Cours « Architecture Orientée Service »

Tarak Chaari

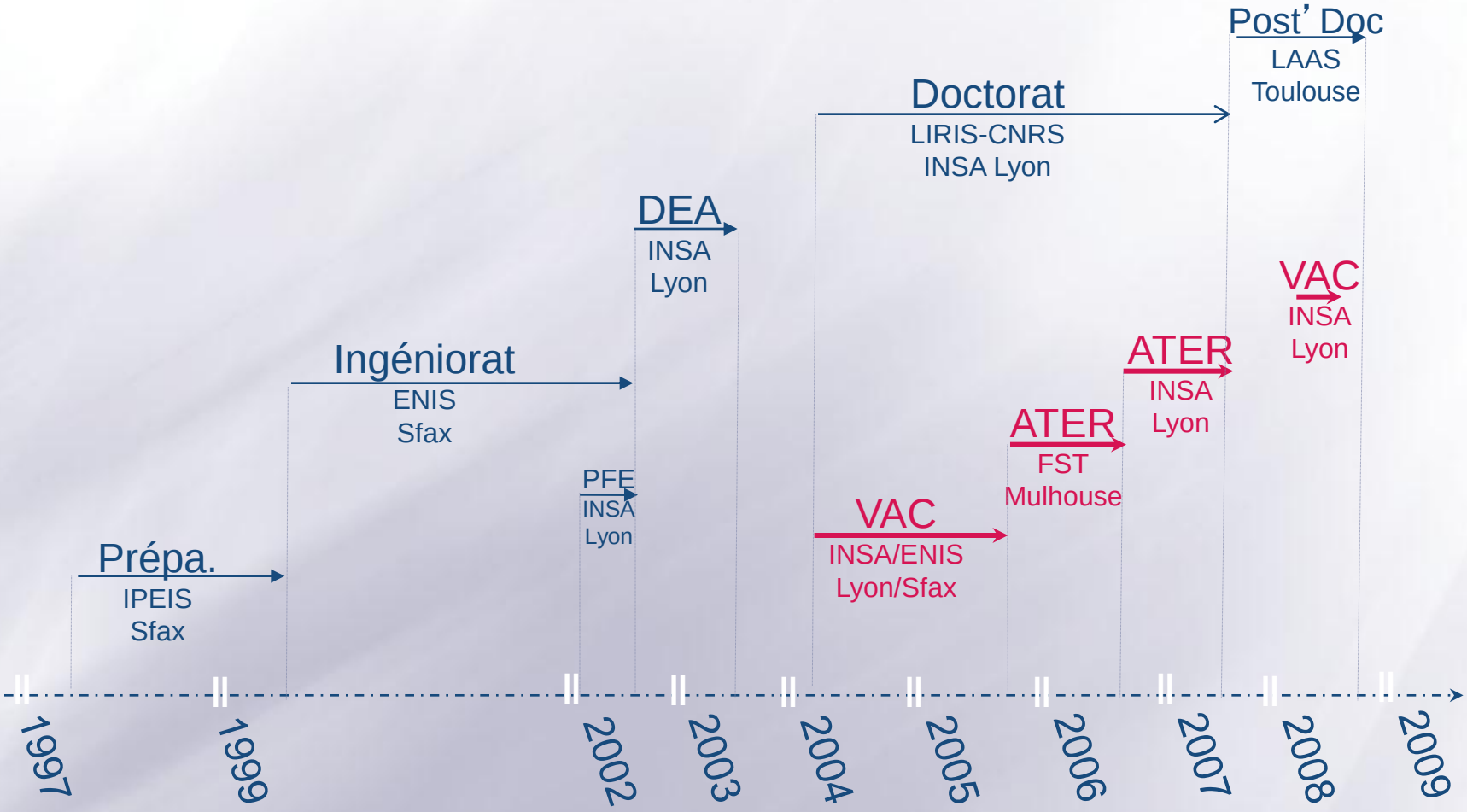
Maître assistant à l'institut supérieur d'électronique et de communication
tarak.chaari@redcad.org

http://www.redcad.org/members/tarak.chaari/cours/cours_soa.pdf

Votre interlocuteur

- Tarak CHAARI
- Maître assistant à l'ISECS
- Membre de l'unité de recherche RedCad
- Enseignement: Ingénierie des systèmes d'information
- Recherche:
  - l'adaptation dans les environnements dynamiques
  - sémantique dans les architectures orientées services

Cursus universitaire



Présentation générale du cours

Le nom du cours

- les architectures orientées service

Volume horaire

- 21 heures
- cours + Projets tuteurés

Objectif

- Avoir une idée sur le fonctionnement, le développement et les travaux de recherche des architectures orientées services

Contenu du cours

- Introduction sur les architectures logicielles
- Première notion de service en informatique : les sockets
- Deuxième notion de service en informatique : les RMI
- Les services WEB
- Travaux de recherche dans les architectures orientées service

En rodage !!!

Un peu d’histoire

Vue générale des technologies de développement (1/2)

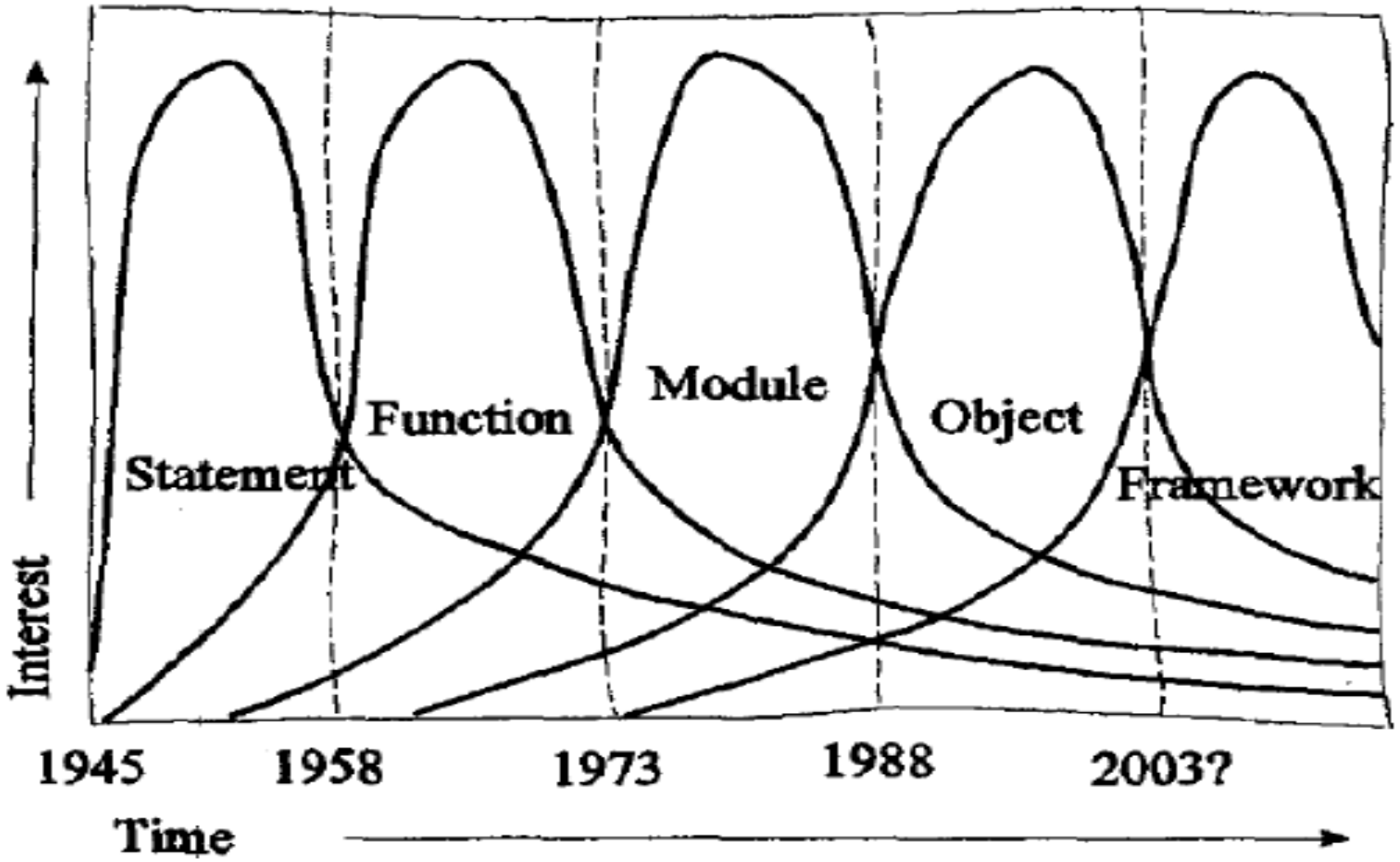
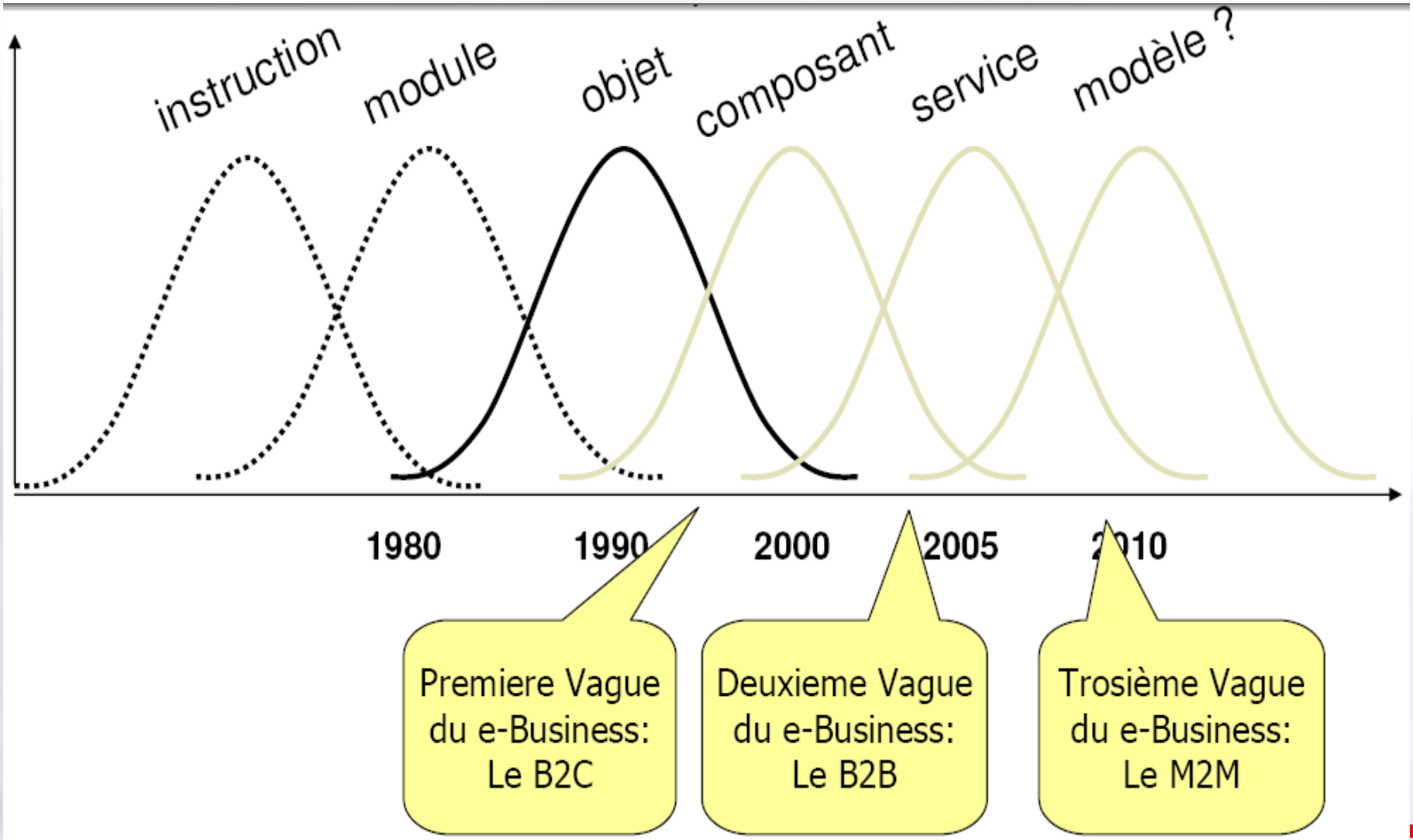


Figure 2: The Organization Stream.

