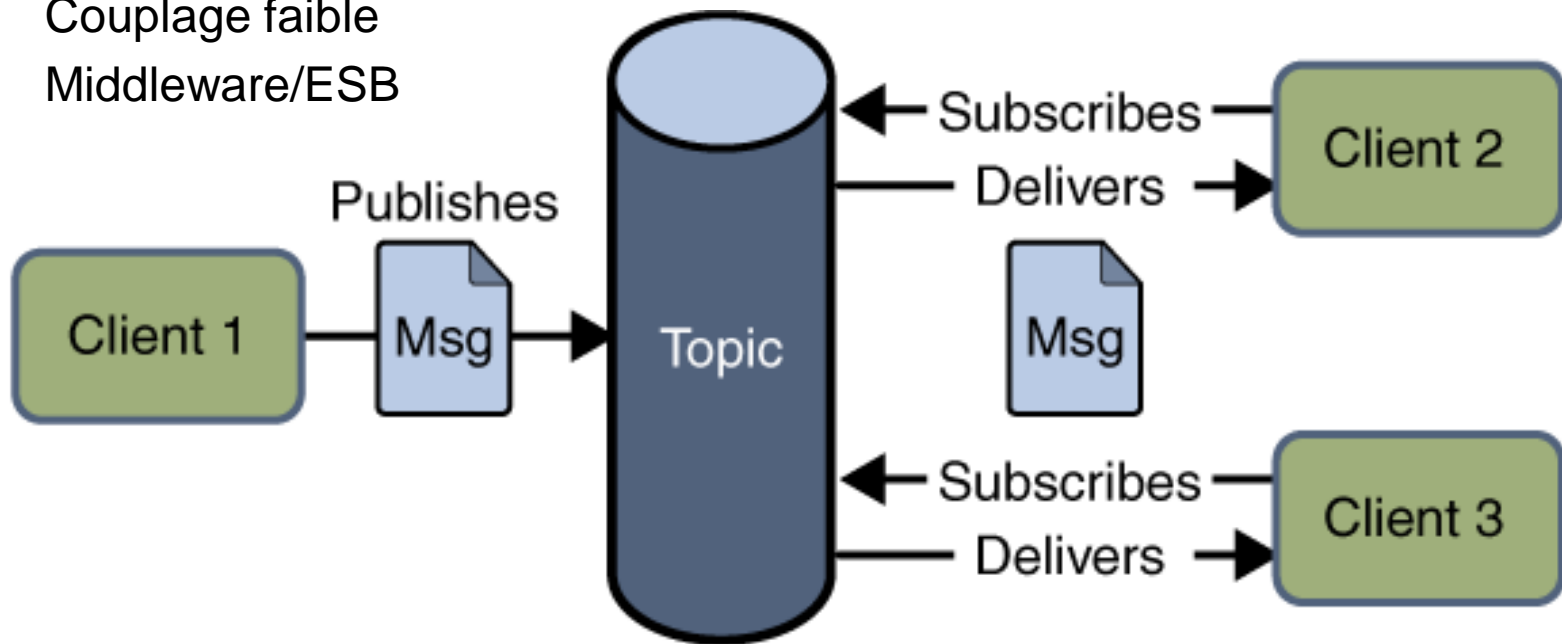


# Architecture

## Architecture évènementielle

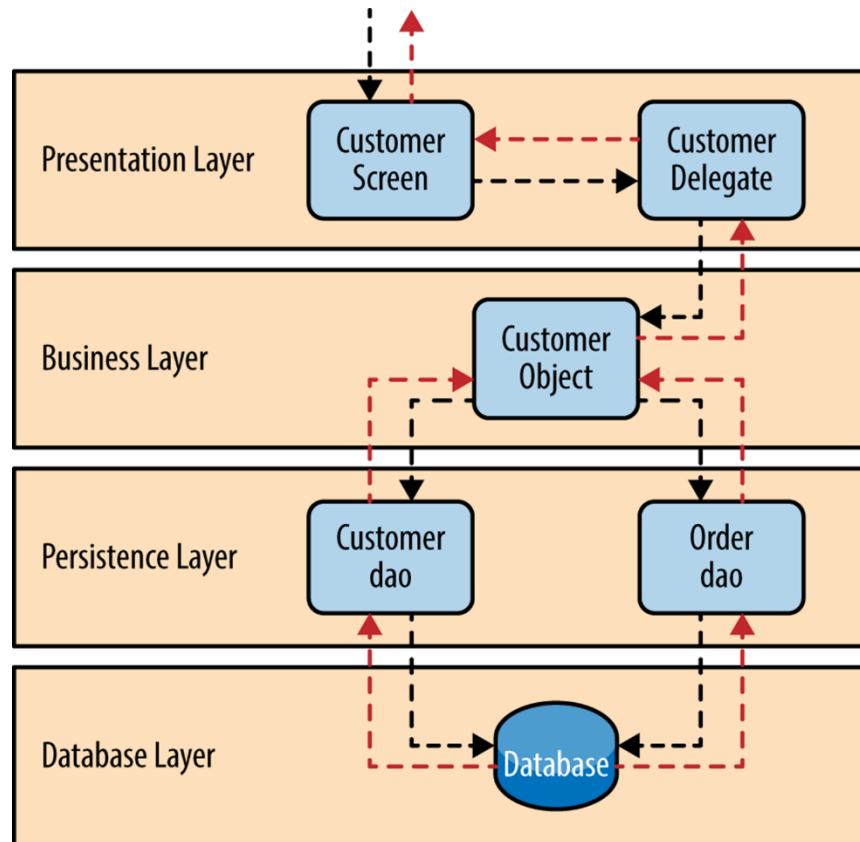


- publish/subscribe
- Asynchrone
- Couplage faible
- Middleware/ESB



# Architecture

## Architecture logicielle en couche



# Applications Réparties/distribuées

## Niveaux de distribution



- **Distribution des données**
  - Données distribuées, traitement centralisé
- **Distribution du contrôle**
  - Données centralisées, contrôle distribué
- **Distribution des utilisateurs**
  - Données et contrôle centralisé, utilisateurs distribués

**Le Web : Une combinaison de tout ça...**

**On parle d'architecture WOA (*Web Oriented Architecture*)**

# Applications Réparties/distribuées

## Besoins, pourquoi la distribution?



### ■ Besoins propres des applications

- Intégration d'applications existantes initialement séparées (legacy)
- Intégration massive de ressources
  - Grilles de calcul, gestion de données
- Pénétration de l'informatique dans des domaines nouveaux d'application
  - Intégration d'objets du monde réel
  - informatique omniprésente (ubiquitous computing)
- Surveillance et commande d'installations

### ■ Possibilités techniques

- Coût et performances des machines et des communications
- Interconnexion généralisée
  - Exemple 1 : interpénétration informatique-télécom-télévision
  - Exemple 2 : Réseaux de capteurs

# Service Oriented Architecture SOA





# SOA

## Définition



L'**architecture orientée services** (SOA) est une forme d'architecture qui est un modèle d'interaction applicative mettant en œuvre des **services** (composants logiciels), caractérisé par:

- **une forte cohérence interne** (par l'utilisation d'un format d'échange pivot, le plus souvent *XML* ou *JSON*)
- **des couplages externes «lâches»** (par l'utilisation d'une couche d'interface interopérable, le plus souvent un service web)

## SOA Service



Un **service** est ***une fonctionnalité métier ou IT*** exposé par un fournisseur. Un service doit répondre aux contrainte suivantes:

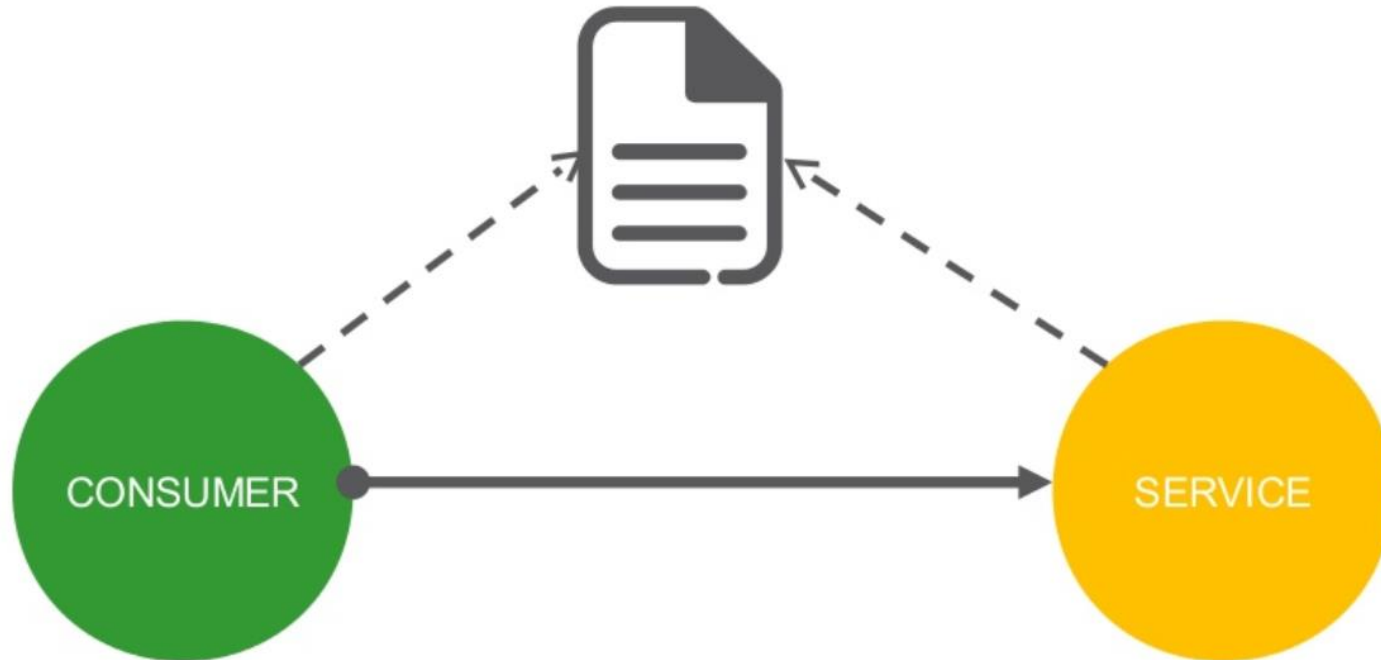
- **Contrat de service**
- **Couplage faible**
- **Autonomie**
- **Abstraction**
- **Réutilisation**
- **Stateless**
- **Découverte de service**
- **Composition de service**
- **Interopérabilité**

# SOA

## 1. Contrat de service



***Les services doivent adhérer à un contrat où le service et les conditions d'utilisations sont définies***

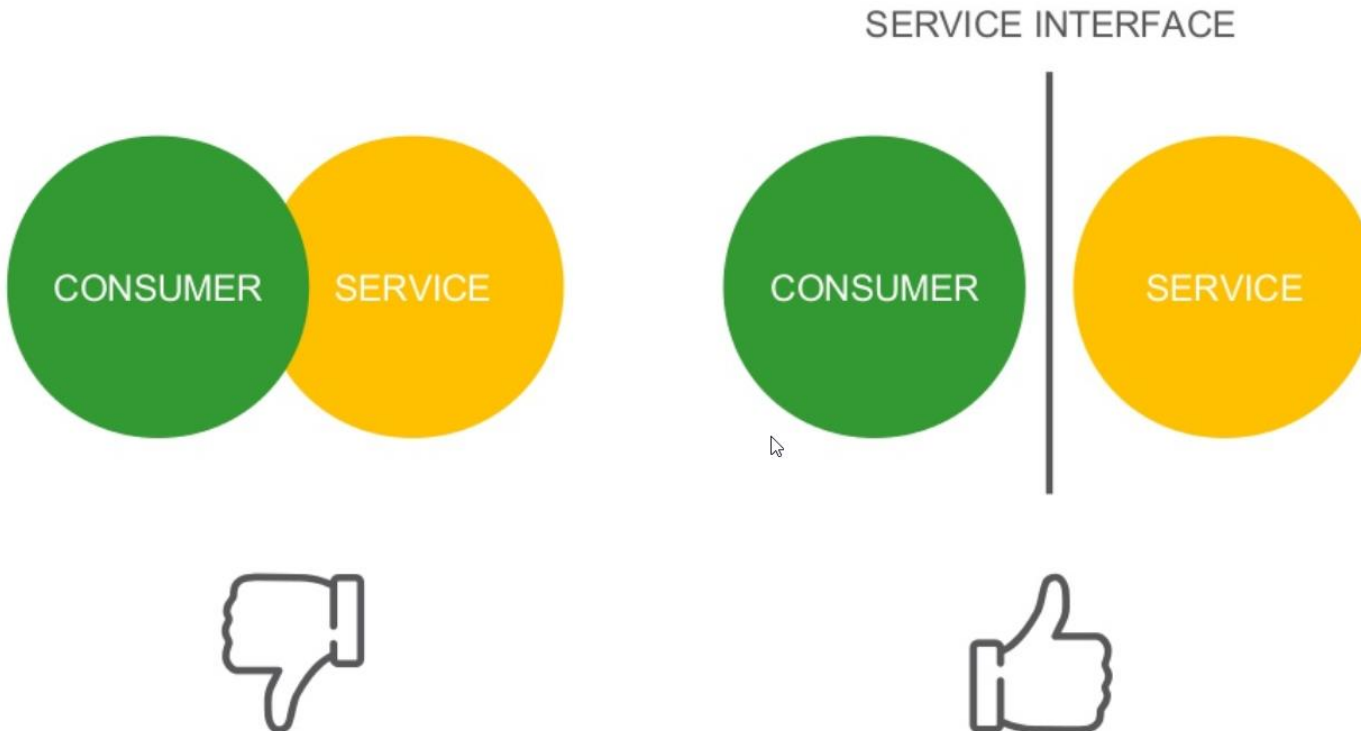


# SOA

## 2. Couplage faible



***Cacher les détails d'implémentation au consommateur et minimiser les dépendances***

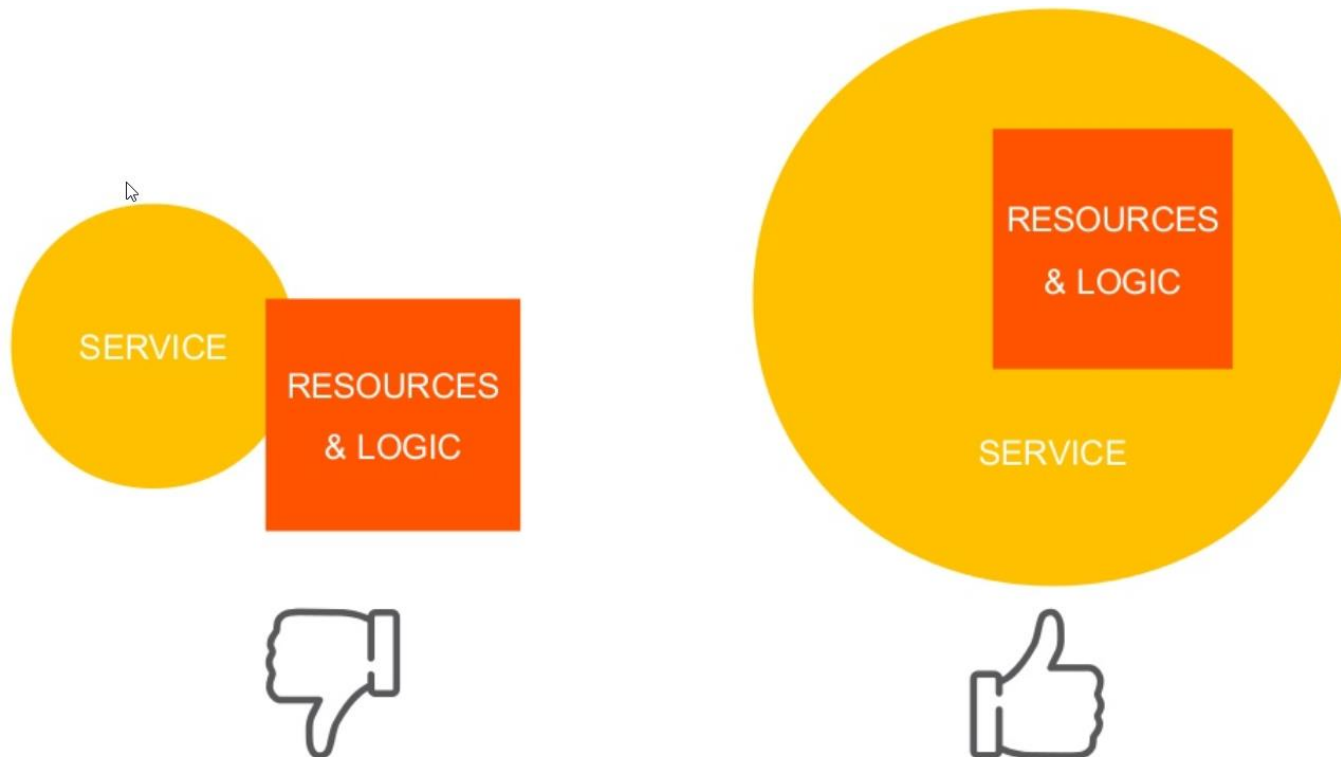


# SOA

## 3. Autonomie



***Les services doivent avoir le contrôle sur les ressources et la logique qu'ils encapsulent***

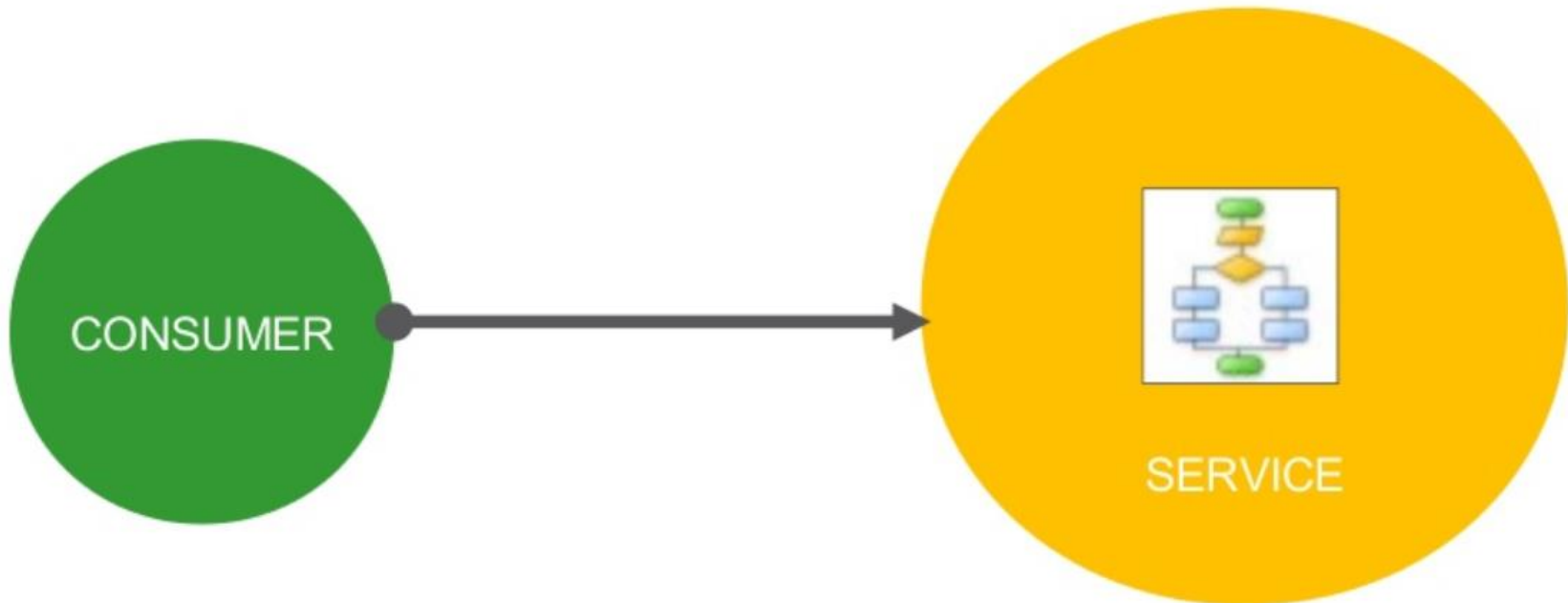


# SOA

## 4. Abstraction



*La logique interne du service doit être caché du consommateur*

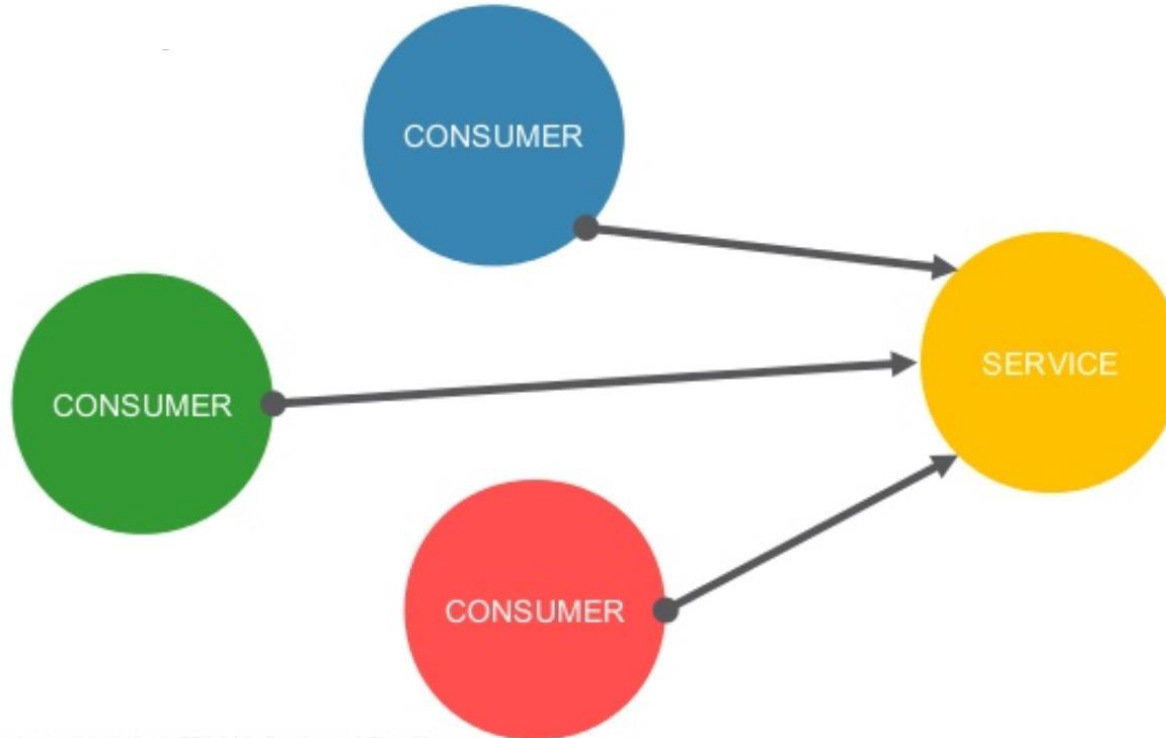


# SOA

## 5. Réutilisation



*La vraie valeur ajoutée d'un service réside dans sa capacité de réutilisation*

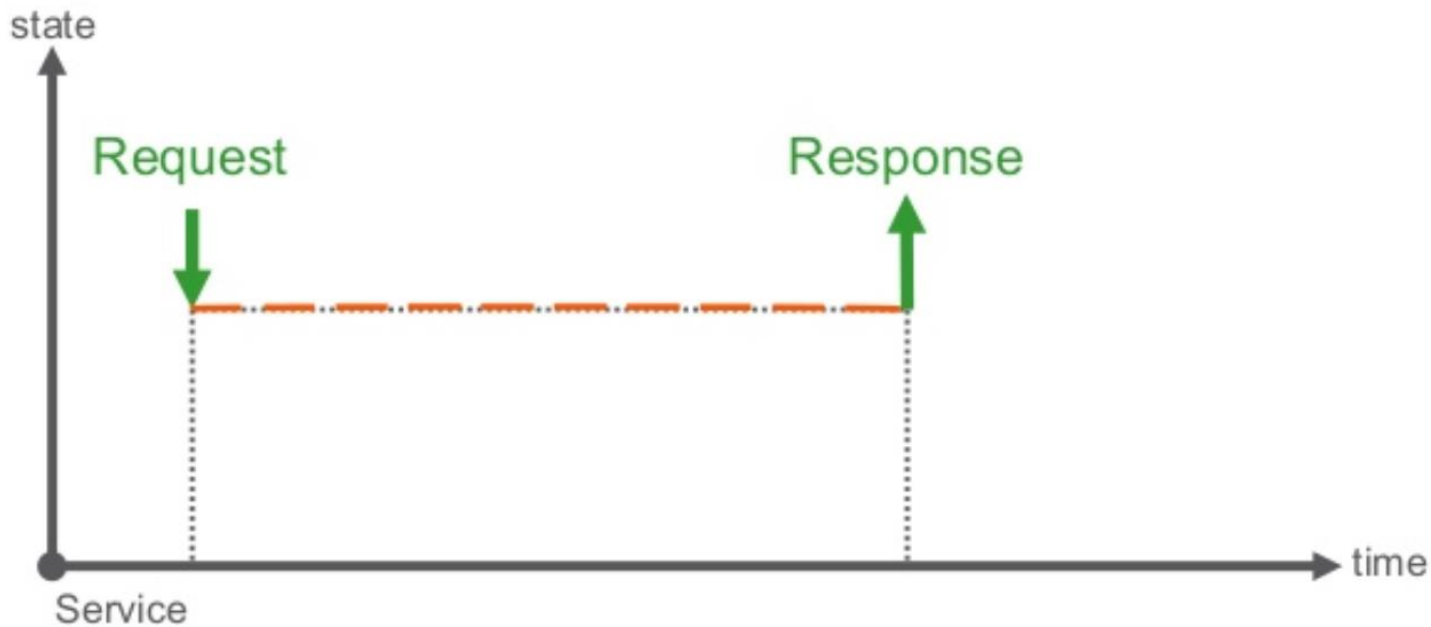


# SOA

## 6. Stateless



***La gestion des états dans un service affecte sa disponibilité et donc peut rendre le service limitée dans sa réutilisation***