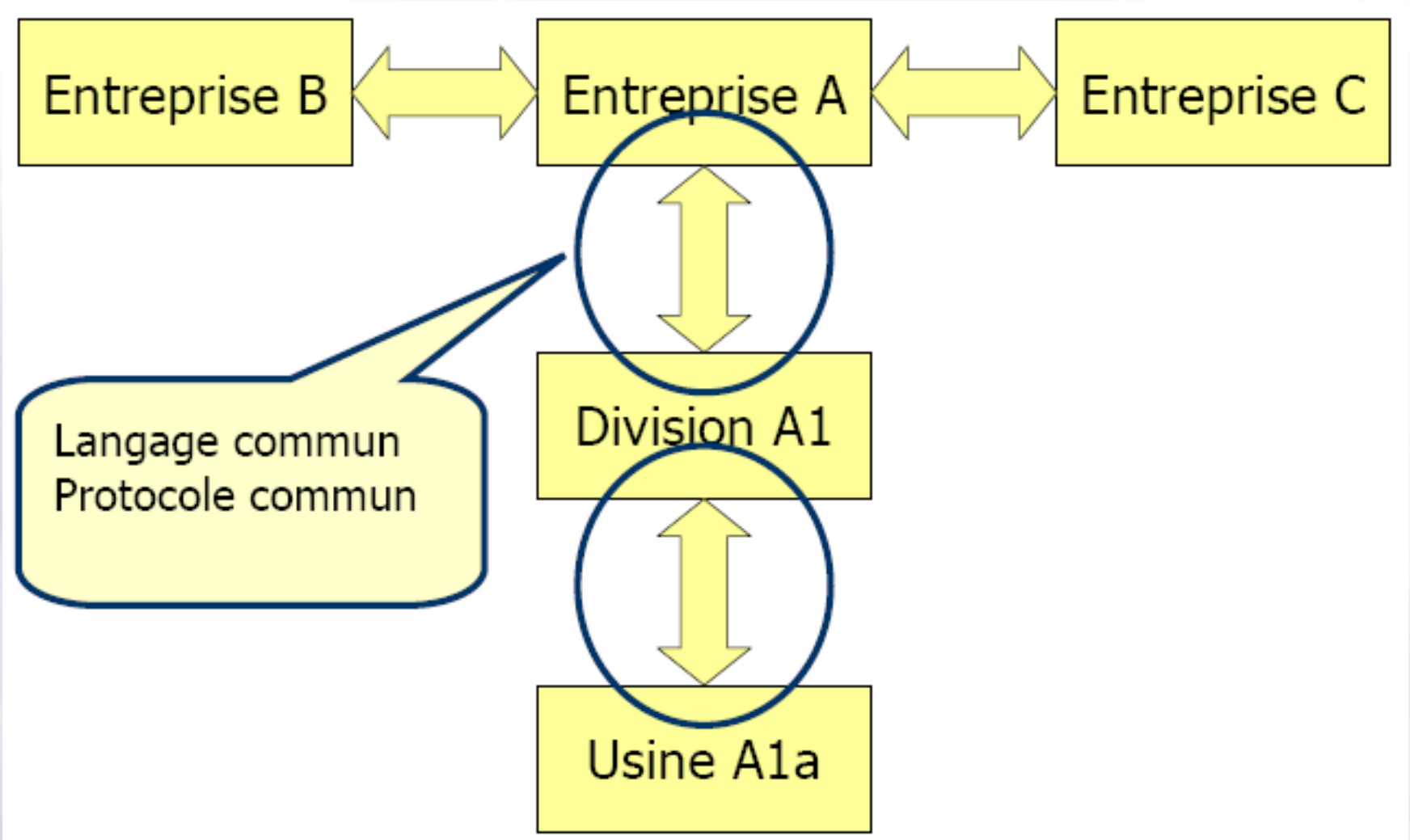
Vue générale des technologies de développement (2/2)



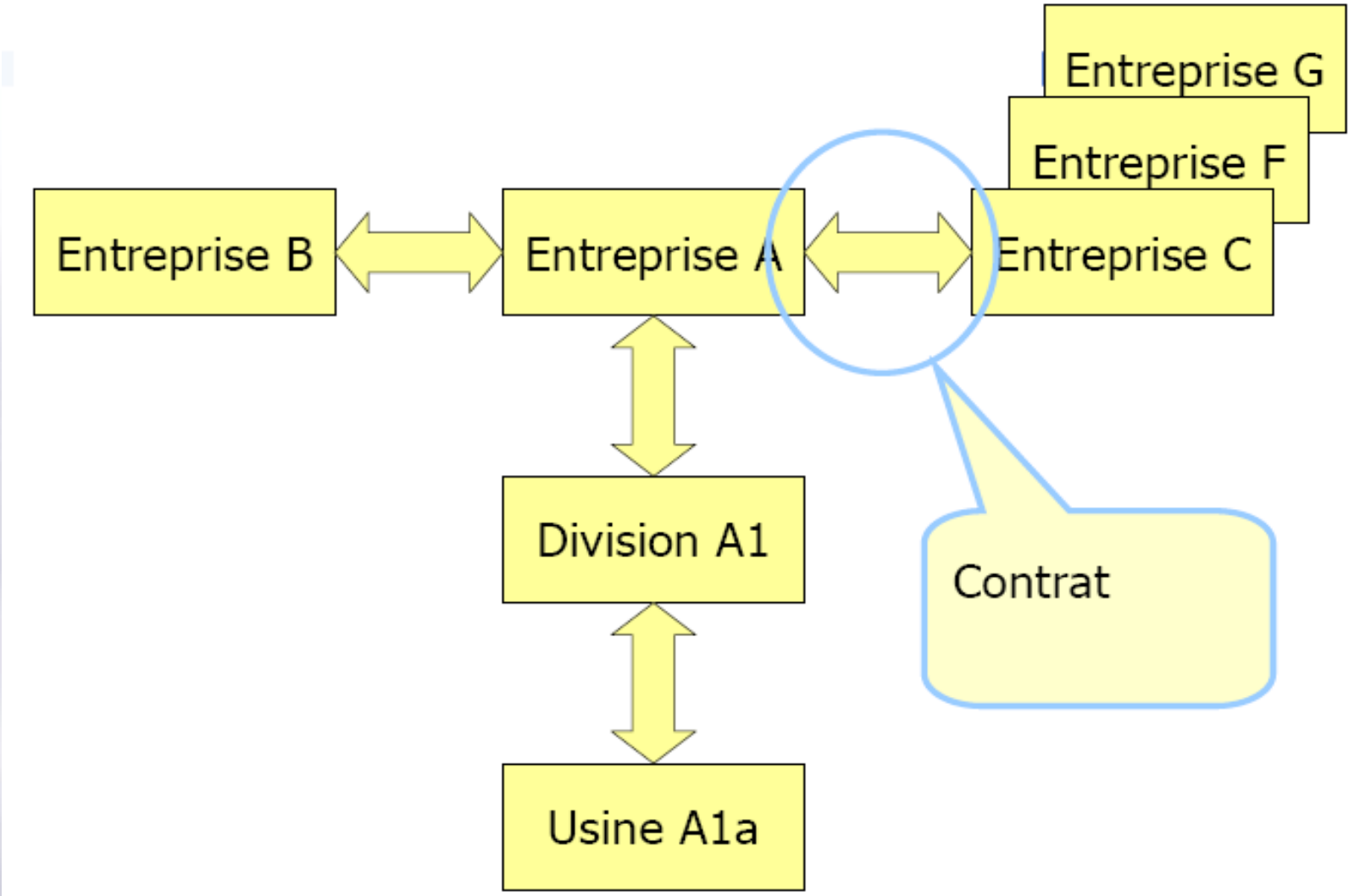
Synthèse sur les technologies de développement

- Débuts : Assembleur/Langages machine
- Puis ce fût au tour des applications tirant partie de langages procéduraux (C,Ada, Fortran, etc..)
- Puis les applications orientées objets (C++, Smalltalk, Eiffel, Java, C#)
  - moins de code à écrire (gestion de la mémoire automatique, pas de pointeur, API objets, …)
  - plus de briques à intégrer, design patterns
- Demain : architectures orientées services
  - intégrer les applications et gérer les évolutions technologiques

Le besoin d'intégration (1/2)

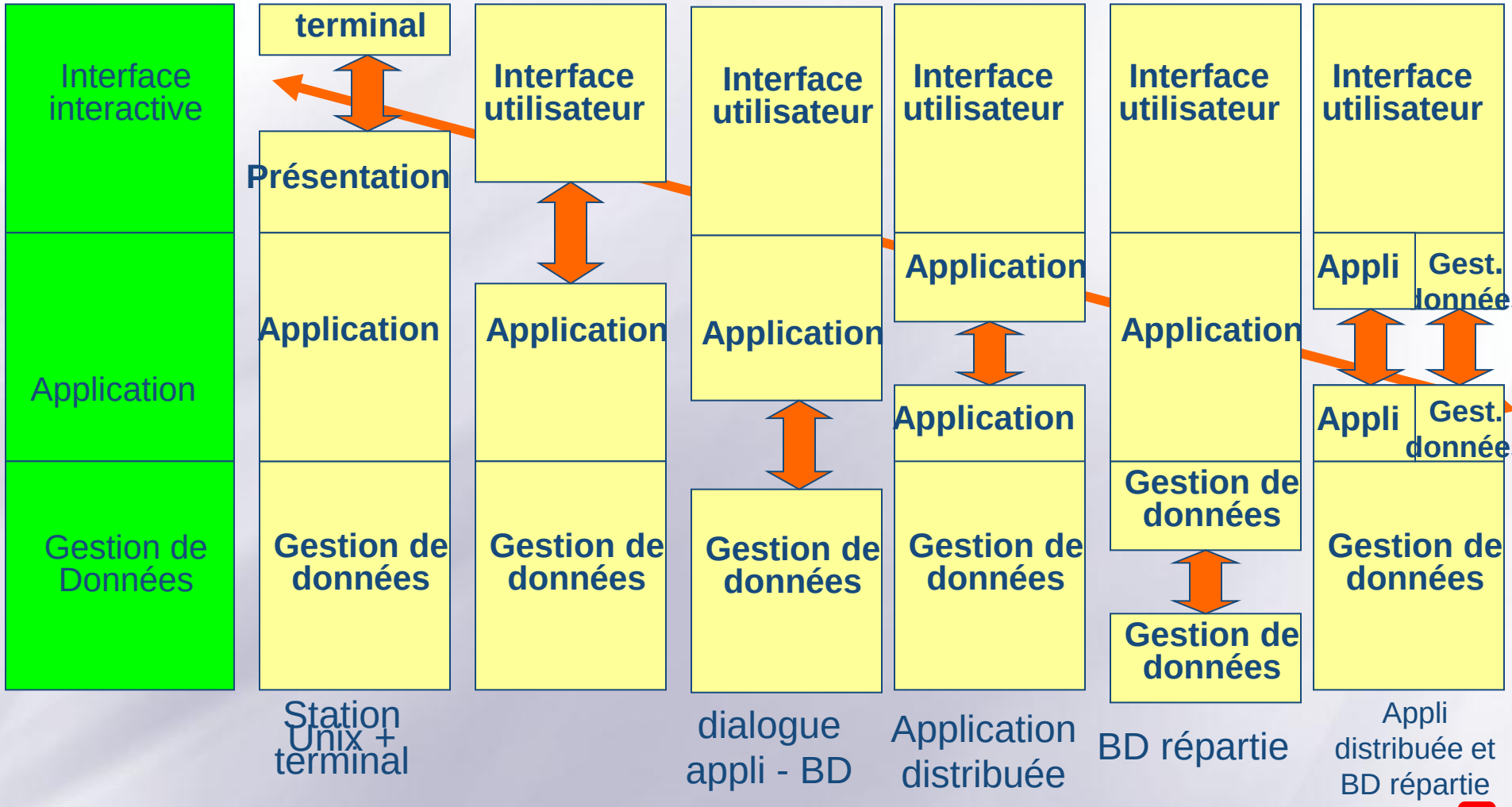


Le besoin d'intégration (2/2)

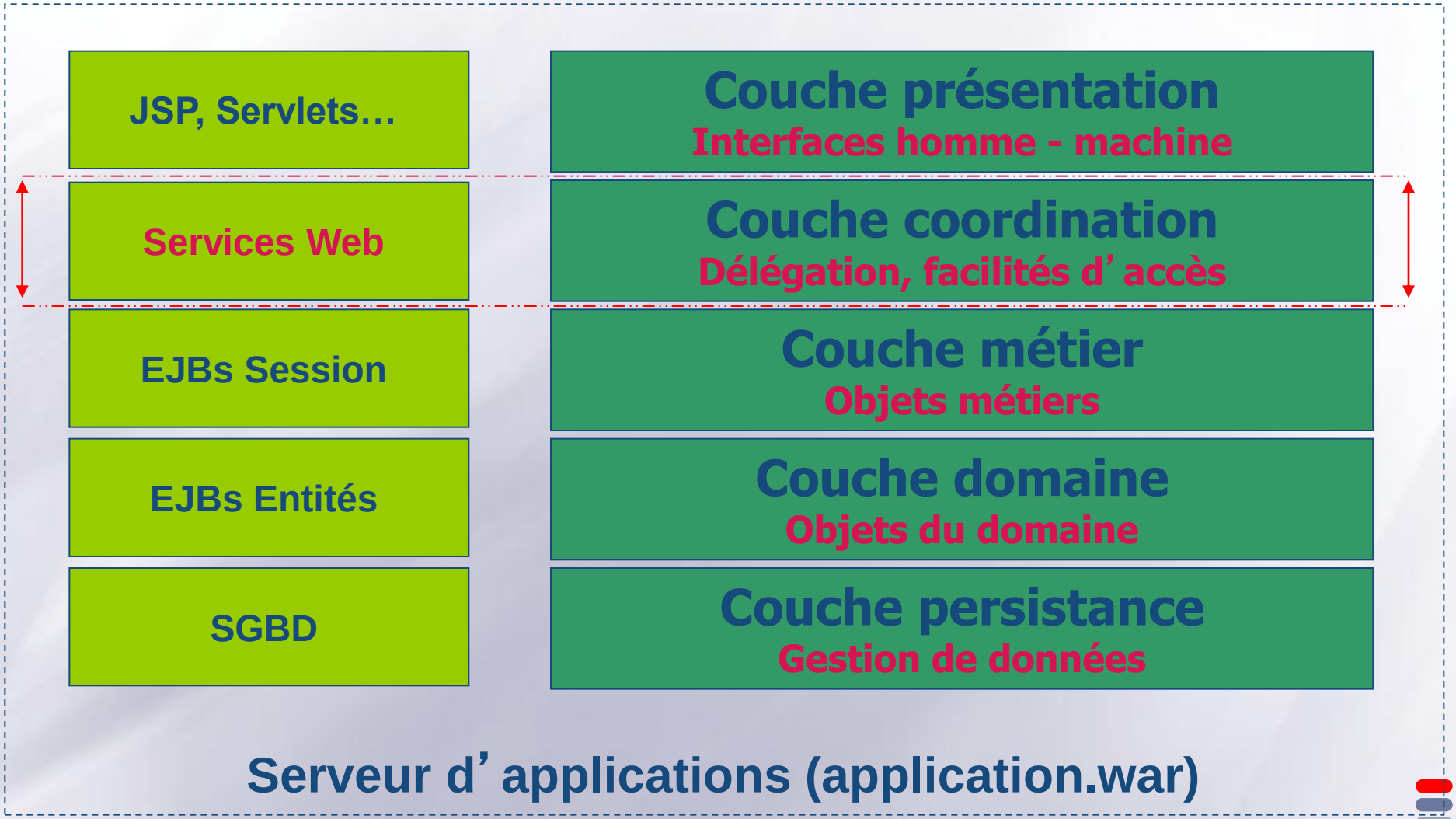


Besoin d'une nouvelle technologie dans le processus de développement logiciel

Découplage applicatif vers l'integration



Exemple d'une architecture à n-niveaux



Conclusion de la première partie

- Des technologies multiples
- Différents modèles d'architectures
- Différents « middlewares » ou « intergiciels »
- Une évolution vers l'intégration de l'hétérogène
- Le nouveau modèle : SOA

Prérequis pour les architectures orientées services :