# SOA

## 7. Découverte de service



*Les services doivent pouvoir être facilement accessibles et référencés*

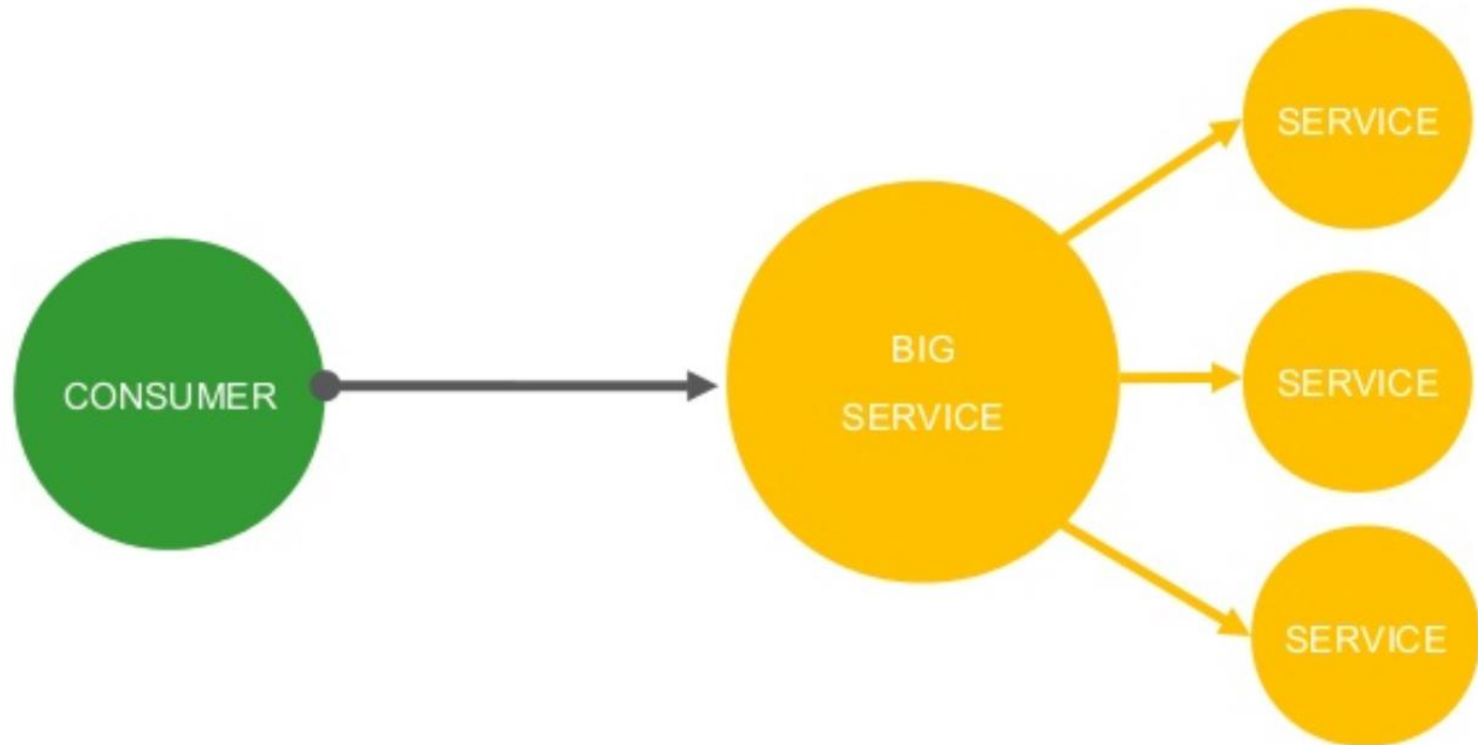


# SOA

## 8. Composition de service



***La conception de service doit permettre de composer des services sur la base de services existants***

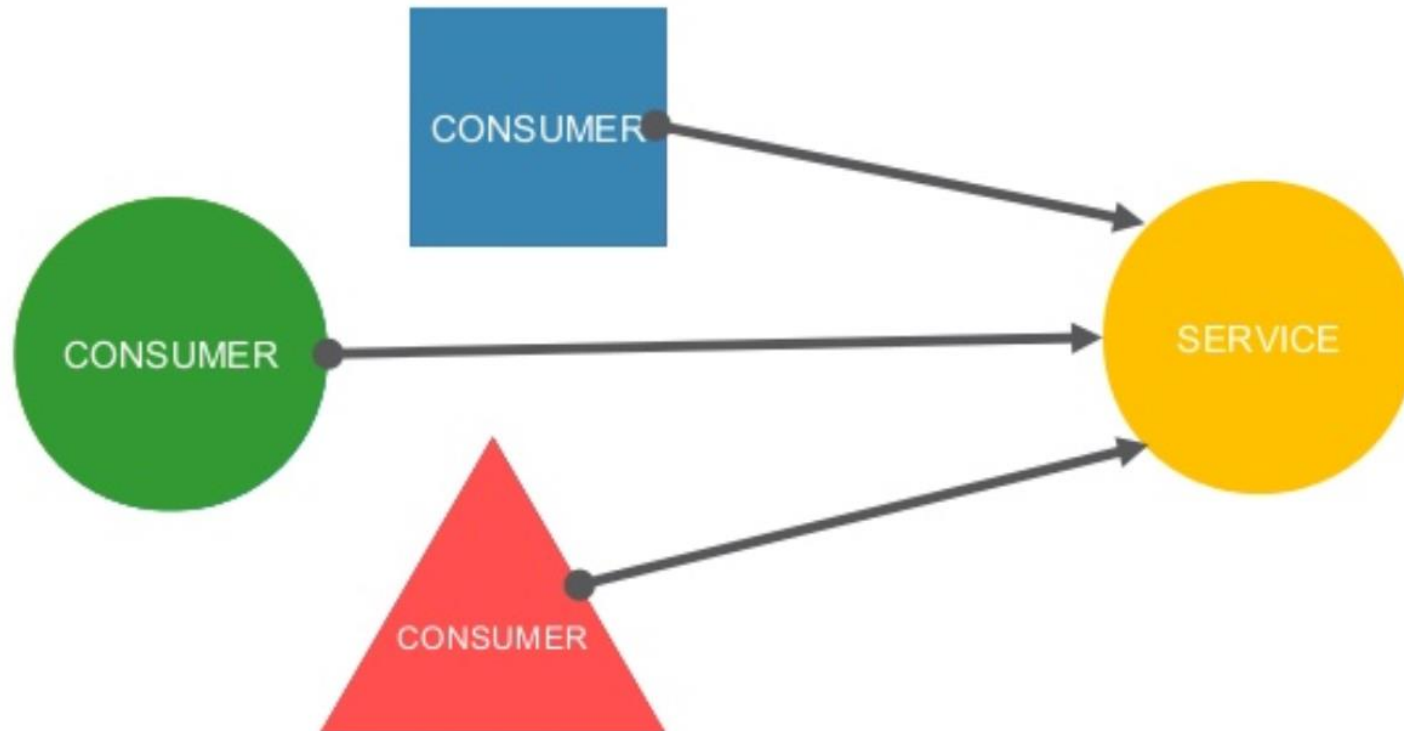


# SOA

## 9. Interopérabilité



***Un service doit baser son accès sur des standards afin de pouvoir servir plusieurs types de consommateurs***

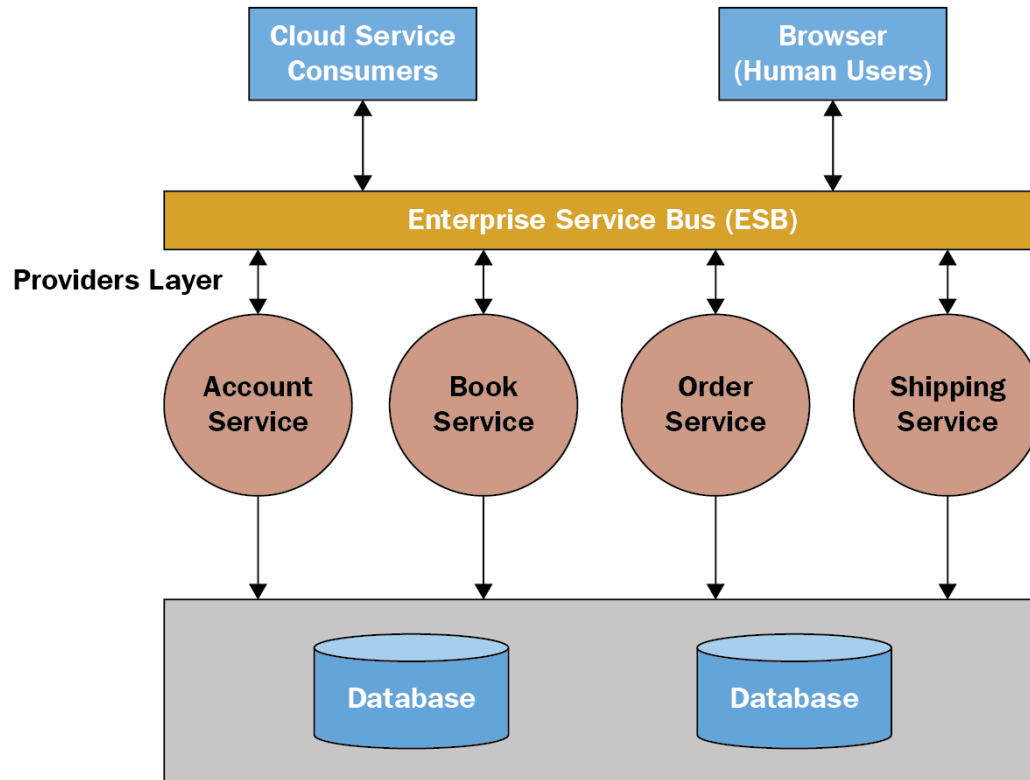


# SOA

## Implémentation typique



Consumers Layer