Quelques rappels TCP/IP

Adresse IP

- Tient sur 32 bits (IPv4)
  - un quadruplet de nombres de 8 bits chaque, séparés par des points.
  - Exple: 127.0.0.1
- ICANN –*Internet Corporation for Assigned Names and Numbers*
  - attribue les adresses IP du réseau public internet
- @IP 127.0.0.1
  - adresse de rebouclage (ang. loopback)
  - désigne la machine locale (ang. localhost)
- 5 classes
  - Classe A: de 1.0.0.1 à 126.255.255.254
    - 127 réseaux de plus que 16 millions d'ordinateurs chaque
    - Réservées au grands réseaux
    - 1er octet réseau (NetID), 3 octets pour l'@ de l'hôte (hostID)

…Adresse IP

- Classe B: de 128.0.0.1 à 191.255.255.254
  - 16575 réseaux de plus que 6500 ordinateurs chaque
  - Réservées aux moyens réseaux
  - les deux 1ers octets réseau, 2 octets suivants pour l'@ de l'hôte
- Classe C: de 192.0.0.1 à 223.255.255.254
  - + que 2 millions de réseaux de 254 ordinateurs chaque
  - Réservées au petits réseaux d'entreprise
  - les trois 1ers octets réseau, 4éme octet pour l'@ de l'hôte
- Classe D: de 224.0.0.1 à 239.255.255.255
  - groupes multicast
- Classe E
  - réservées pour futur usage

IPv6

- Résout le problème de pénurie d'adresses IP
- @IP sur 128 bits

Port d’écoute

- 16-bits unsigned integer,
  - de 1 à 65535.
- Ports utilisateur >= 1024.
- Exples de ports réservés,
  - FTP, 21/TCP
  - Telnet, 23/TCP
  - SMTP, 25/TCP
  - Login, 513/TCP
  - HTTP, 80/TCP

Prérequis:

Les Flux en JAVA

Le package java.io

La gestion de fichiers en java

Sérialisation d'objets

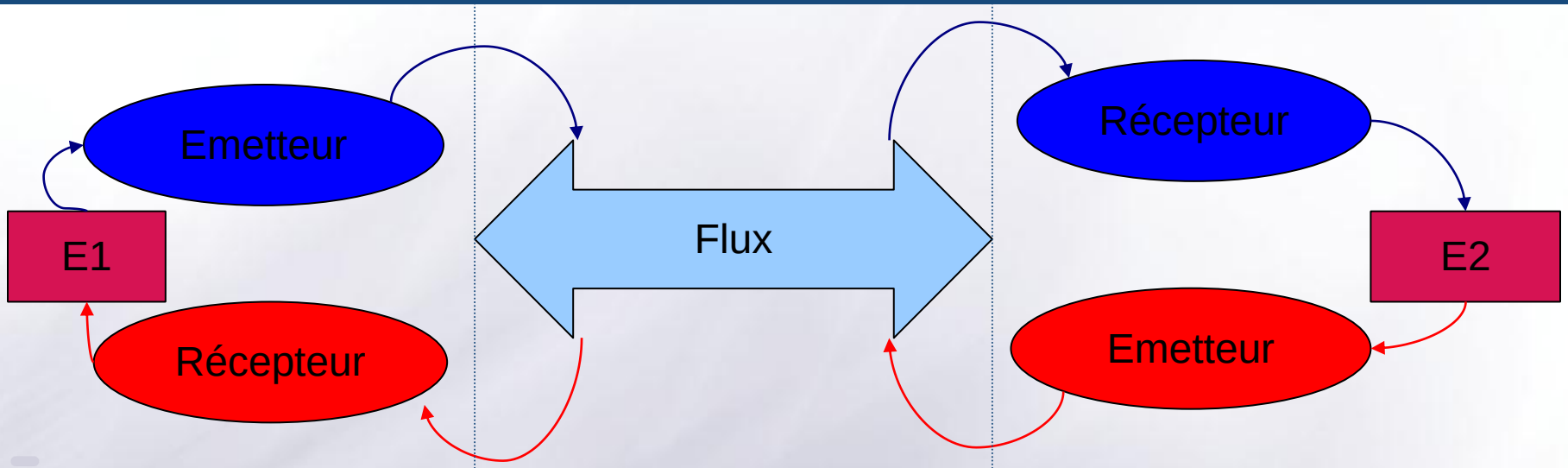
Les Flots/Flux/Streams

- Toutes les entrées/sorties en JAVA sont gérées par des flux (streams)
  - Lecture du clavier
  - Affichage sur la console
  - Lecture/Ecriture dans un fichier
  - Echange de données réseaux avec les Sockets
- Stream: objet JAVA possédant la caractéristique de pouvoir envoyer ou recevoir des données

Flux / Flots / Stream

- Un flux est une serie d'informations envoyé sur un canal de communication
- C'est le paradigme utilisé dans le monde objet pour représenter une communication entre 2 entités qui ne sont pas des objets
- Un flux se comporte comme une rivière :
  - Il faut une source/émetteur
  - Et un recepteur/consommateur
    - Un flux est bidirectionnel, par contre les points d'accès (émetteur/consommateur) ne peuvent travailler que dans un seul sens

Circulation d'information sur un flux



- Les flux informatiques ne véhiculent que des octets
- Les émetteurs et les récepteurs se chargent de transformer les octets sous des formes plus intéressantes pour les Entités (data, buffer, crypto...)
- Emetteur et recepteur doivent se mettre d'accord sur le format des structures envoyées (notion de protocole d'accord)
- La bibliotèque java.io définit l'ensemble des transformations que l'ont peut désirer sur un ensemble d'octets arrivant dans le flux
- Il existe également des emetteurs et des récepteurs standards

Emetteurs et recepteur standards du système d'exploitation

- Il existe 3 flux standards pour un système d'exploitation
  - Les octets circulant entre une application (A) et l'écran (E) pour indiquer une information standard System.out
  - Les octets circulant entre une application (A) et l'écran (E) pour indiquer une information d'erreur System.err
  - Les octets circulant entre le clavier (C) et une application (A) System.in

- Exercice: Représentez sur un schéma les flux et les points d'entrée qui vous interesserait en tant que programmeur

Exemple de saisie avec System.in

- **Voici le code qui lit l'entrée du clavier et qui envoie le caractère sur la sortie standard (affichage du code ASCII du caractère)**

```
import java.io.IOException;

public class MainClass {

public static void main(String[] args)  throws IOException{
        System.out.println("Entrez un charctere");
            int inChar = System.in.read();
        System.out.print("Vous avez saisie: "+inChar);
                    }

}
```

Intanciation des émetteurs / récepteurs

- Il existe des entités classiques pouvant être la source ou la destination d'octets

  - Le fichier : Il permet de stocker des octets
    - Classe: java.io.File (nommage, droits d'accès et propriétés d'un fichier ou d'un répertoire)
    - File f=new File("/tmp/toto");
  - La socket réseau : elle permet d'envoyer des octets à une autre machine
    - Classe: java.net.Socket (point d'entrée pour une connexion TCP/IP entre deux machines)
    - Socket s=new Socket("www.ENIS.rnu.tn", 80)

Instanciation des flux de communication

- Après la création de la source ou destination, il est nécessaire de construire des objets d'accès au flux pour pouvoir échanger les données

- Les accès sur les points d'entrés ne sont pas toujours homogènes :

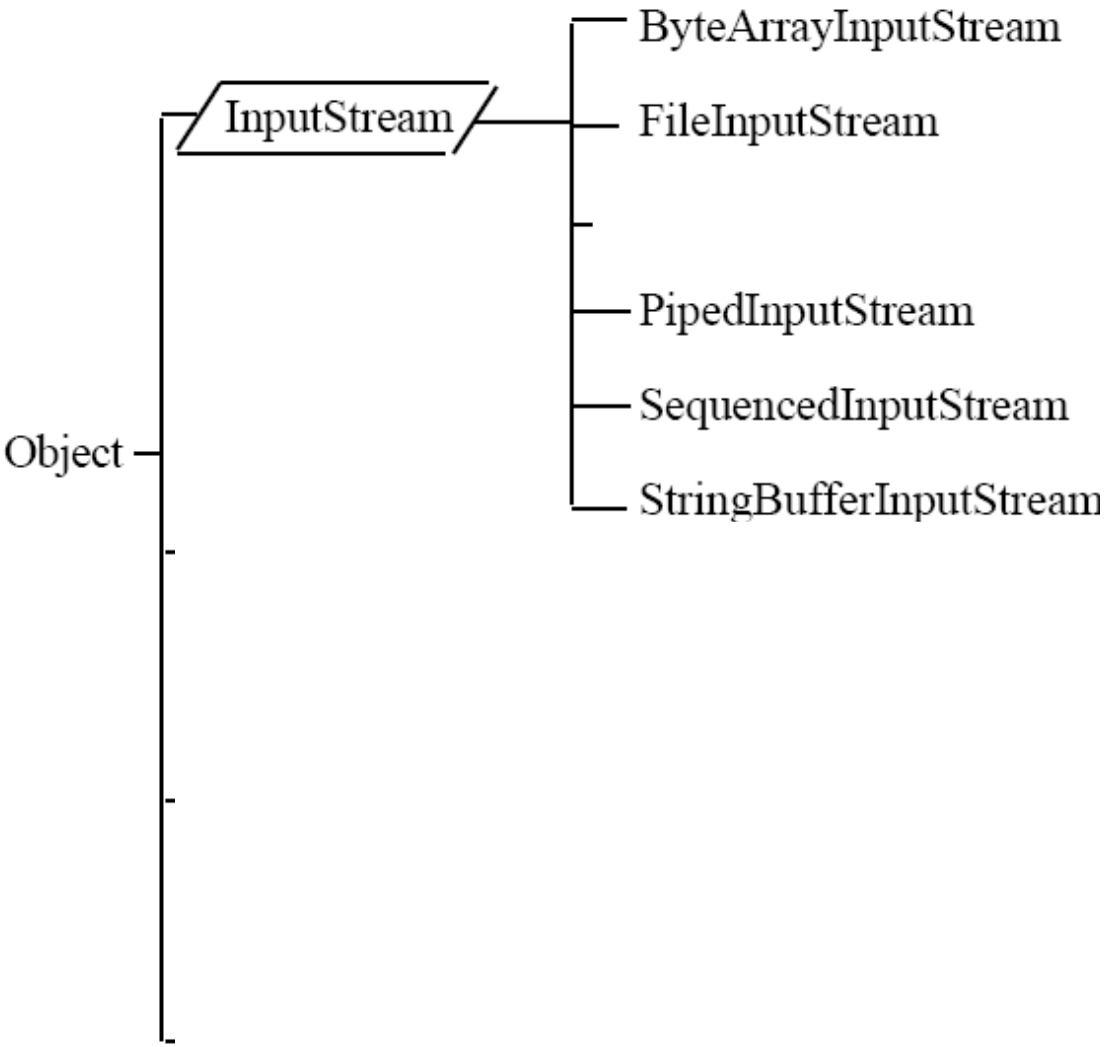
```
File f=new File("/tmp/toto");
FileInputStream fin=new FileInputStream(f);
int i = fin.read();

Socket s=new Socket("www.ENIS.rnu.tn", 80);
InputStream sin=s.getInputStream();
int i = sin.read();

private void lecture (InputStream in){
  int i=in.read();
  while(i!=-1) {  System.out.println(i); i=in.read();   }
}
```

- Que peut ont dire des classes FileInputStream, et InputStream ? Quelles sont les fonctions que l'ont peut appliquer sur un flux d'entrée ?

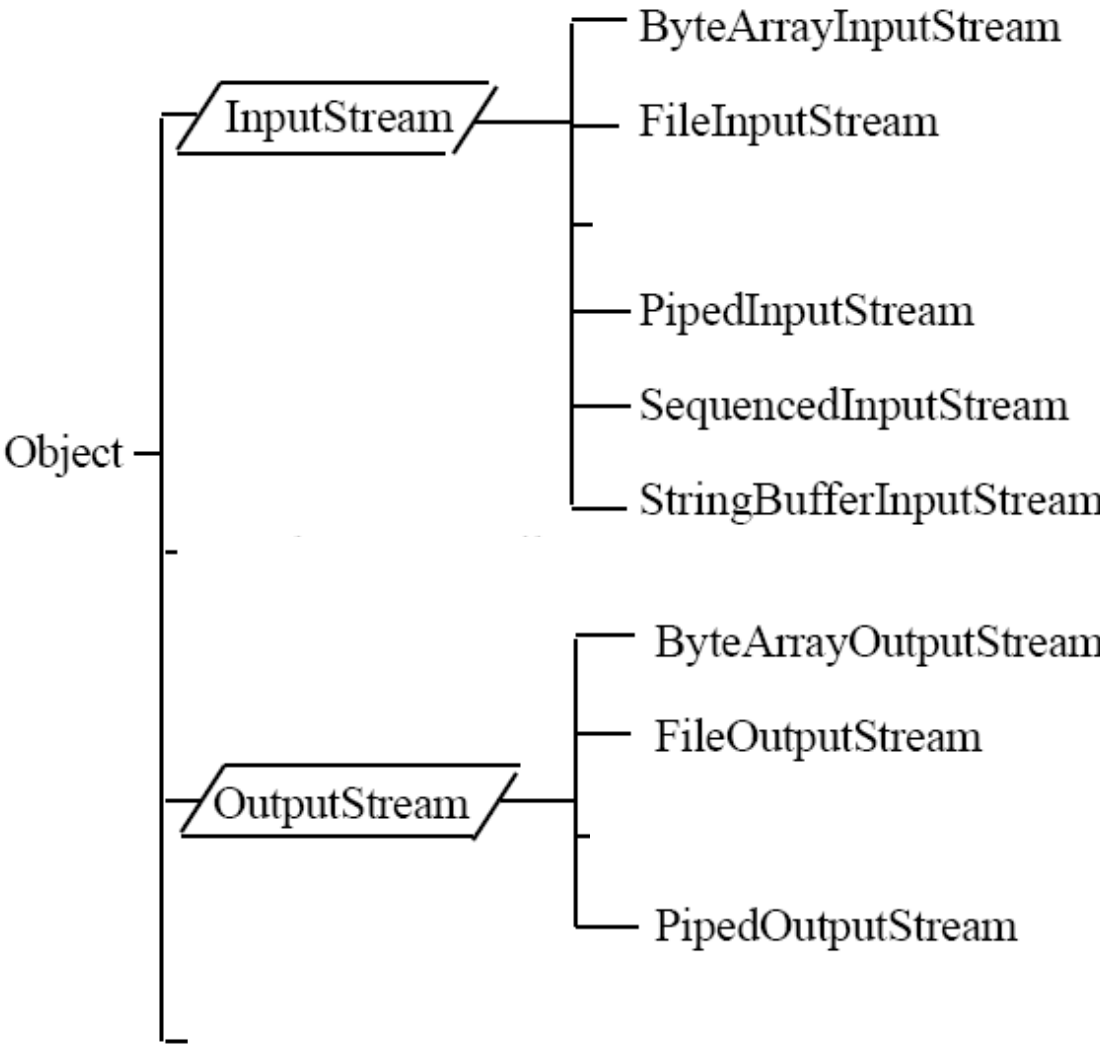
java.io: arborescence de classes



API : InputStream

- **int available()**
  - Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
- **void close()**
  - Closes this input stream and releases any system resources associated with the stream.
- **abstract int read()**
  - Reads the next byte of data from the input stream.
- **int read(byte[] b)**
  - Reads some number of bytes from the input stream and stores them into the buffer array b.
- **int read(byte[] b, int off, int len)**
  - Reads up to len bytes of data from the input stream into an array of bytes.

java.io: arborescence de classes



Exercice: copie de fichiers

- Ecrire une classe Copy qui réalise la copie d'un fichier dans un autre.