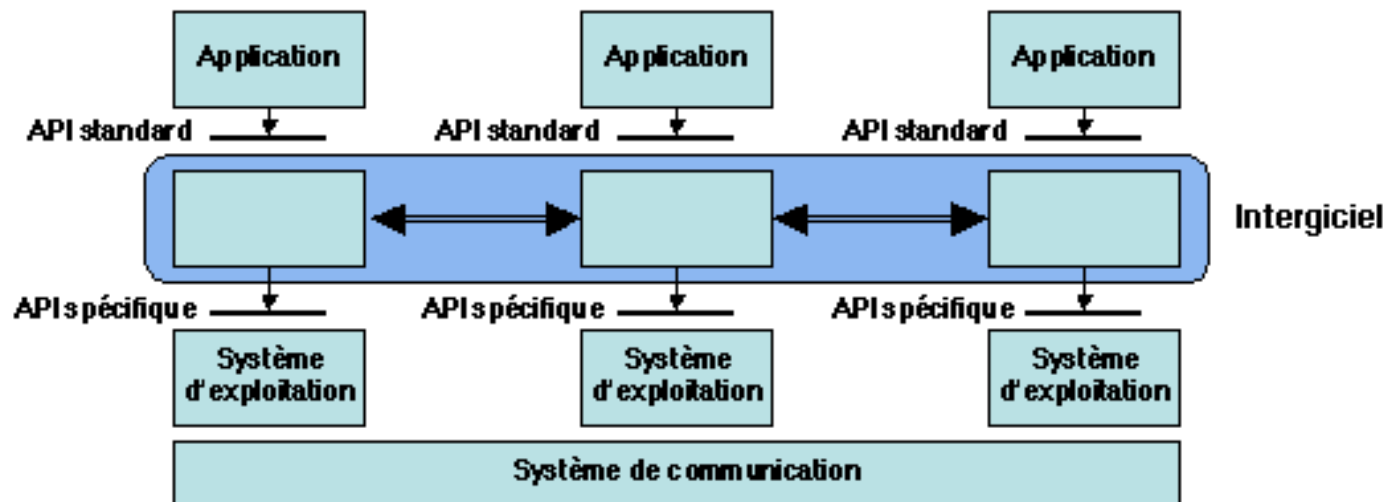


# Notion de Middleware

## ESB, Broker de messages

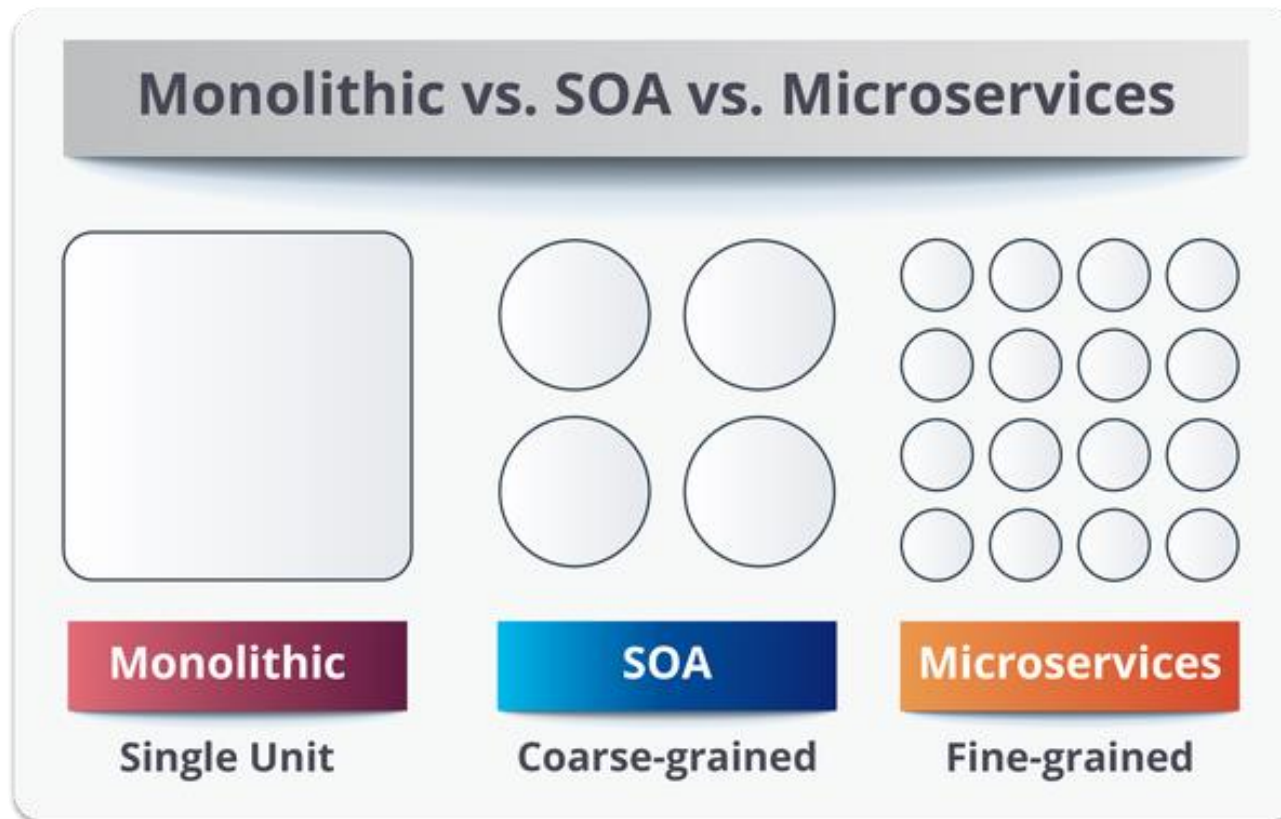


Un middleware est un logiciel (ou couche logicielle) tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques (souvent hétérogènes)



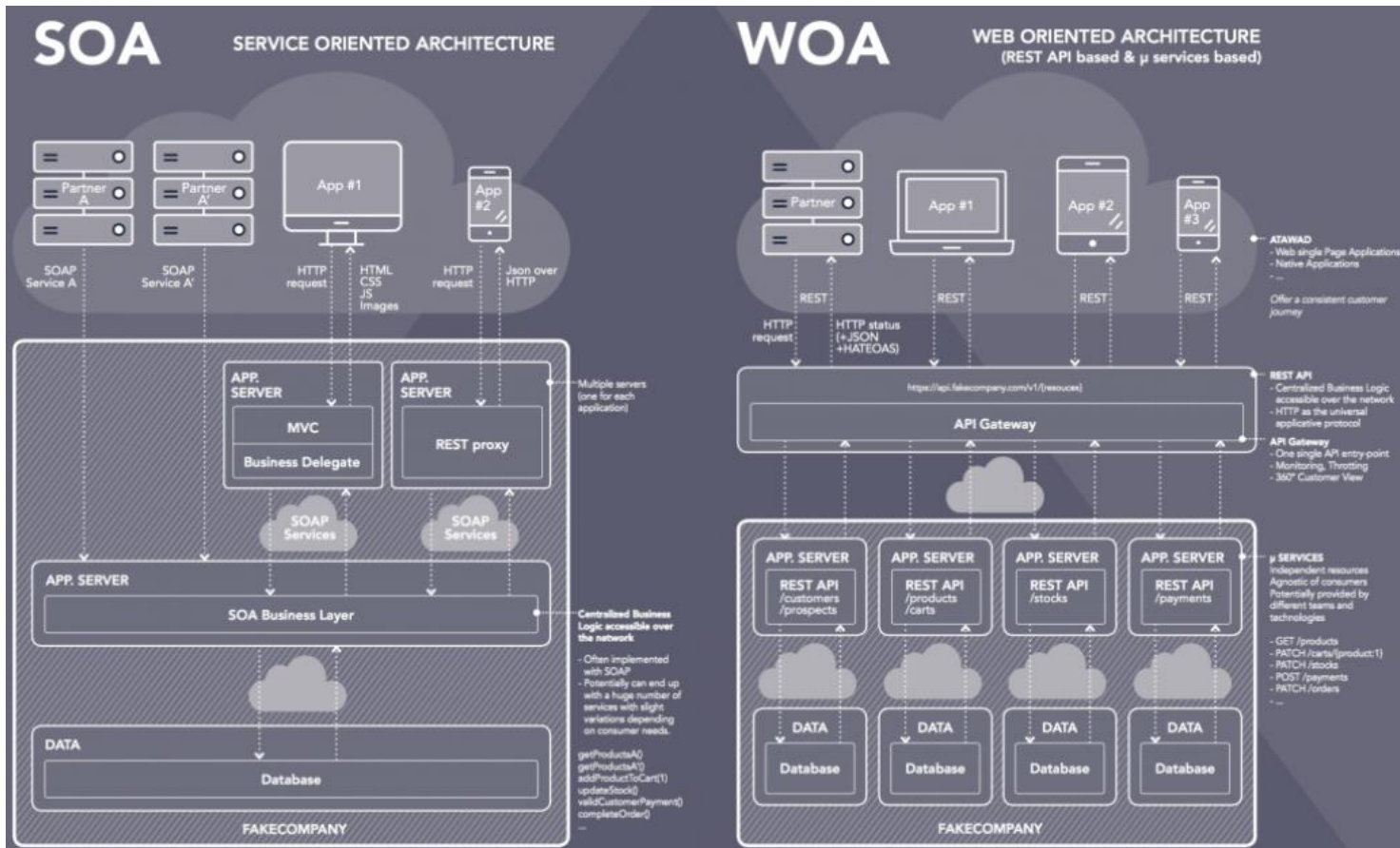
# SOA

## de SOA à WOA



# SOA

## de SOA à WOA



# SOA

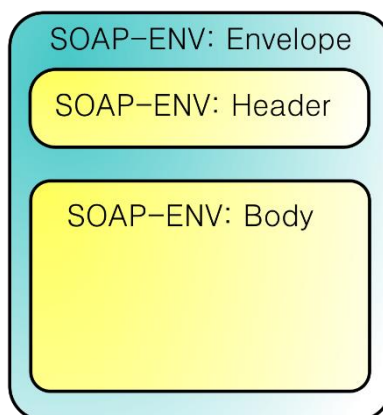
## SOAP - définition



**SOAP** (ancien acronyme de *Simple Object Access Protocol*) est un protocole d'échange d'information structurée dans l'implémentation de services web bâti sur *XML*.