- Exemple d'utilisation:

java Copy sourceFile.txt destFile.txt

Que se passe-t'il si ?

- Que doit faire une entité lorsque qu'elle veut envoyer un long, double… ?
- Que doit faire une entité lorsqu’elle lit le byte, celui-ci n'est pas encore disponible ?

==> Il faut “décorer” nos flux

Décoration de flux

- La décoration est l'art de traiter une série d'octets avant de les transmettre à l'application demandeuse.
- Il existe 4 familles de décorations
  - Data : permet de convertir les bytes en structure primitive (long, boolean, double...)
  - Character : permet de convertir les bytes en texte
  - Object : permet de convertir les bytes en structure objet
  - Service : buffer, crypto, gzip permet d'appliquer des fonctions spécifiques sur la lecture
- Instanciation de décorateur:

```
DataInputStream dis=new DataInputStream((new
  FileInputStream("/tmp/toto");
```

Flux de données: Data(Input|Output)Stream

- **Ces classes permettent de lire et d'écrire des types primitifs sur des flux.**

```
FileOutputStream fos = new FileOutputStream("source.txt");
DataOutputStream dos = new DataOutputStream(fos);

dos.writeInt(123);
dos.writeDouble(123.456);
dos.writeChars("Une chaine");


FileInputStream fis = new FileInputStream("source.txt");
DataInputStream dis = new DataInputStream(fis);

int i    = dis.readInt();
double d = dis.readDouble();
String s = dis.readLine();
```

Flux de lecture / écriture avec buffer

- Cette classe permet de renvoyer, sur le flux de sortie *theOutput*, la chaîne de caractère lue sur le flux d'entrée *theInput*.

```
private void echoServer(InputStream theInput, OutputStream theOutput){
 // Un BufferedReader permet de lire par ligne.
        BufferedReader plec = new BufferedReader(
                                  new InputStreamReader(theInput));
        // Un PrintWriter possède toutes les opérations print classiques.
        PrintWriter pred = new PrintWriter(   new BufferedWriter(
                                new OutputStreamWriter(theOutput)),   true);
  while (true)
  {
  String str = plec.readLine(); // lecture du message
  if (str.equals("END")) break;
  System.out.println("ECHO = " + str); // trace locale
  pred.println(str); // renvoi d'un écho
  }
}
```

Flux de charactères

Ajout de la notion de flux de caractères (charater Streams)

- Les InputStream et OutputStream se basent sur la lecture et écriture d'octets (bytes)

- Les caractères sont codés sur 2 octets

→ Utilisations de deux classes (writer et reader) à la place de InputStream et OutputStream

→ Ecriture facile des caractères UNICODE (avec accents par exemple)

→ Pour lire et écrire directement des caractères unicode dans un fichier, il est conseillé d'utiliser FileReader et FileWriter à la place de FileInputStream et FileOutputStream

Flux d’objets: Sérialisation

- La sérialisation (Serialization) est le processus de transformation d'un objet dans un flux d'octets, et la déserialisation le processus inverse.
- La sérialisation permet à un objet d'être facilement sauvé sur un fichier ou transféré sur le réseau
- Les classes doivent implanter l'interface Serializable et éventuellement surcharger les méthodes readObject() et writeObject()

Flux d'objets :Syntaxe

- La classe de l'objet qu'on veut sérialiser doit implémenter l'interface java.io.Serializable

- Généralement, il n'y a pas de méthodes à implémenter sauf dans le cas où le programmeur veut définir des méthodes de sérialisation ou désérialisation spécifiques à l'aide de:

```
-private void writeObject (java.io.ObjectOutputStream
  out) throws IOException
-private void readObject(java.io.ObjectInputStream in)
  throws IOException, ClassNotFoundException;
```

Exemple de sérialisation d'un objet dans un fichier

```
import java.util.*;
import java.io.*
public class EcritObjPers {
    Date d = new Date();
    FileOutputStream f;
    ObjectOutputStream s;
    try {
        f = new FileOutputStream(« date.ser »);
        s = new ObjectOutputStream(f);
        s.writeObject(d);
        s.close();
    } catch (IOException e) {}
}
}
```

Exemple de désérialisation d'un objet à partir d'un fichier

```
import java.util.*;
import java.io.*
public class LiitObjPers {
    Date d = null;
    FileInputStream f;
    ObjectInputStream s;
    try { f = new FileInputStream(« date.ser »);
          s = new ObjectInputStream(f);
          d=(date)s.readObject();
          s.close();
          System.out.println(« date : »+d);
    } catch (IOException e) {}
}}
```

Exercice: Ecriture de logs

- **Ecrire un utilitaire de Gestion de logs « WriteLog ». Chaque log représente une ligne dans un fichier texte. Sur chaque ligne on trouve la date d'écriture, l'identifiant de son écrivain et son commentaire. Utilisez une classe Entry ayant pour attributs: la date, l'identifiant de l'écrivain et son commentaire. Cette classe doit définir la méthode de sérialisation de ses attributs dans un flux de sortie OutputStream ouvert.**

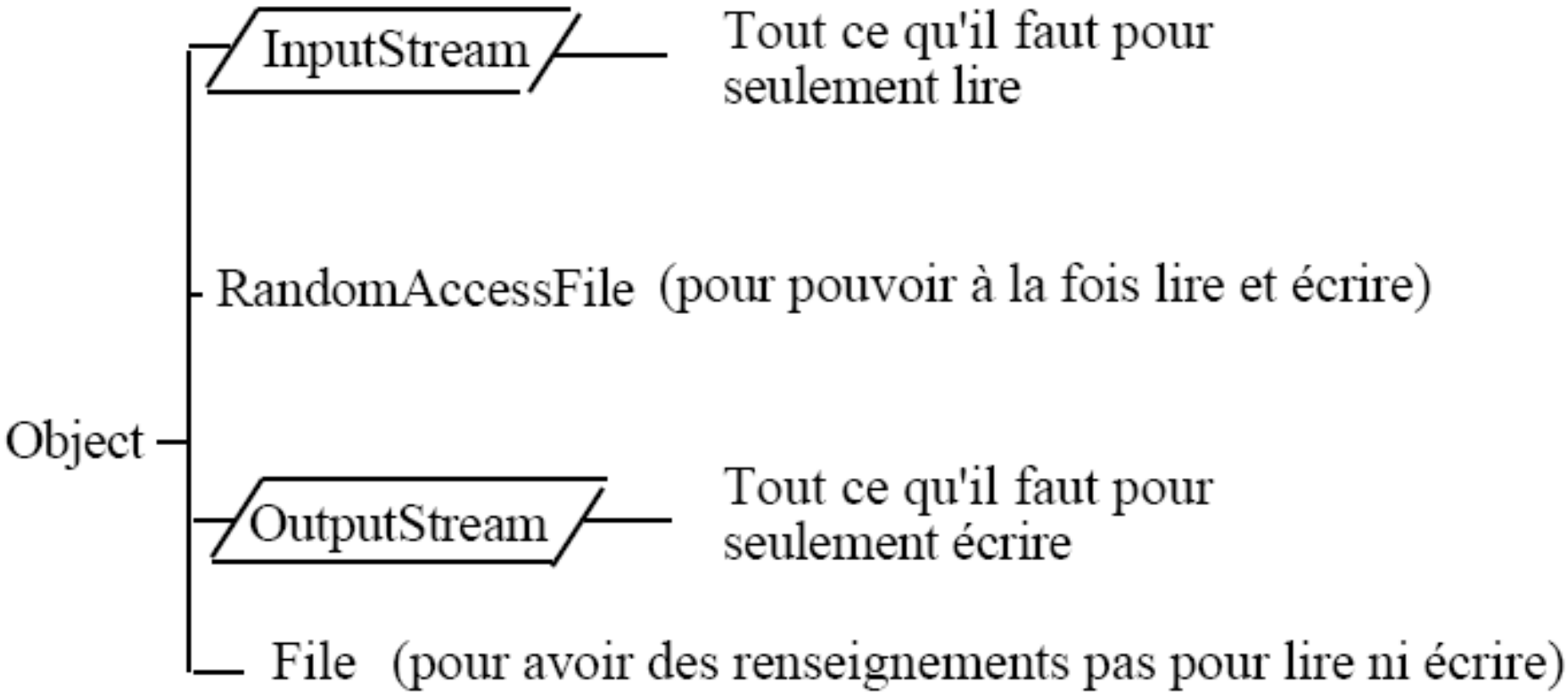
```
// dans la classe WriteLog
public void writeEntry(Entry e)
{

}
```

Exercice: Lecture de logs

- Ecrire un utilitaire « ScanLog » de lecture des logs enregistrés dans l'exercice précédent. Cet utilitaire doit permettre un parcours sélectif en utilisant les options suivantes:
  - *-after d* : pour afficher les enregistrements écrits après la date *d*
  - *-before d* : pour afficher les enregistrements écrits avant la date *d*
  - *-user u* : pour afficher que les enregistrements écrits par l'écrivain *u*

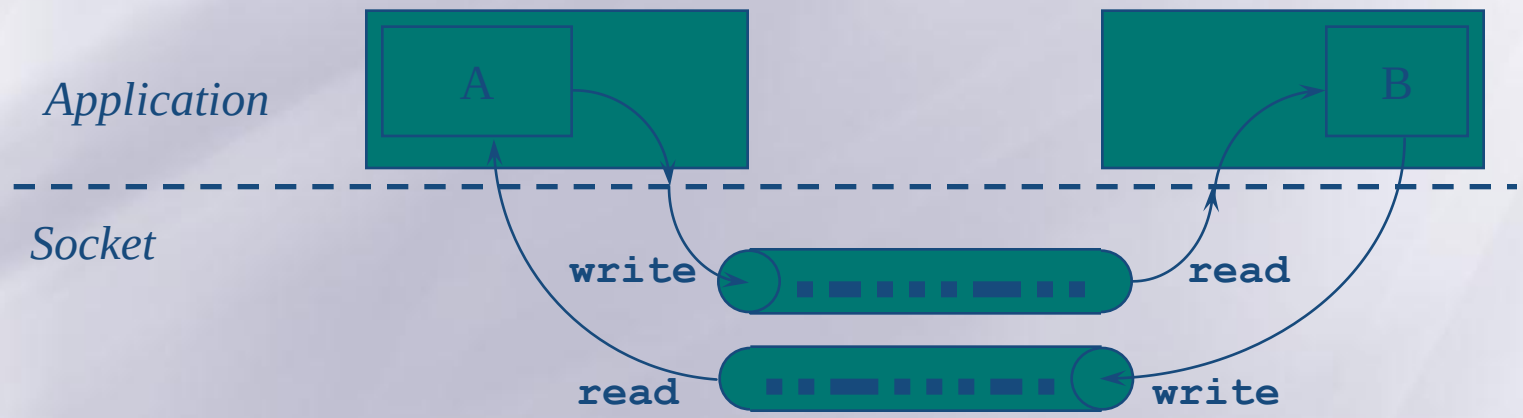
java.io: ce qu'il faut retenir



LES SOCKETS

Un Socket et un point de communication bidirectionnel entre applications

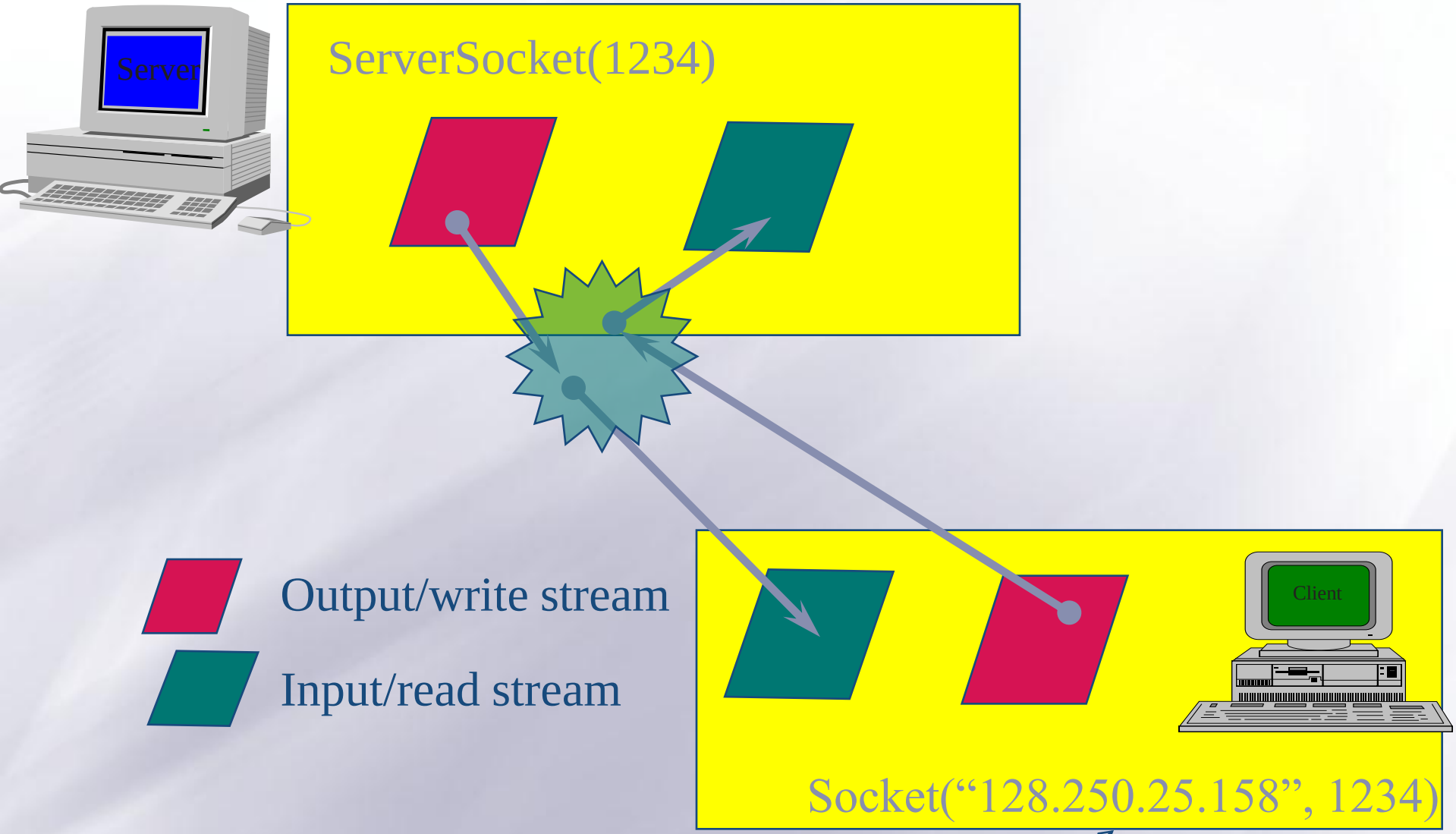
- Introduit dans Berkeley Unix 4.2 (BSD) en 1981
- Représentation point à point d'un flux de données
- Un programme peut
  - Écrire des données
  - Lire des données



Définitions

- Un Socket et un point de communication bidirectionnelle entre applications
- Un socket est associé à un port de communication pour différencier les applications réseau sur une même machine
- L'API java fourni un package java.net pour l'utilisation des sockets
  - java.net.Socket pour l'implémentation coté client
  - java.net.ServerSocket pour l'implémentation coté serveur

Schéma de communication des sockets



Java.net.InetAddress : nommage

- Cette classe permet de manipuler les adresses Internet.
- Pour obtenir une instance de cette classe :
- public static InetAddress getLocalHost()
  throws UnknownHostExeception

Elle retourne un objet contenant l'adresse Internet de la machine locale.

- public static synchronized InetAddress getByName(String host_name) throws UnknownHostExeception

Elle retourne un objet contenant l'adresse Internet de la machine dont le nom est passé en paramètre.

- Les méthodes principales de cette classe sont :
- public String getHostName ()

Retourne le nom de la machine dont l'adresse est stockée dans l'objet.

- public String toString ()

Retourne une chaîne de caractères qui liste le nom de la machine et son adresse.

- Exemple :

```
InetAddress adresseLocale = InetAdress.getLocalHost ();
InetAddress adresseServeur = InetAdress.getByName("www.google.fr");
System.out.println(adresseServeur);
```

Fonctions d'un sockets

- 4 fonctions principales:
  - Se connecter à une machine distante
  - Envoyer des données
  - Recevoir des données
  - Fermer la connexion
- Un socket ne peut se connecter qu'à une seule machine
- Un socket ne peut pas être se reconnecter après la fermeture de connexion

Instanciation d'un socket

- La création d'un socket se fait à l'aide de l'un de ces constructeurs
  - public Socket(String host, int port) throws UnknownHostException, IOException
  - public Socket(InetAddress address, int port) throws IOException
  - public Socket(String host, int port, InetAddress localAddr, int localPort) throws IOException
  - public Socket(InetAddress address, int port, InetAddress localAddr, int localPort) throws IOException
- Ces constructeurs créent un objet Socket et ouvre une connexion réseau avec la machine distante 'host'
- Tous ces constructeurs lèvent une exception en cas d'impossibilité de connexion

Connexion d'un socket

- Pour qu'un socket puisse se connecter à une machine distante, cette dernière doit écouter les requêtes des clients sur le port spécifié
- Il faut définir au minimum l'adresse de la machine distante et son port d'écoute pour pouvoir se connecter
- On peut utiliser les constructeurs pour déterminer les ports d'écoute d'une machine

Exercice

- Ecrire un programme Java qui affiche sur l'écran les ports d'écoute d'une machine distante donnée (en ligne de commandes)

- Rappel: les ports TCP d'une machine peuvent aller de 1 à 65535

Envoi et réception de données

- Les données sont envoyées et reçues à travers les flux d'entrée/sortie

```
public InputStream getInputStream() throws
  IOException
public OutputStream getOutputStream() throws
  IOException
```

- Il y a aussi une méthode qui permet de fermer le flux d'envoi et de réception de données

```
public synchronized void close() throws
  IOException
```

Réception de données

- La méthode getInputStream() permet d'obtenir le flux d'entrée pour la réception des données
- On peut utiliser les méthodes classiques de InpuStream pour récupérer les données entrantes
- Généralement, on utilise les décorateurs pour faciliter la manipulation des données reçues

```
DataInputStream input;
try {
        input = new DataInputStream(mySocket.getInputStream());
        String textFromServer = input.readLine();
      System.out.println(textFromServer);
   }
catch (IOException e) {System.out.println(e);}
```

Exemple d'un Socket coté client

- Ce programme permet de se connecter à un serveur qui fourni le temps sur le port TCP 13

```
try {
   Socket s = new Socket("metalab.unc.edu", 13);
   InputStream in = s.getInputStream();
   InputStreamReader isr = new
  InputStreamReader(in);
   BufferedReader br = new BufferedReader(isr);
   String theTime = br.readLine();
   System.out.println(theTime);
}
catch (IOException e) {
   return (new Date()).toString();
}
```

Envoi de données

- La méthode getOutputStream() permet d'obtenir le flux de sortie pour l'envoi des données
- On peut utiliser les méthodes classiques de InpuStream pour envoyer des données à travers les sockets
- Généralement, on utilise les décorateurs pour faciliter l'envoi de données

```
PrintStream output;
try {
        output = new PrintStream(mySocket.getOutputStream();
        output.print("Coucou serveur, je suis le client ☺");
}
catch (IOException e) {System.out.println(e);}
```

Exercice

- Ecrire un programme java qui affiche le contenu d'une page web distante en utilisant son URL

- Quelques indications
  - Instanciation de l'url: URL u = new URL(args[0]);
  - u.getPort() pour récupérer le port TCP
  - u.getProtocol() pour récupérer le protocole de l'url (on accepte que http)
  - u.getHost() pour récupérer l'hôte distant
  - u.getFile() pour récupérer le nom de la page demandée

- Requête HTTP
  - "GET " + u.getFile() + " HTTP/1.0\r\n"
  - "Accept: text/plain, text/html, text/*\r\n"
  - "\r\n"

Quelques autres méthodes des Sockets

Informations générales sur la connexion :

```
public InetAddress getInetAddress()
public InetAddress getLocalAddress()
public int getPort()
public int getLocalPort()
```

La méthode usuelle toString() :

```
public String toString()
```

La partie Serveur

- Dans une communication client/serveur, on a le programme client qui initialise la connexion d'un coté et la partie serveur qui répond aux requettes des clients

- Les clients et les serveurs sont connectés par des sockets

- Contrairement au client qui se connecte à un poste distant, le serveur attend les connections d'autres postes pour répondre à leurs requêtes

Les sockets Serveur

- Un socket serveur est attaché à un port particulier sur la machine locale
- Ensuite, il écoute sur ce port pour attendre les requête des clients
- Quand un serveur détecte une tentative de connexion, il lance une méthode *accept()*. Ceci crée automatiquement une socket entre le client et le serveur qui représente un canal de communication entre eux

Files d’attente

- Les connexions entrantes sont mises en file d’attente jusqu’à ce que le serveur puisse les accepter
- En général, la taille de la file d’attente est entre 5 et 50
- Si la file d’attente est pleine, le socket serveur refuse les connexions supplémentaires

La classe java.net.ServerSocket

- La classe `java.net.ServerSocket` représente un socket serveur
- Un socket serveur est affecté à un port TCP dès son instanciation. Ensuite, il suffit d'appeler sa méthode `accept()` pour écouter les connexions entrantes
- `accept()` bloque le programme serveur jusqu'à ce qu'une connexion est détectée. Dans ce cas, la méthode accept() retourne un objet socket (java.net.Socket)

Instanciation

- Il y a 3 constructeurs principaux qui permettent:
  - l'affectation de la socket à un port TCP
  - La longueur de la file d'attente
  - L'adresse IP sur laquelle la socket sera affectée

```
public ServerSocket(int port) throws IOException
public ServerSocket(int port, int backlog) throws
  IOException
public ServerSocket(int port, int backlog,
  InetAddress bindAddr) throws IOException
```

Instanciation classique :

```
try {
   ServerSocket serverSocket = new ServerSocket(80);
 }
 catch (IOException e) {
   System.err.println(e);
 }
```

Affectation aux ports

- Quand un `ServerSocket` est crée, il essai de s'attacher au port défini dans le constructeur
- S'il existe un autre socket serveur qui écoute déja sur le port spécifé, l'exception `java.net.BindException`, (sous classe de `IOException`) est levée
- Aucun processus serveur ne peut être affecté à un port déja utilisé par un autre processus serveur qu'il soit développé en java ou non.

Méthodes principales d'un server socket

- Les méthodes principales d'un server socket sont :
  - public Socket accept() throws IOException
  - public void close() throws IOException
- Un socket serveur ne peut plus être réinitalisé une fois qu'il est fermé

Lecture de données à l'aide d'un socket serveur

`ServerSocket` utilise la méthode `accept()` pour se connecter à un client

public Socket accept() throws IOException

- Il n'y a pas des méthodes comme `getInputStream()` ou `getOutputStream()` pour un `ServerSocket`
- `accept()` retourne un objet de type java.net.Socket avec lequel on peut obtenir les flux d'entrée et de sortie à l'aide des méthodes `getInputStream()` et `getOutputStream()`

Exemple d'un socket serveur

```
try {
  ServerSocket ss = new ServerSocket(2345);
  Socket s = ss.accept();
  PrintWriter pw = new
   PrintWriter(s.getOutputStream());
  pw.print("Hello There!\r\n");
  pw.print("Really, I have nothing to say…\r\n");
  pw.print("Goodbye now. \r\n");
  s.close();
}
catch (IOException e) {
  System.err.println(e);
}
```

Un autre exemple

```
try {
  ServerSocket ss = new ServerSocket(port);
  while (true) {
   try {
    Socket s = ss.accept();
    PrintWriter pw = new PrintWriter(s.getOutputStream());
    pw.print("Hello " + s.getInetAddress() + " on port "
     + s.getPort() + "\r\n");
    pw.print("This is " + s.getLocalAddress() + " on port "
     + s.getLocalPort() + "\r\n");
    pw.flush();
    s.close();
   }
   catch (IOException e) {}
  }
}
catch (IOException e) { System.err.println(e); }
```

Techniques recommandées d'un socket serveur

- Il est recommandé d'ajouter du multithreading dans les serveurs

- Il faut avoir une boucle continue qui traite les requêtes des clients

- Au lieu de traiter les requêtes directement, chaque socket devrait être passé à un Thread

```
while (true) {
 try {
  Socket s = ss.accept();
  ThreadedEchoServer   tes = new ThreadedEchoServer(s) ;
  tes.start();
 }
catch (IOException e) {System.out.println("problème de connexion");}
```

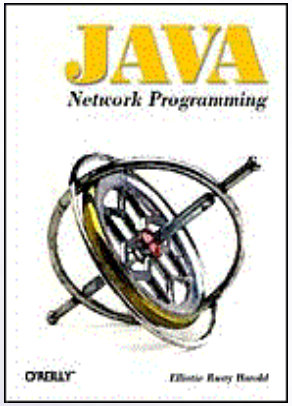
Exercice

- Réaliser deux versions d'un serveur d'echo:
  - une sans les Thread
  - une en utilisant les Thread
- Développer un client pour tester les serveurs d'echo
- Développer un serveur HTTP simple qui renvoi le contenu du fichier demandé dans une requête *Get*

Pour apprendre plus sur les sockets

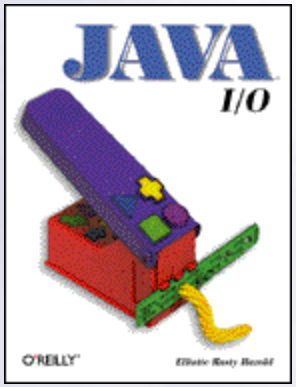
Java Network Programming

- O'Reilly & Associates, 1997
- ISBN 1-56592-227-1



Java I/O

- O'Reilly & Associates, 1999
- ISBN 1-56592-485-1



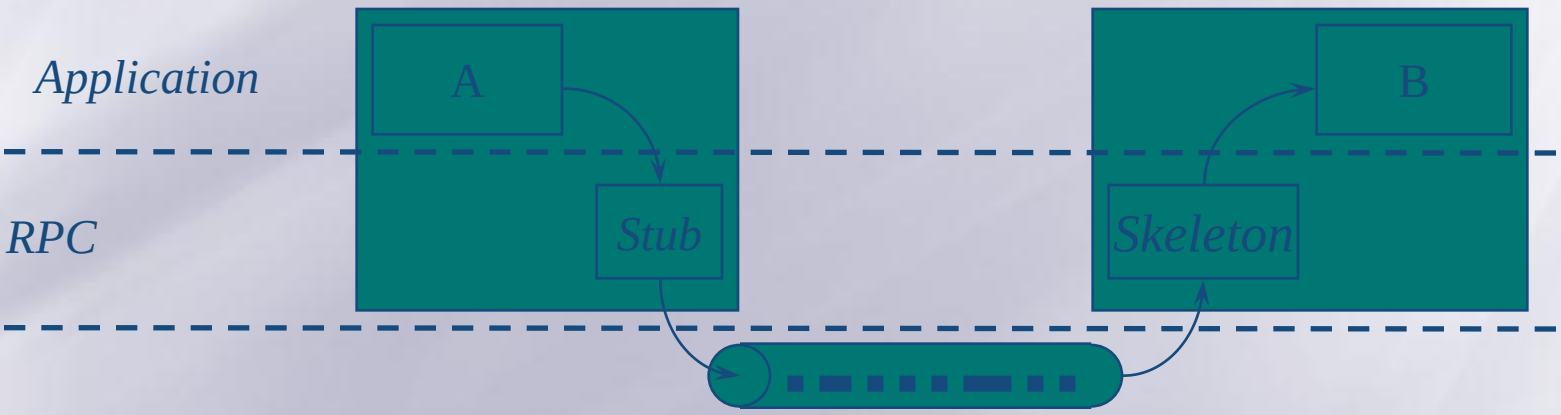
Deuxième implémentation de la notion de « service »