Il permet la **transmission de messages** entre objets distants, ce qui veut dire qu'il autorise un objet **à invoquer des méthodes d'objets physiquement situés sur un autre serveur**. Le transfert se fait le plus souvent à l'aide du protocole **HTTP**.

C'est également **l'implémentation de référence** de SOA





# SOA

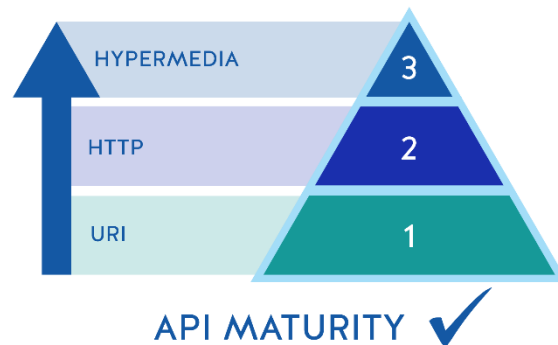
## REST - définition



**REST** (*representational state transfer*) est **un style d'architecture** logicielle définissant un ensemble de contraintes à utiliser pour créer des services web.

Les services web conformes au style d'architecture REST, aussi appelés services web **RESTful**, établissent une interopérabilité entre les ordinateurs sur Internet.

D'autres types de services web tels que les services web SOAP exposent leurs propres ensembles d'opérations arbitraires.

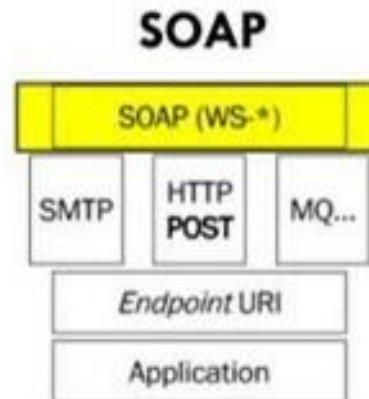
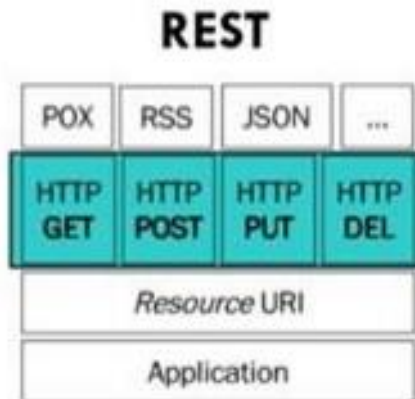


# SOA

## SOAP vs REST



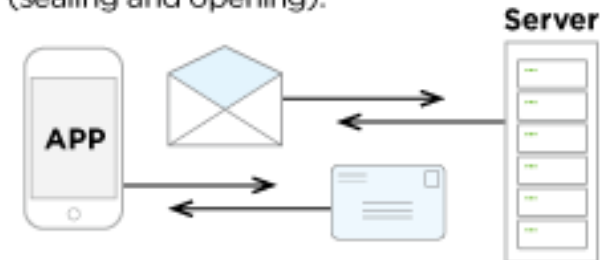
### Protocol Layering



### SOAP vs. REST APIs

SOAP is like using an envelope

Extra overhead, more bandwidth required, more work on both ends (sealing and opening).



REST is like a postcard

Lighterweight, can be cached, easier to update.

# SOA

## SOAP vs REST



#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

# SOA

## de SOA à WOA, différences



### SOA

Modélisation par **opération**

Cherche à unifier le modèle de programmation distribué et local par **l'intermédiaire d'un proxy**

Repose sur un **toolkit** pour être interprété

Consommé que par **des serveurs**

Basé sur **protocole spécifique**

### WOA

Modélisation par **ressources**

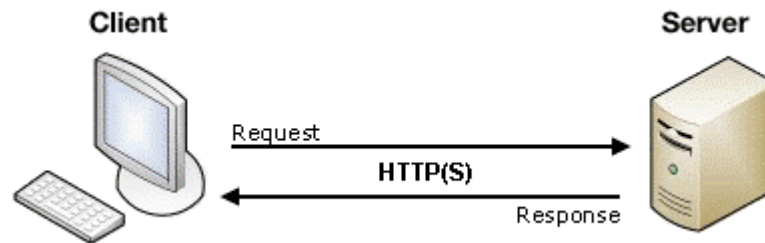
Modèle de programmation distribuée **explicite**, et baser sur le **WWW**

Mise sur une **bonne expérience développeurs** tous niveaux

Consommé par **tout type de terminal**, y compris les serveurs

Basé sur **HTTP**

# Applications Web Client-serveur

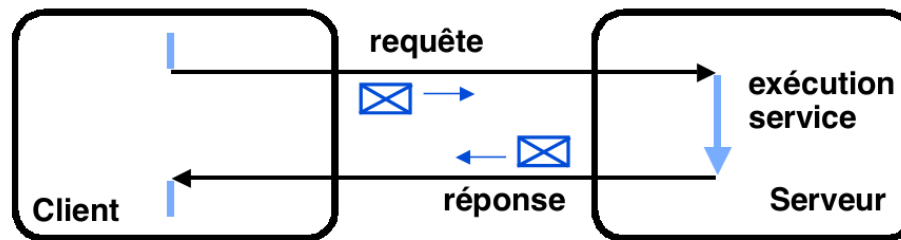


# Architecture Client Serveur

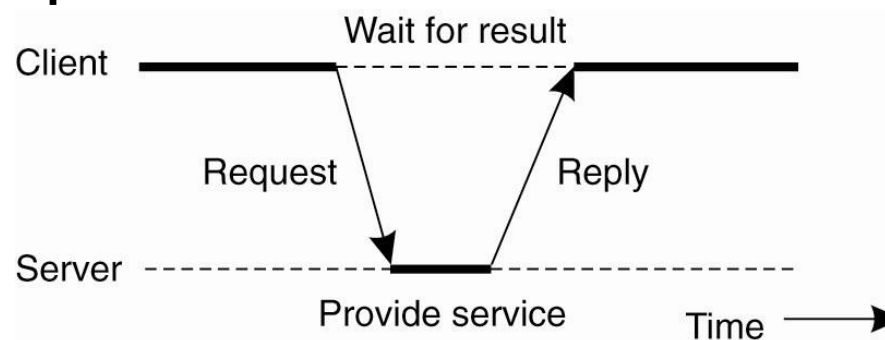
## Concepts et notions de bases



### ■ Concepts

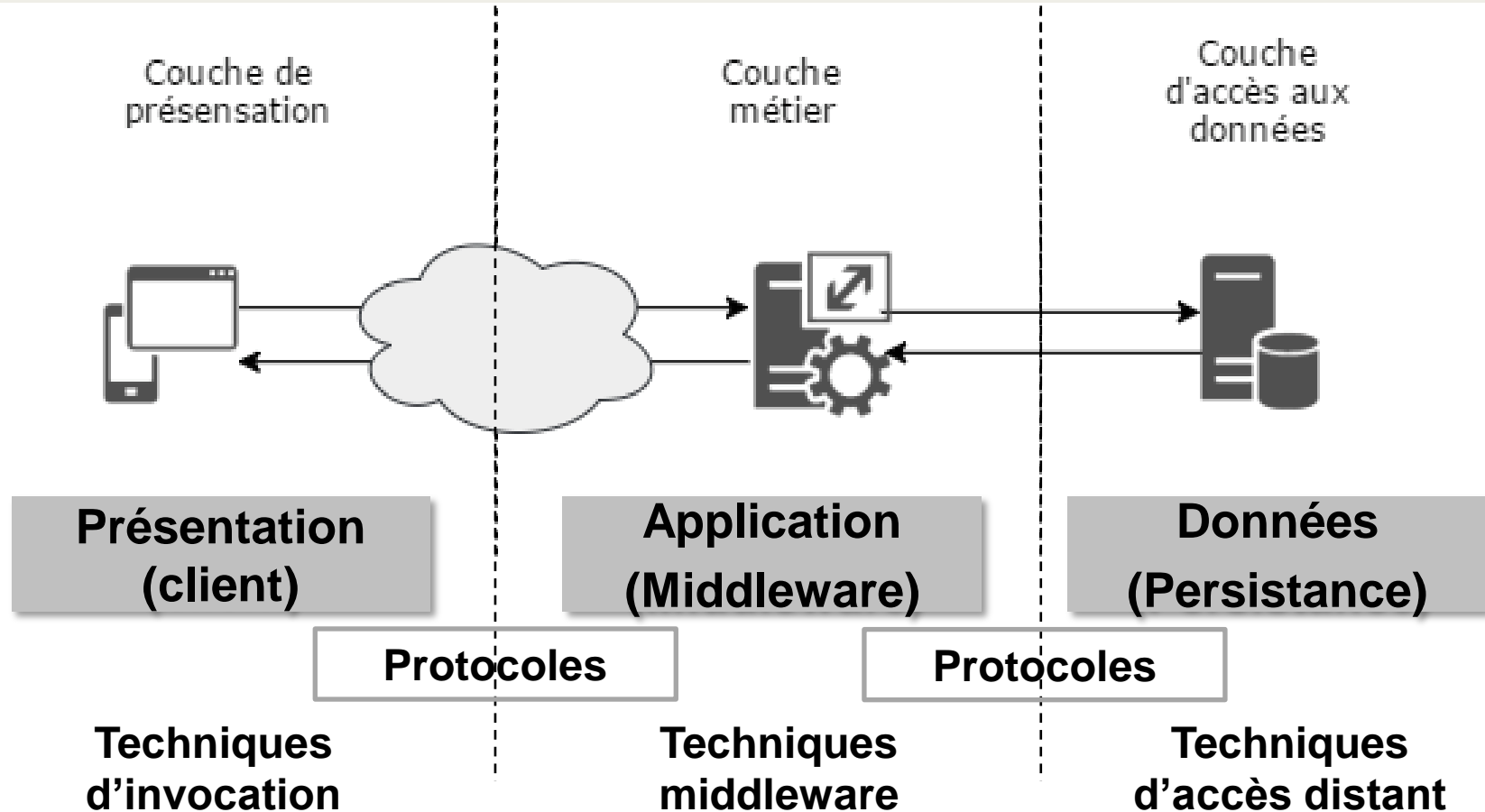


### ■ Exécution temporelle



# Architecture Client Serveur

## Architecture 3Tiers

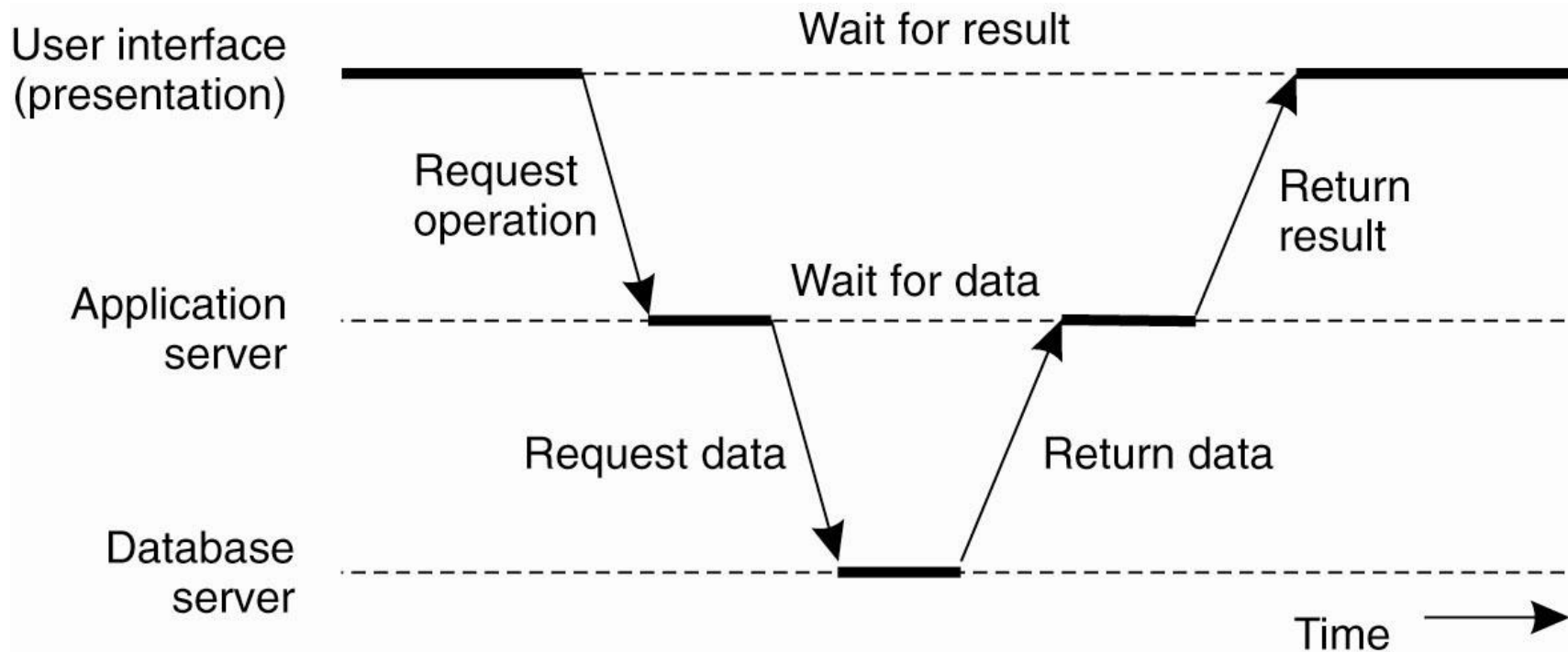


# Architecture Client Serveur

## Architecture 3Tiers



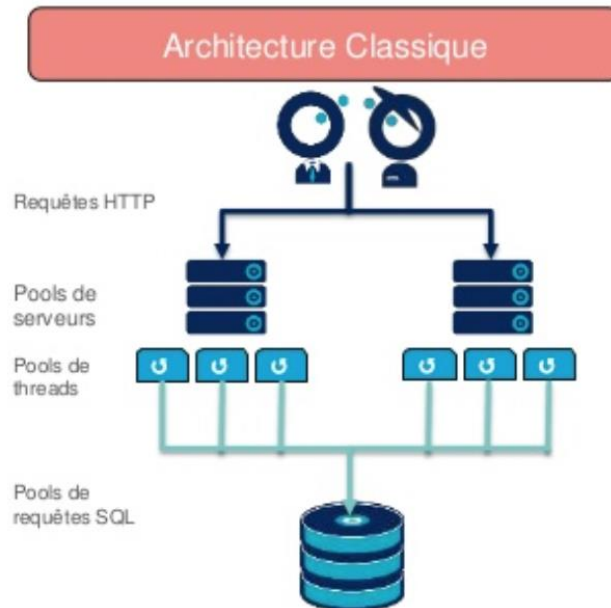
### ■ Requête client-serveur



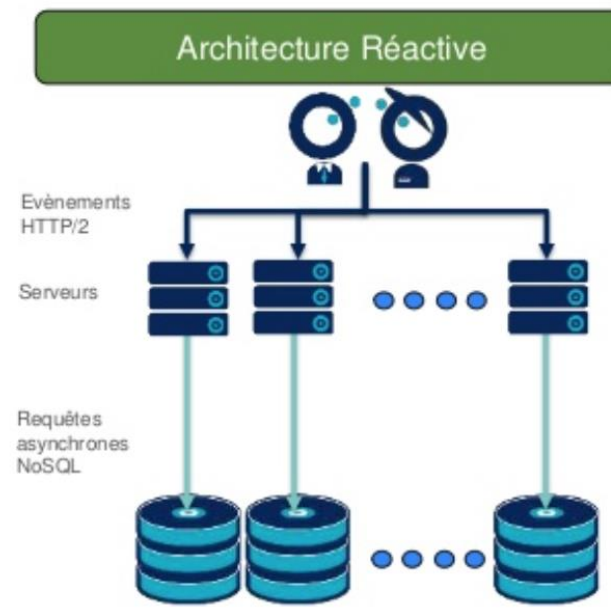


# Architecture Client Serveur

## Modèle d'exécution : bloquant vs non bloquant



- > Nombre de serveurs limité
- > Appels synchrones bloquants
- > Utilisation de pools limités de ressources
- > SPOF sur la base de données
- > Mauvaise exploitation des ressources
- > Et pourtant, limitation physique à tous les étages
- > Utilisation de verrous



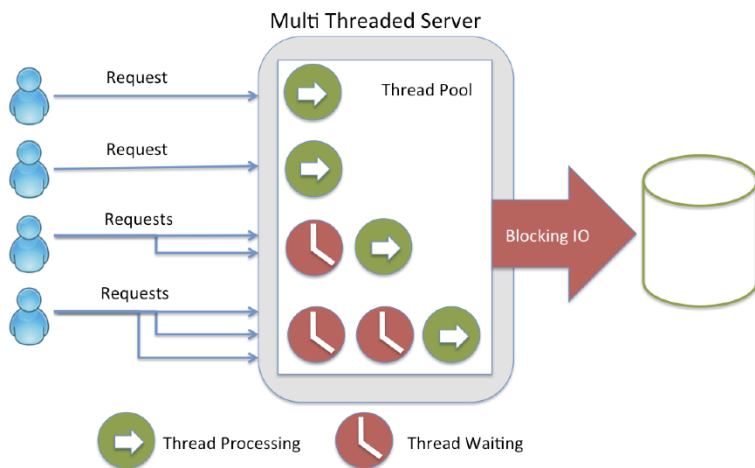
- > Nombre de serveurs illimité
- > Appels asynchrones non bloquants
- > Pas de pool de ressources
- > Pas de SPOF
- > Utilisation optimisée des ressources
- > Aucune limite physique
- > Pas ou peu de verrous

# Architecture Client Serveur

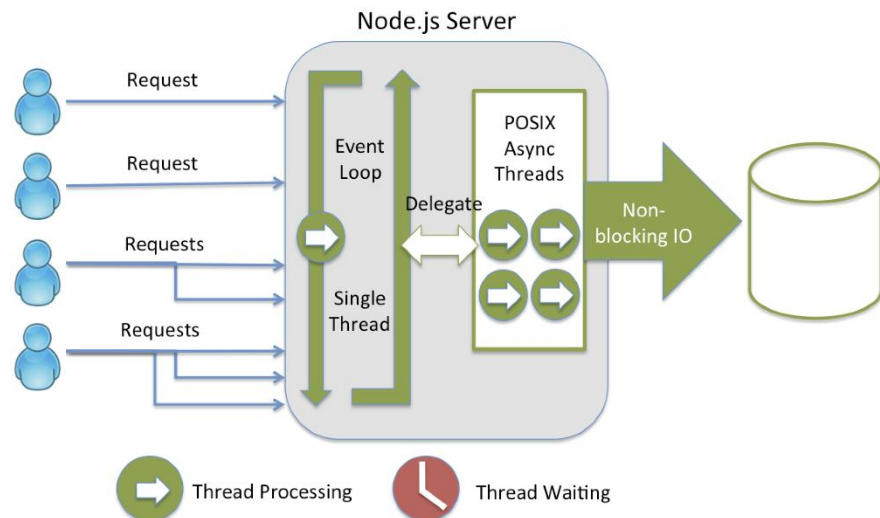
## Modèle d'exécution : bloquant vs non bloquant