RPC

Remote Procedure Call

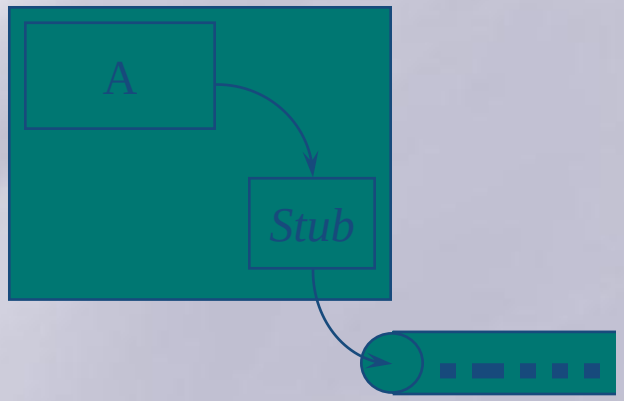
Remote Procedure Calls (RPC)

- Permet de donner l'impression d'un appel local pour un appel distant
- Code généré
  - Du côté appelant: stubs
  - Du côté appelé: skeletons
- Passage de paramètre par valeur
- RPC est fait pour le langage C
  - Toujours utilisé aujourd'hui, par exemple pour NFS



Remote Procedure Calls

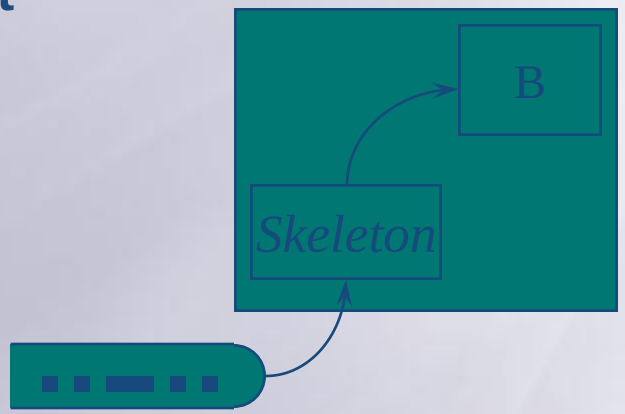
- **Le rôle du Stub est de:**
  - **Transformer les structures de données passées en paramètres de la procédure en flux d'octets (*marshalling*)**
  - **Écrire ces données dans la socket**
  - **Attendre jusqu'à ce que des données soient disponibles en lecture**
  - **Lis les données et les converties en structure (*unmarshalling*)**
  - **Retourne le résultat à l'appelant comme résultat de son appel de méthode**



Remote Procedure Calls

Le Skeleton doit

- Attendre que des données soient disponibles en lecture
- Lire un flux d ' octets et construire la structure correspondante
- Appeler la méthode correspondante sur B avec les paramètres
- Transformer le résultat en flux d' octets
- Envoyer ce résultat à l' appelant



Remote Procedure Calls

Limitations de RPC

- Peu de transparence
  - Les machines distantes sont trouvées avec leur adresse IP et le numéro de port
- Pas de nommage symbolique
  - Impossible de trouver un service suivant son nom ou son type
- Pas de chargement de code dynamique
  - Le stub doit être pré installé chez l'appelant, le skeleton chez l'appelé
  - Le code pour le *marshalling/unmarshalling* est spécifique et doit être présent des 2 côtés
- Aucun mécanisme de tolérance aux pannes ou de gestion des erreurs

Java-RMI

- RMI signifie *Remote Method Invocation*
- Introduit dés JDK 1.1
- Partie intégrante du cœur de Java
- RMI = RPC en Java + chargement dynamique de code
- Même notions de stubs et skeletons
- Fonctionne avec l'API de sérialisation (utilisée également pour la persistance)
- Possibilité de faire interagir RMI avec CORBA et DCOM

RMI: Fonctionnalités

- Masque pratiquement tout le réseau à l'application
- Très similaire à RPC
- Utilisation de l'interface *Remote* Java
- Appels distants bloquants
  - L'appelant est bloqué jusqu'à ce que le résultat soit disponible
- Les références vers des objets « normaux »sont passées par copie profonde (*deep copy*) grâce à la sérialisation
- Les références vers des objets distants sont passées par référence

Implémenter un objet distant

- Un objet qui a des méthodes appelables à distance est appelé *objet distant*
- Les seules méthodes accessibles à distance seront celles spécifiées dans l'interface *Remote*
  - Écriture d'une interface spécifique à l'objet, étendant l'interface *java.rmi.Remote*
- Chaque méthode distante doit annoncer lever l'exception *java.rmi.RemoteException*
  - Sert à indiquer les problèmes liés à la distribution
- L'objet devra fournir une implémentation de ces méthodes

Implémenter un objet distant

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface MonInterfaceDistante extends Remote {
    public void echo() throws RemoteException;
}
```

- Cette interface indique que l'objet qui l'implémentera aura la méthode *echo()* appelable à distance

Implémenter un objet distant

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class MonObjetDistant extends UnicastRemoteObject     {
                                    implements MonInterfaceDistante

      public MonObjetDistant() throws RemoteException {}

      public void echo() throws RemoteException{
           System.out.println(« Echo »);
       }
}
```

- L'objet distant doit finalement implémenter les méthodes de l'interface
  - Hériter de *java.rmi.server.UnicastRemoteObject*
  - Et avoir un constructeur sans paramètre levant aussi l'exceptions

Utiliser un objet distant

- **Pour utiliser un objet distant il faut**
  - **Connaître son interface**
  - **Le trouver!**
  - **L'invoquer**
- **RMI fournit un service de nommage permettant de localiser un objet par son nom : le *registry***
  - **L'objet s'enregistre sous un nom « bien connu »**
  - **Les clients demandent une référence vers cet objet**

Utiliser un objet distant

```
import java.rmi.RemoteException;

public class Client {
      public static void main(String args) {
         MonInterfaceDistante mod = ….. // du code pour trouver l'objet
          try {
           mod.echo();
          } catch (RemoteException e) {
            e.printStackTrace();
         }
    }
}
```

- Une fois la référence obtenue, il suffit d'appeler la méthode dessus, en prenant garde aux éventuelles exceptions

Trouver un objet distant

- L'objet doit s'enregistrer dans le *registry*
  - Programme lancé auparavant sur la même machine que l'objet distant : *rmiregistry*
  - Utilise le port 1090 par defaut
  - Possibilité de le démarrer depuis l'application (*LocateRegistry*)
- Il agit comme un service d'annuaire
- Les noms ressemblent à des URLs
  - *protocole://machine:port/nom*
  - *Protocole, machine* et *port* sont optionnels
  - Objet toto sur la machine locale: *///toto*
- Les méthodes de gestion sont regroupées dans la classe *Naming*
- L'objet distant appelle *Naming.bind*
- Le client appel *Naming.lookup*

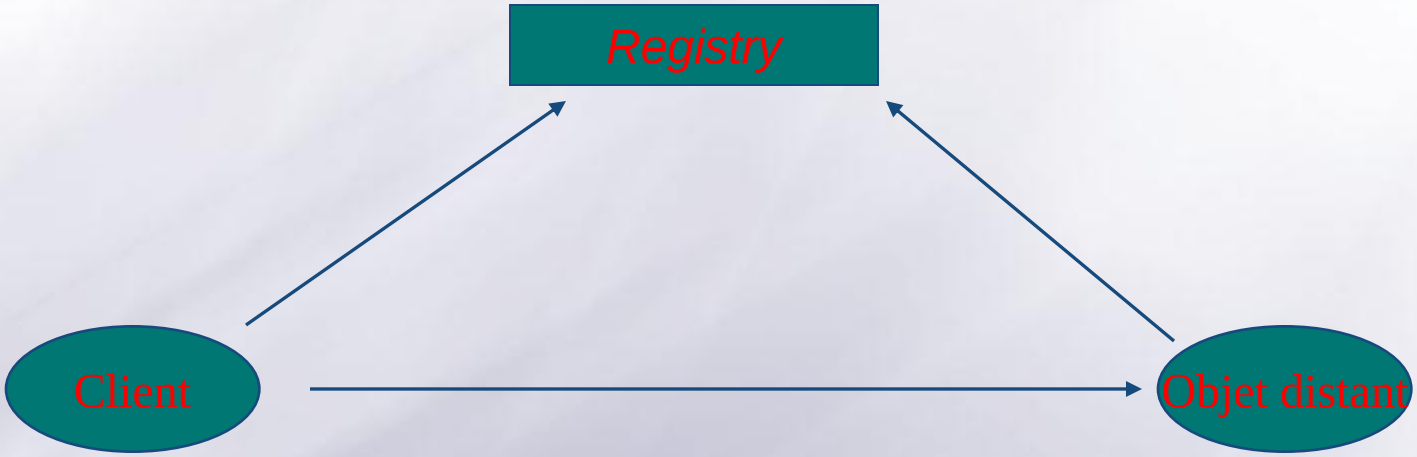
Générer les stubs et skeletons

- Une fois l'objet distant écrit, il est possible de générer les stubs et les skeletons
- Outils fournit dans Java: *rmic*
- Prends le nom complet de la classe distante (*package+nom*)
- Génère 2 fichiers (*_Stub* et *_Skel*)
- Ne met dans le stub que les méthodes spécifiées dans l'interface distante
- Possibilité de voir le code source avec l'option *–keep*

Développement RMI en résumé

1. Écrire l'interface distante
2. Écrire le code de l'objet distant
   1. Implémenter l'interface
   2. Ajouter le code pour le registry (en général dans le *main* ou le constructeur)
3. Compiler
4. Générer les stub et skeleton
5. Écrire le client
   1. Obtenir une référence vers l'objet distant
   2. Appeler les méthodes
6. Compiler
7. Exécuter
   1. Démarrer le serveur
   2. Démarrer le client
   3. Debugger ☺

Cycle de vie d'une application RMI



1. L'objet distant s'enregistre dans le *registry*
2. Le client demande une référence au registry
3. La référence sert ensuite pour appeler les méthodes

Exemple complet RMI : l'interface

```
// Definition de l'interface de l'objet distant
//
// Cette interface doit etre publique
// sinon les clients auront des pbs !!!
//

public interface Hello extends java.rmi.Remote
{
    String lireMessage() throws java.rmi.RemoteException;
}
```

Exemple complet RMI : l' objet distant

```
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

// On specifie dans la definition de la classe que celle-ci implante l'interface Hello et que ses
// instances seront des objets repartis

public class HelloServeur extends UnicastRemoteObject implements Hello
{
    // Ceci est l'implantation de la methode qui sera invoquee de facon (eventuellement) distante par les clients
    public String lireMessage() throws RemoteException
    {       return "hello" ;}

    // Le constructeur pour la classe
    public HelloServeur() throws RemoteException {}

}
```

Exemple complet RMI : Enregistrement de l'objet distant

```
// Methode principale
public static void main(String args[]) throws IOException
{
    // Creation et installation du security manager
    if (System.getSecurityManager() == null)
        System.setSecurityManager(new RMISecurityManager());
    try
    {   // Creation de l'objet qui va etre invoque par les clients
        HelloServeur MonServeur = new HelloServeur();
        Naming.rebind("//192.168.1.5:1099/HelloServeur", MonServeur);
        System.out.println("HelloServeur enregistre : " + nomService);
    }
    catch (RemoteException e)
    {
        System.out.println("HelloServeur err: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Exemple complet RMI : Enfin le client

```
import java.rmi.*;