### ■ Classique (bloquant)



### ■ Réactif (non-bloquant)

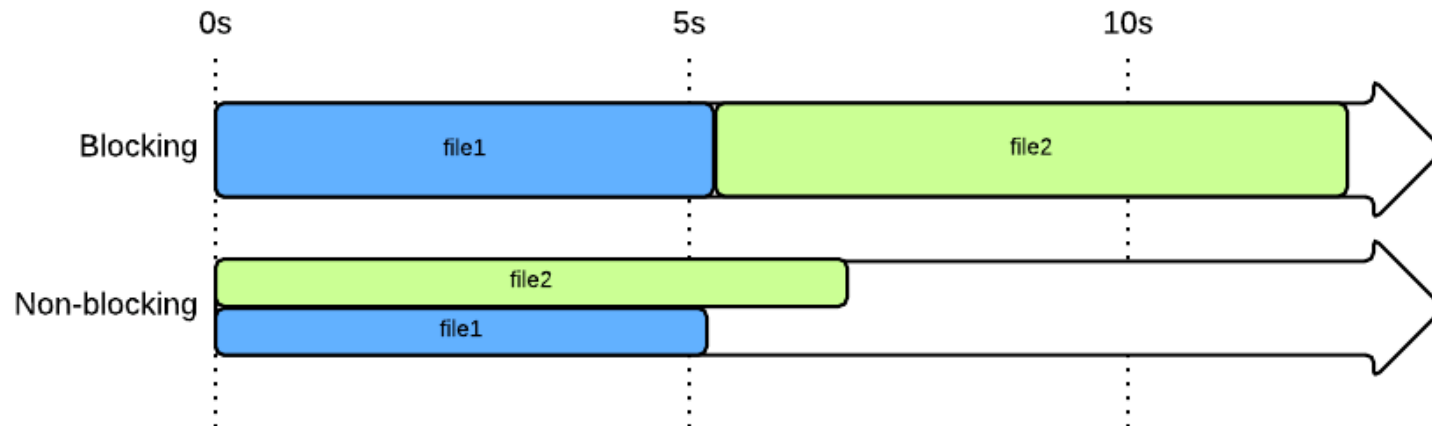


# Architecture Client Serveur

## Modèle d'exécution : bloquant vs non bloquant

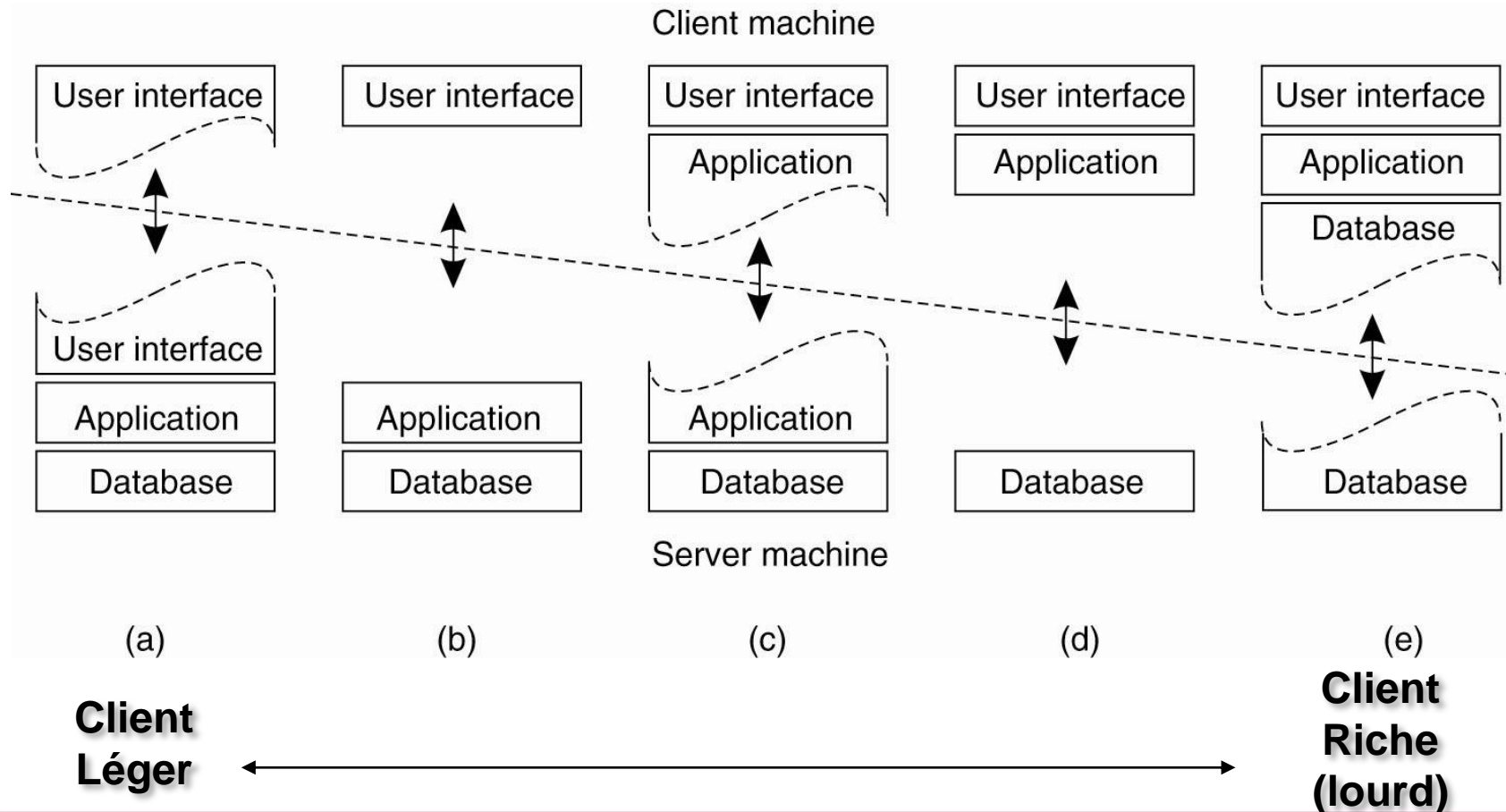


### ■ Comparaisons temps d'exécutions



# Architecture Client Serveur

## Les types de clients et évolutions



# Architecture Client Serveur

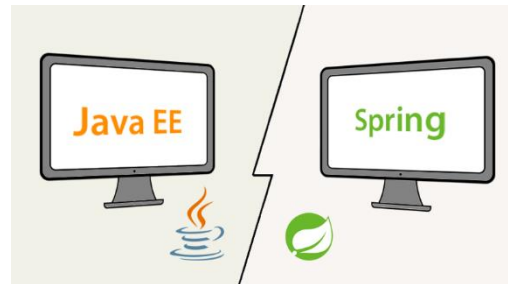
## Service Web



- Un « Web Service » est un composant logiciel
  - Accessible sur le réseau (URI)
  - Que l'on peut invoquer (SOAP, HTTP, TCP/IP, SMTP, MIME)
  - Décrit (WSDL, JSON Schema)
  - Publié et qui peut être recherché (UDDI, Service Discovery)
  - Qui peut être composé avec d'autres Web Services

En résumé un service web est un composant logiciel respectant les concepts SOA et accessible via une URL

# Java EE versus Spring



# Java EE

# Java Enterprise Edition



# Spécifications de la plateforme Java Avant 2017



## ■ Java Micro Edition (JME)

- développement d'applications embarquées et/ou mobiles

ORACLE®



## ■ Java Standard Edition (JSE)

- développement d'applications classiques de bureaux

ORACLE®



## ■ Java Enterprise Edition (JEE)

- développement d'applications d'entreprise

ORACLE®





# Spécifications de la plateforme Java

## Après 2017



- **Java Platform, Micro Edition (JME)**

- ☐ développement d'applications embarquées et/ou mobiles

ORACLE



- **Java Standard Edition (JSE)**

- ☐ développement d'applications classiques de bureaux

ORACLE



- **Jakarta EE**

- ☐ développement d'applications d'entreprise

ECLIPSE  
FOUNDATION



## Java EE selon Wikipédia

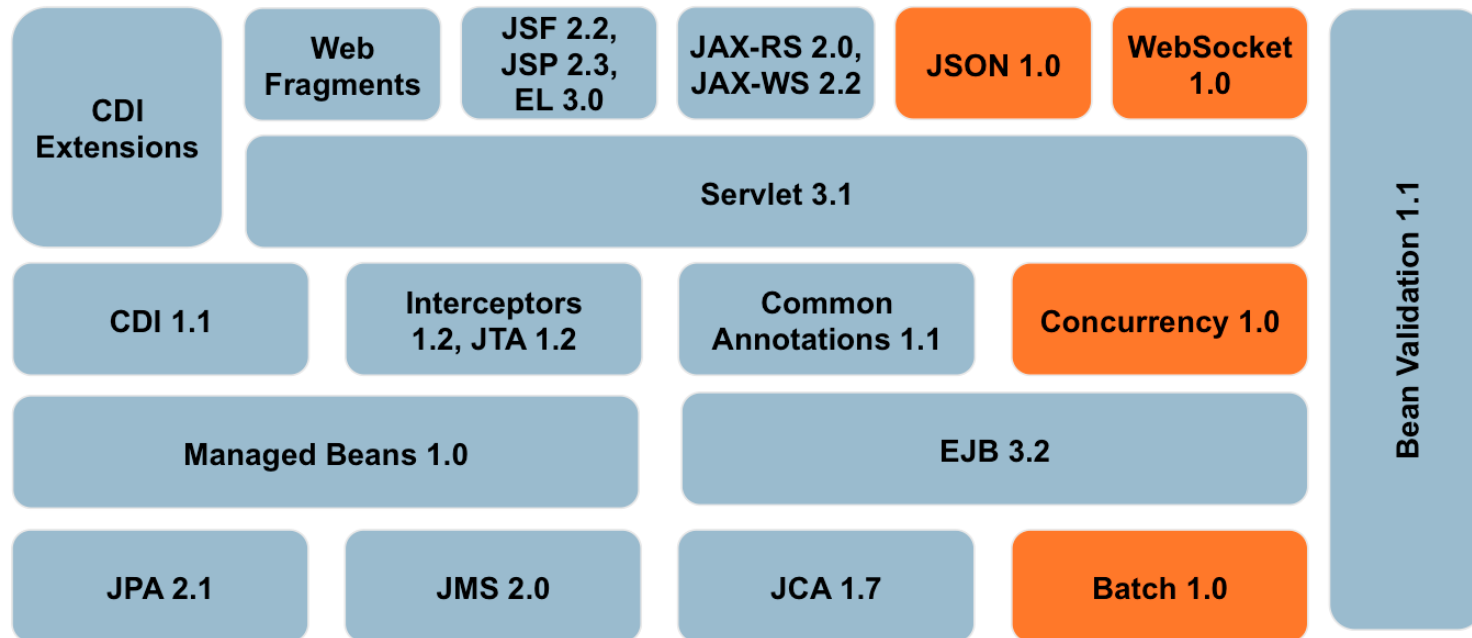
### Définition



- **Java Enterprise Edition**, ou *Java EE* (anciennement *J2EE*), est une **spécification** pour la technique Java de *Sun/Oracle* plus particulièrement **destinée aux applications d'entreprise**
- Ces applications sont considérées dans une approche multi-niveaux. Dans ce but, toute implémentation de cette spécification contient un **ensemble d'extensions au framework** Java standard (*JSE, Java Standard Edition*) afin de faciliter notamment la création d'applications réparties
- Une application Java EE **doit** s'exécuter **dans un serveur d'application standardisée JEE** (qui implémente la spécification)

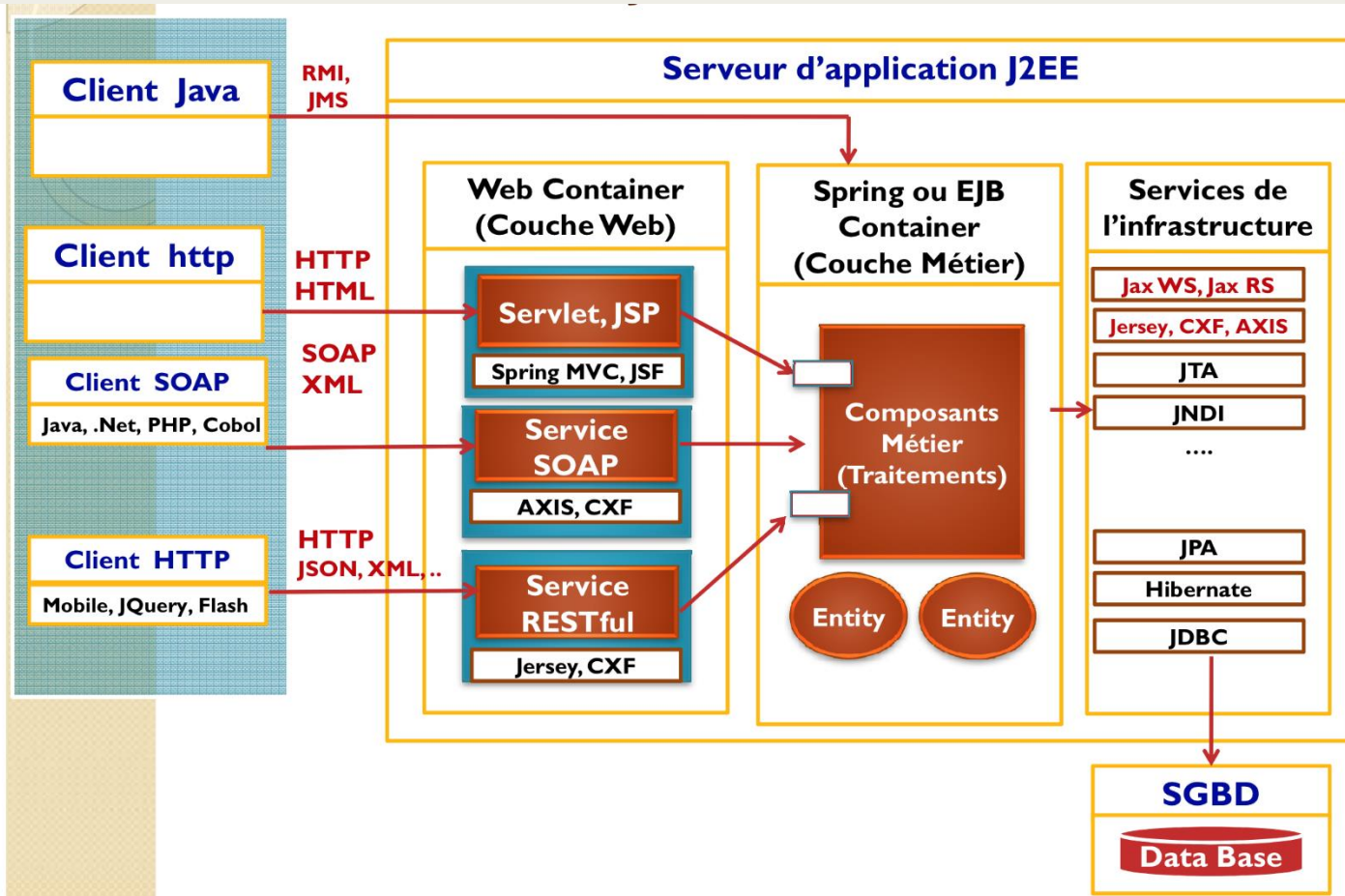
# Composants additionnels JEE

## Vue d'ensemble



# Architecture de référence JEE

## Synthèse



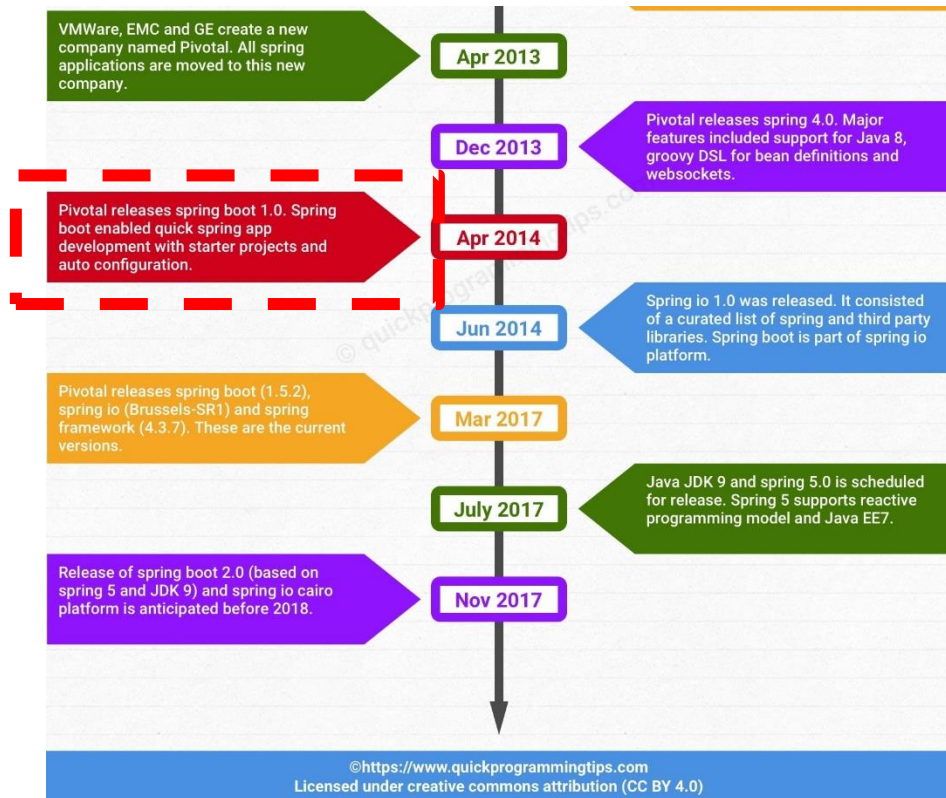
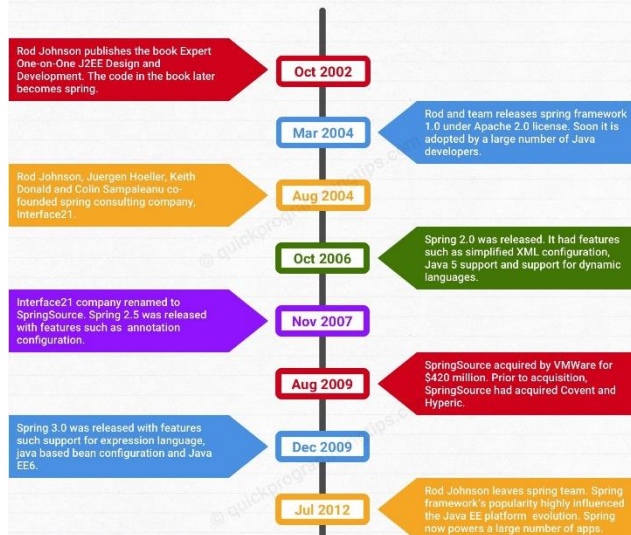
# Spring Framework



# Spring Historique



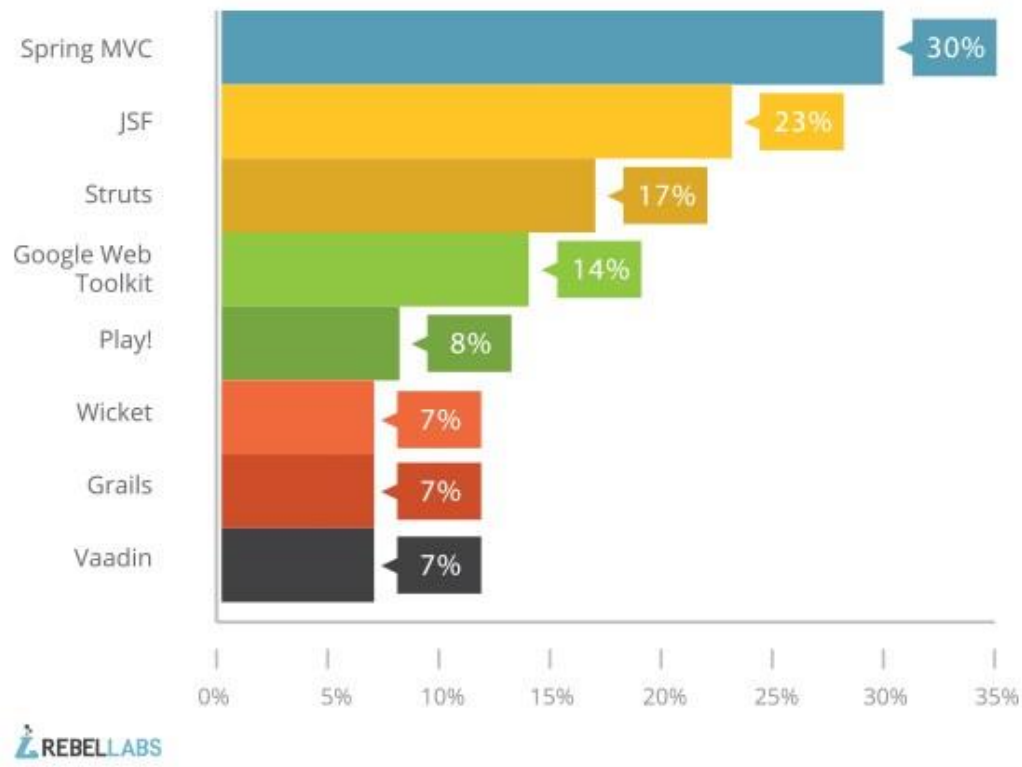
## Story of Spring Framework



# Spring Comparatif avec frameworks MVC



## ■ Statistiques des framework MVC Web / Java utilisés



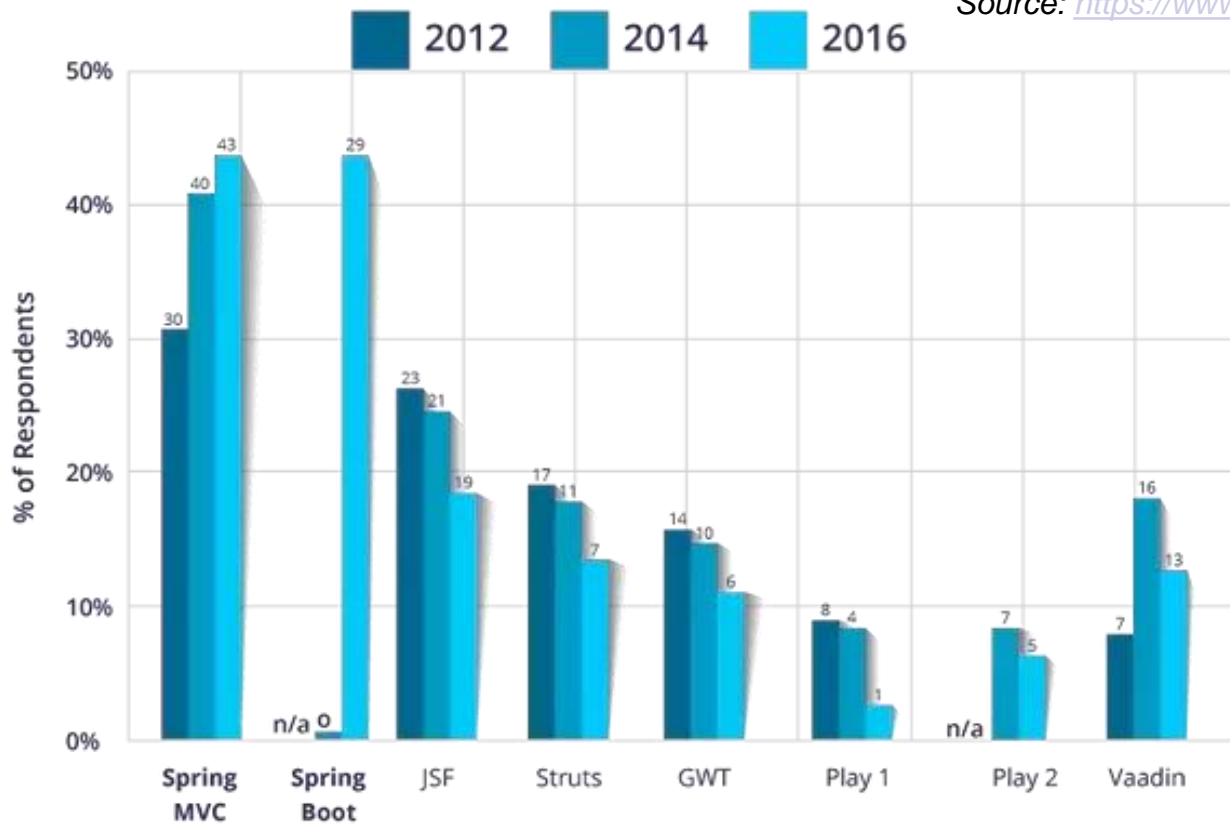
## Spring Définition



- **Spring** est un **framework libre** pour construire et définir l'infrastructure d'une application *Java*, dont il facilite le développement et les tests
- **Spring** est un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un **serveur d'applications JEE**. Les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le framework (au contraire des serveur d'applications JEE). C'est en ce sens que **Spring** est qualifié de conteneur « léger ». »
- **Spring** s'appuie principalement sur l'intégration de trois concepts clés :
  - **l'inversion de contrôle (IOC)** est assurée de deux façons différentes : la recherche de dépendances et l'injection de dépendances
  - **la programmation orientée aspect (AOP)**
  - **une couche d'abstraction**





# Spring Evolutions frameworks MVC



# Spring Versus JEE



## Java EE et/ou Spring

 <b>Java EE 7</b>	 <b>Spring 4</b>
Ensemble de <b>spécifications</b> définies par Sun/Oracle.	Ensemble de <b>bibliothèques</b> (Jar).
Nécessite un <b>serveur d'application</b> qui implémente les spécifications Java EE.	<b>Ne nécessite pas</b> un serveur d'application.
<b>Plusieurs</b> fournisseurs, tel que Redhat, IBM et ASF.	Un seul fournisseur.
License du fournisseur	License <b>Open source</b> ( <a href="#">Apache License</a> )
Moins flexible, les dépendances sont offertes par le serveur d'application.	Très <b>flexible</b> , tous les jars sont intégrés dans l'archive: ⇒ facile à mettre à jour les dépendances. ⇒ facile à changer l'environnement d'exécution.
Assure les besoins d'entreprise : Web, gestion des transactions, sécurité, messaging, intégrations ...	

# Spring Versus JavaEE



## Java EE et/ou Spring