// HelloClient : client qui va lire un message genere par le serveur
public class HelloClient
{
    // Methode principale
    public static void main (String args[])
    {
        try
        {
            String nomService = "//192.168.1.5:1099/HelloServeur";
            System.out.println (" Connexion au service : " + nomService);
            Hello obj = (Hello) Naming.lookup (nomService);
            // Finalement, on peut invoquer la methode de l'objet qui est heberge par le serveur
            System.out.println ("Le message est : " + obj.lireMessage ());
        }
        catch (Exception e)
        {
            System.out.println ("Hello exception: " + e.getMessage ());
        }
    }
}
```

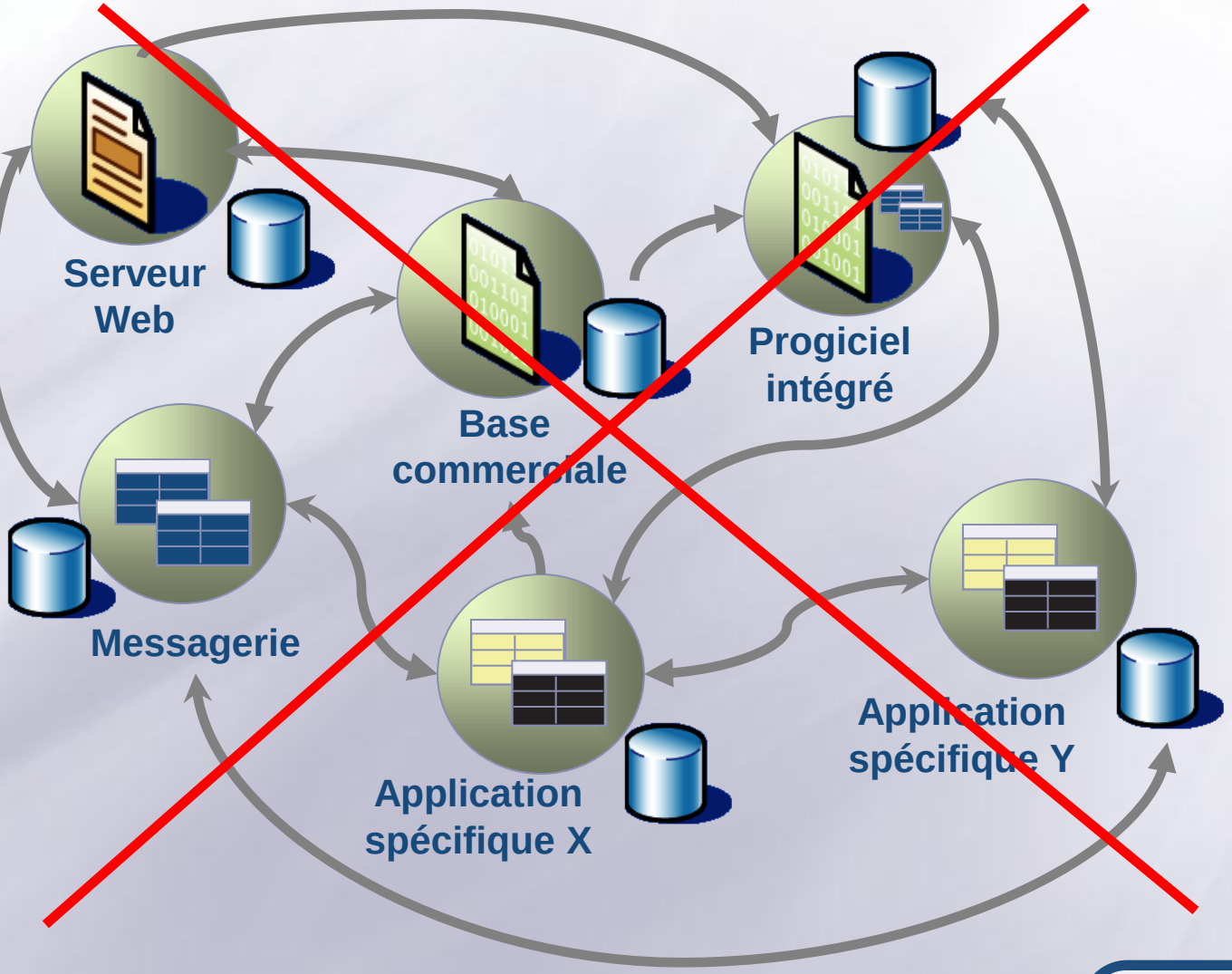
Exercice

- **Développer un annuaire téléphonique distant en RMI**
  - cet annuaire est une liste de paires <nom, Info>
  - Info est une classe qui contient deux attributs adresse et numéro de téléphone
  - l'annuaire offre les fonctions suivantes :
    - boolean inserer(nom, info) peut lever une exception de type *ExisteDeja*
    - boolean supprimer(nom) peut lever une exception de type *PasTrouve*
    - Info rechercher(nom) peut lever une exception de type *PasTrouve*
- **Développer un client de cet annuaire pour tester la connexion RMI**

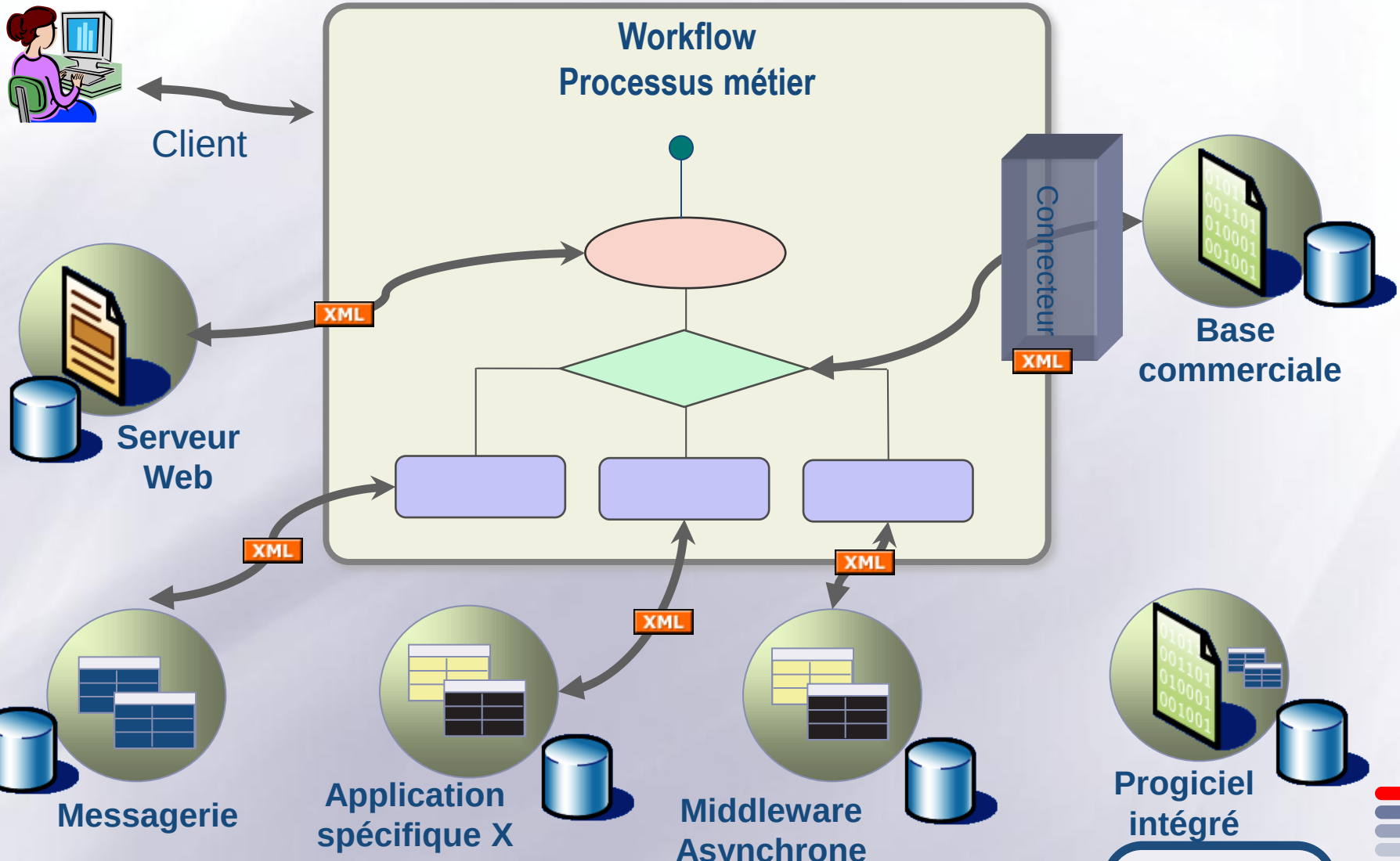
Les Architectures Orientées service

Tout devient « service »

Aujourd'hui : plat de spaghettis



Demain : Architecture urbanisée

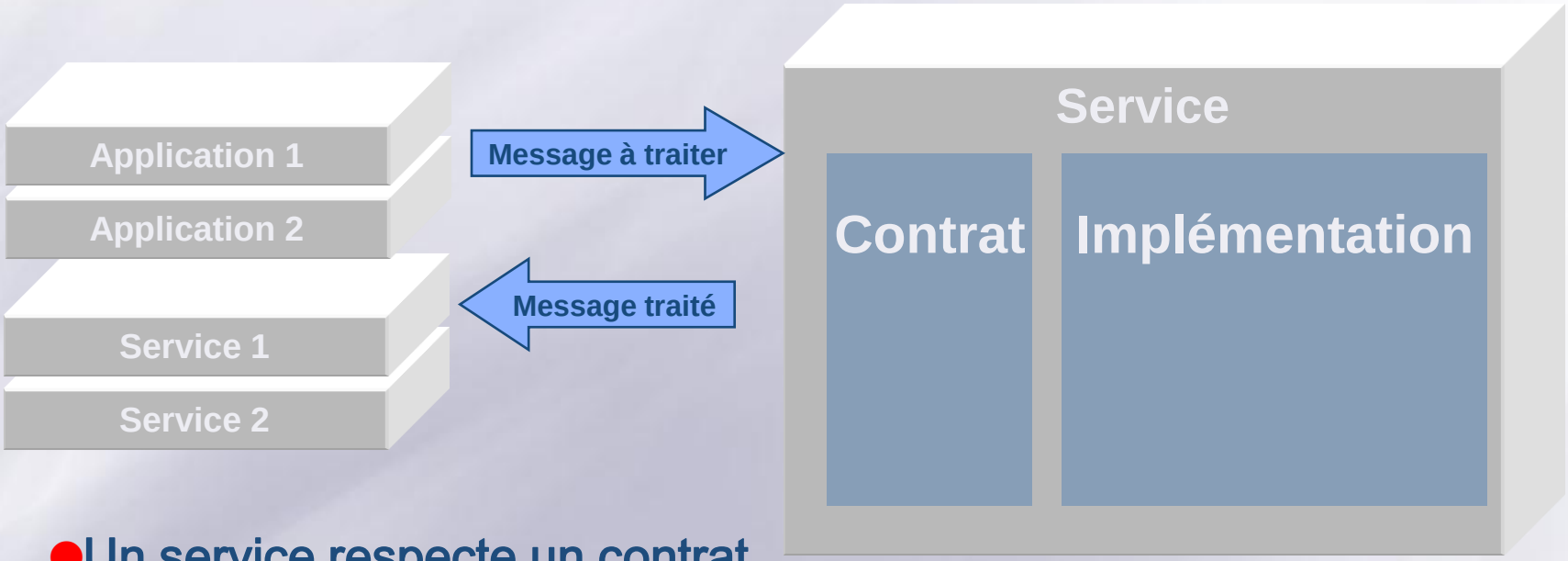


Qu'est-ce que SOA ?

- *« Une vision d'un système destinée à traiter toute application comme un fournisseur de services »*
- **Tout traitement applicatif est vu comme un service potentiellement utilisable par un client**
- **Le client est découplé de l'architecture technique du service qu'il invoque**
- **SOA véhicule des Messages et non des objets**
- **Pour adapter les technologies hétérogènes, on fait appel à des Connecteurs/Adaptateurs**

Noyau des architectures orientées services

- Un service est une fonction qui reçoit des messages et qui les restitue après traitement.



- Un service respecte un contrat
- Un service est réutilisable
- Un service est sans état

Architecture orientée services

- Une architecture orientée services est une infrastructure permettant de diffuser des services.

- SOA – Service Oriented Architecture
- WSOA – Web Services Oriented Architecture

- SOA n'est pas un concept neuf. Les sites centraux utilisaient déjà la notion de services.

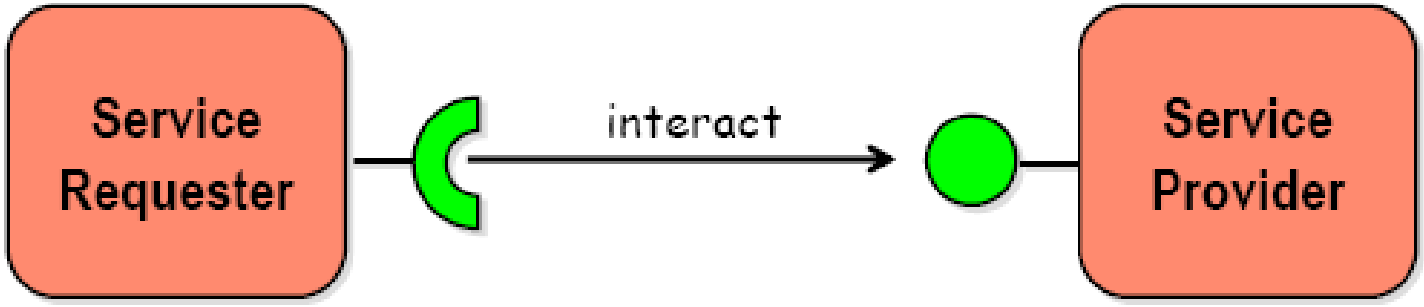
SOA ou Web Services

- La notion de « service » est le concept important. Les Services Web sont juste un moyen de les implémenter. Peut-être le meilleur
- Une bonne SOA est une histoire de conception pas de technologie.
- Il est très facile de construire des services web médiocres. Beaucoup le feront.
- La technologie Services Web ne fait pas votre architecture.
- Un bon service doit être « abstrait » : il n'est pas lié à une implémentation.
- Les services web forment un très bon ensemble de spécifications pour construire une SOA : indépendance, sécurité, transaction, orchestration,…

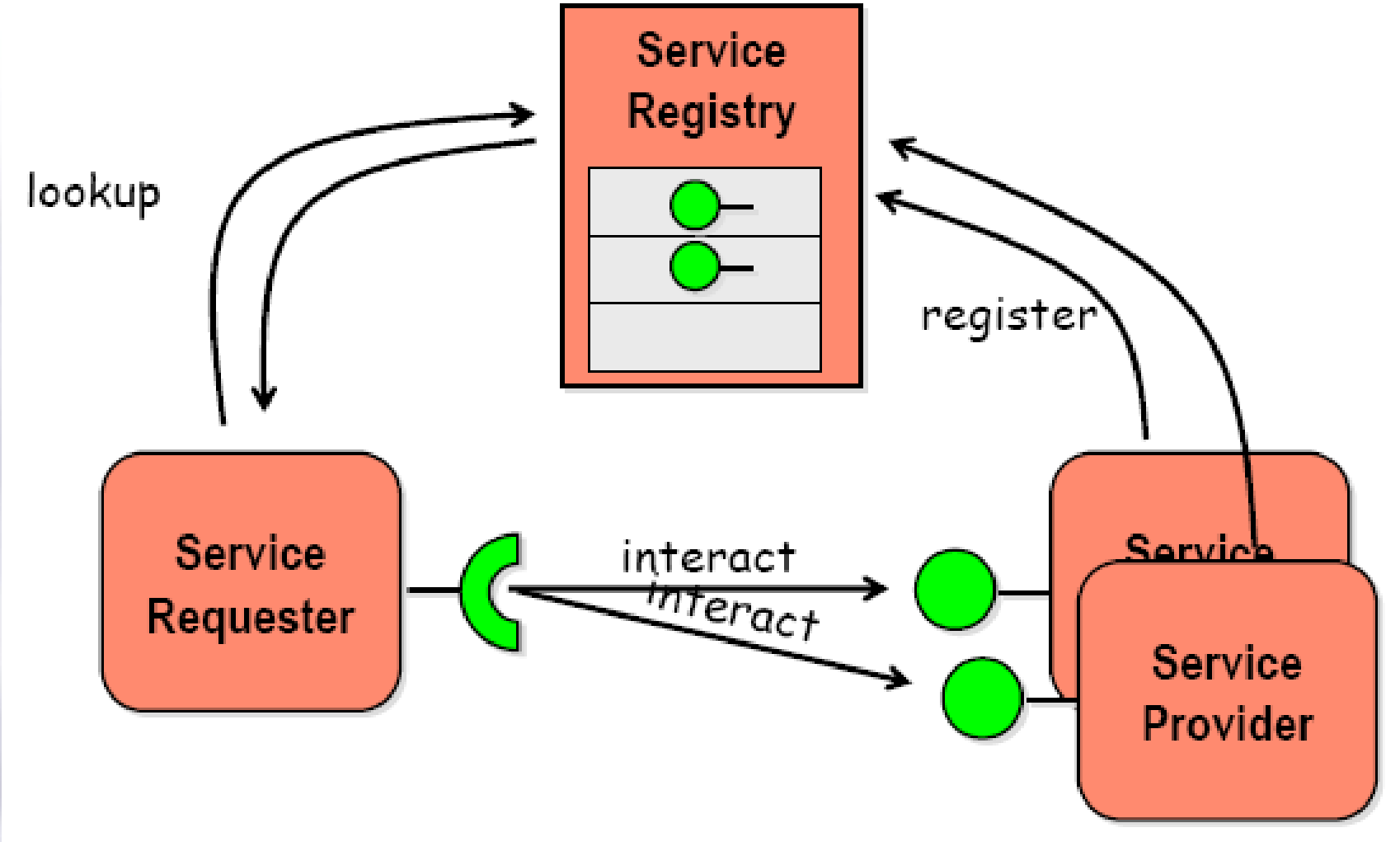
Services Web standars

Rappel : définition d'un service

- « un **service** est un **comportement** défini par **contrat**, qui peut être réalisé et **fourni** par tout *composant* pour être **utilisé** par tout *composant*, sur la base unique du **contrat** »
[Bieber and Carpenter 2002].



Rappel : les 3 acteurs principaux dans SOA



Définition d'un service WEB

- C'est un objet métier accessible sur Internet
- On connaît la tâche qu'il réalise, ses entrées et sorties
- Indépendance du langage de programmation
- Exemple simple: Un service S, S(a,b)=a+b
- Complexité inconnue

Exemples existants de services Web

- **GoogleSearch Web Service**
  - **Le fameux moteur de recherche**

  http://api.google.com/search/beta2

- **Service California Traffic Conditions**
  - **Conditions de circulation sur les rues de Californie en donnant le numéro de la rue**

  http://services.xmethods.net:80/soap/servlet/rpcrouter

- **AirportWeather Web Service**
  - **Information sur les conditions météo des aéroports du monde en donnant le code de l'aéroport**

  http://live.capescience.com/ccx/AirportWeather

- **Service Weather Temperature**
  - **température d'un lieu donné aux US à partir de son code postal**

  http://services.xmethods.net:80/soap/servlet/rpcrouter

Exemple détaillé: Google

Search requests

- Soumet une requête avec un ensemble de paramètres à Google Web APIs service et reçoit en réponse un ensemble de résultats de recherche.

Cache requests

- Soumet une URL à Google Web APIs service et reçoit en réponse le contenu de l'URL lors de la dernière visite du crawler Google.

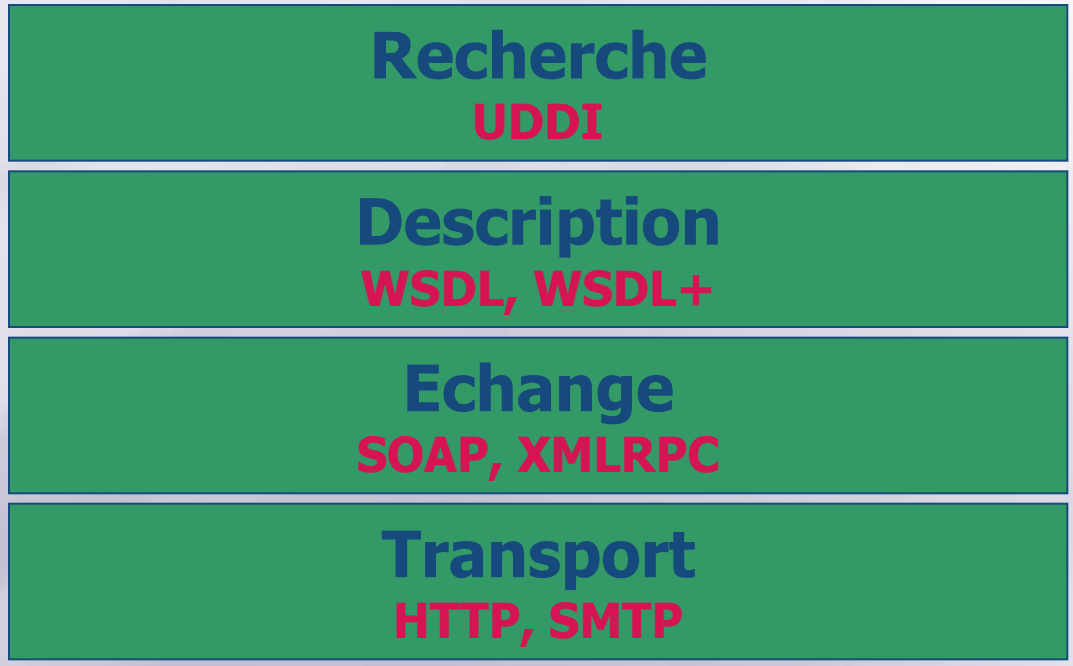
Spelling requests

- Soumet une requête à Google Web APIs service et reçoit en réponse une suggestion de correction orthographique pour la requête.

Les Composants

- **Service Provider (Fournisseur de service)**
  - Application s'exécutant sur un serveur et comportant un module logiciel accessible par Internet en XML
- **Service Registry (Annuaire de service)**
  - Annuaire des services publiés par les providers (UDDI)
  - Géré sur un serveur niveau application, entreprise ou mondial
- **Service Requester (Demandeur de service)**
  - Application cliente se liant à un service et invoquant ses fonctions par des messages XML (SOAP)

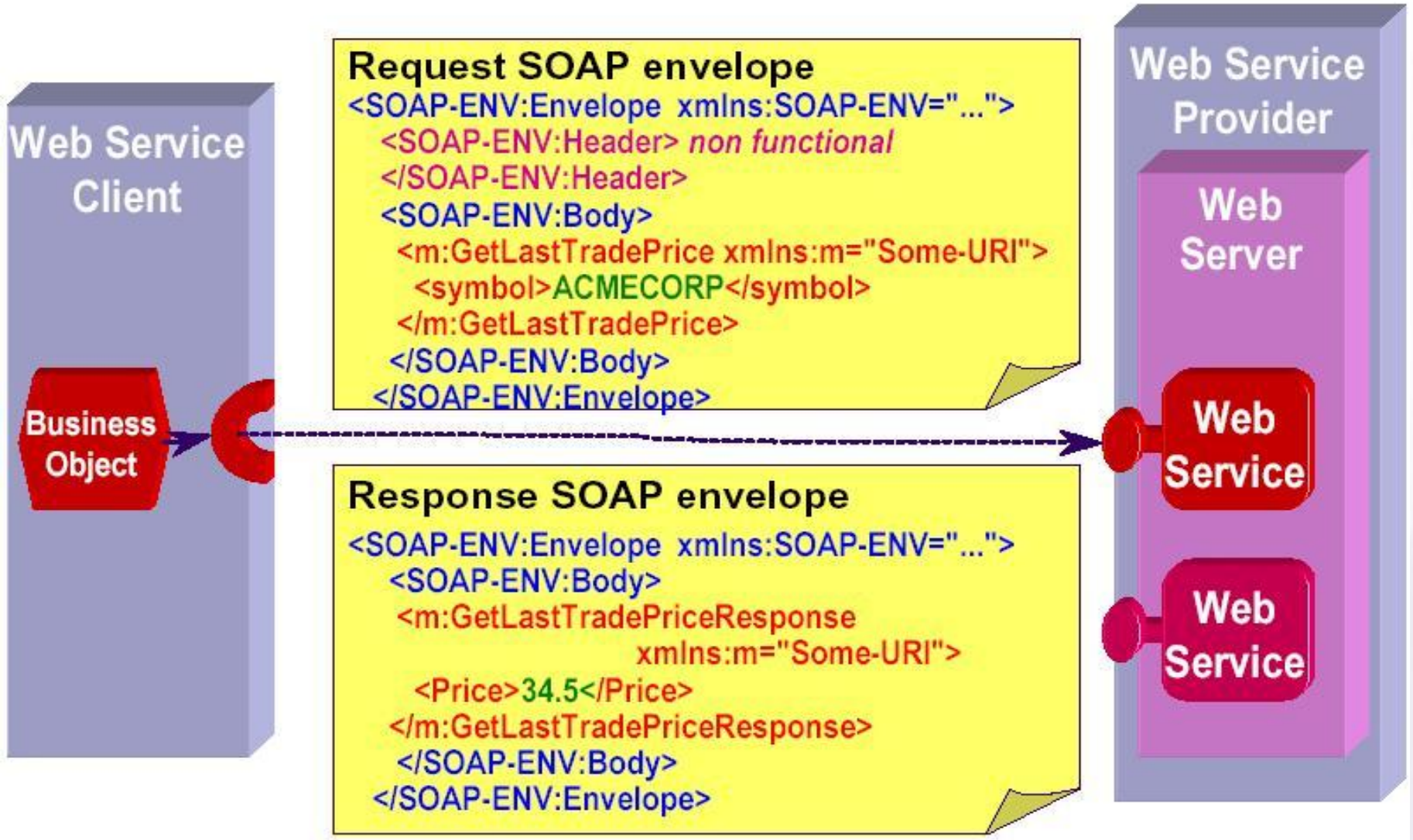
Architecture à base de services



Simple Object Access Protocol

- **Transport**
  - RPC, RMI, HTTP, SMTP/POP3/IMAP…
- **Encodage**
  - Sérialisation XML de types simples et complexes
- **L'enveloppe SOAP**
  - Entête: Aspects non fonctionnels
    - sécurité, mode d'accès, Session…
  - Corps: Aspects fonctionnels
    - Méthodes et leurs entrées/sorties

SOAP: exemple



Web Service Description Language

- **Décrit un service comme un ensemble d'opérations (méthodes) échangeant des messages et liées à des serveurs**

- **Définition des structures utilisées:**

```
<xsd:complexType name="addresse">
        <xsd:all>
                <xsd:element name="NumeroRue" type="xsd:int"/>
                <xsd:element name="NomRue" type="xsd:string"/>
                <xsd:element name="CodePostal" type="xsd:string"/>
                <xsd:element name="Commune" type="xsd:string"/>
        </xsd:all>
</xsd:complexType>
```

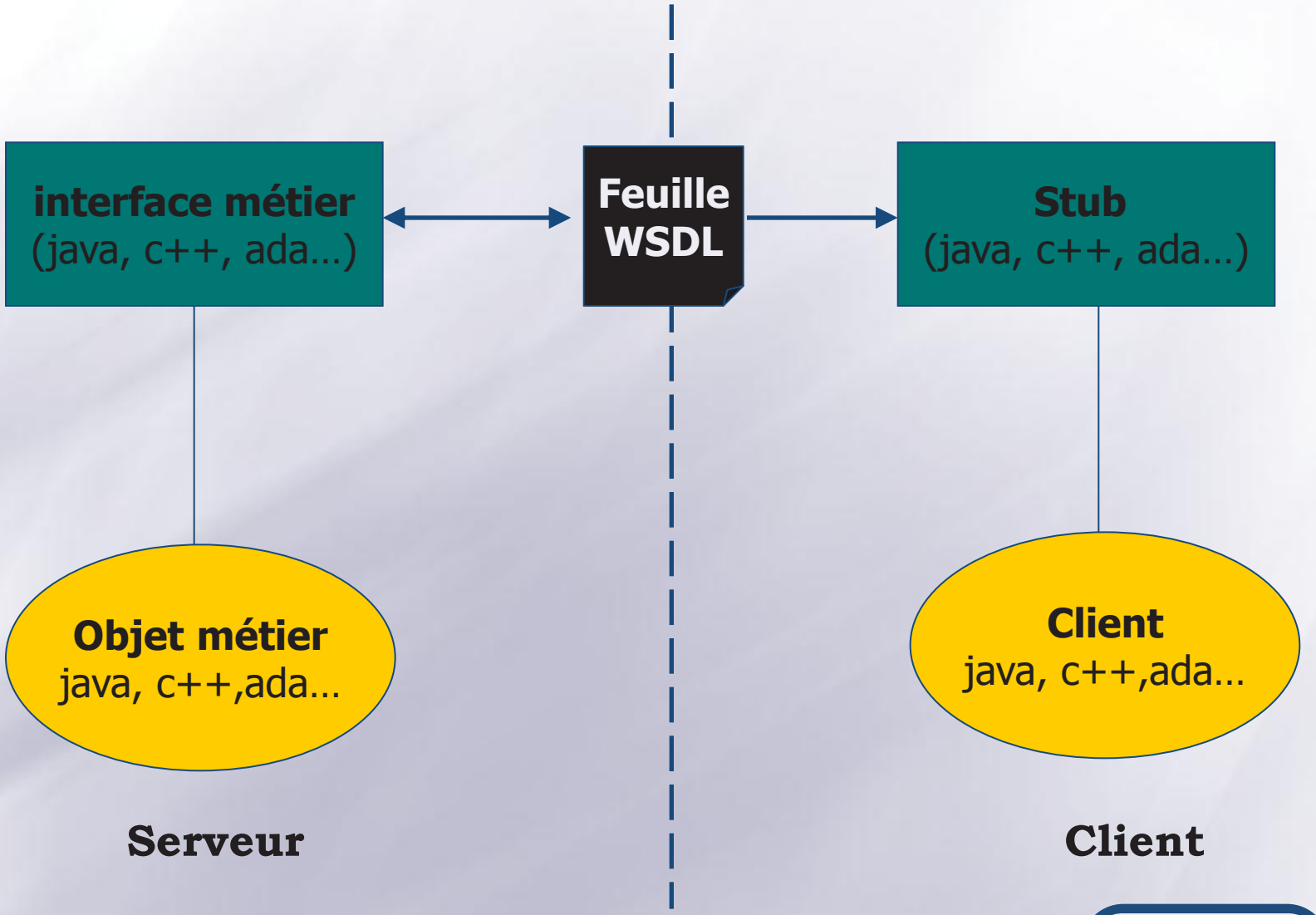
- **Définition d'un message**

```
<message name="Entree">
    <part name="nom" type="xsd:string"/>
    <part name="prenom" type="xsd:string"/>
    <part name="addresse" type="typens:addresse"/>
</message>
```

- **Définition d'une opération (portType représente l'interface du service)**

```
<portType name="Addresses">
  <operation name="AjouterAdresse">
        <input message="Entree"/>
        <output message="Confirmation"/>
```

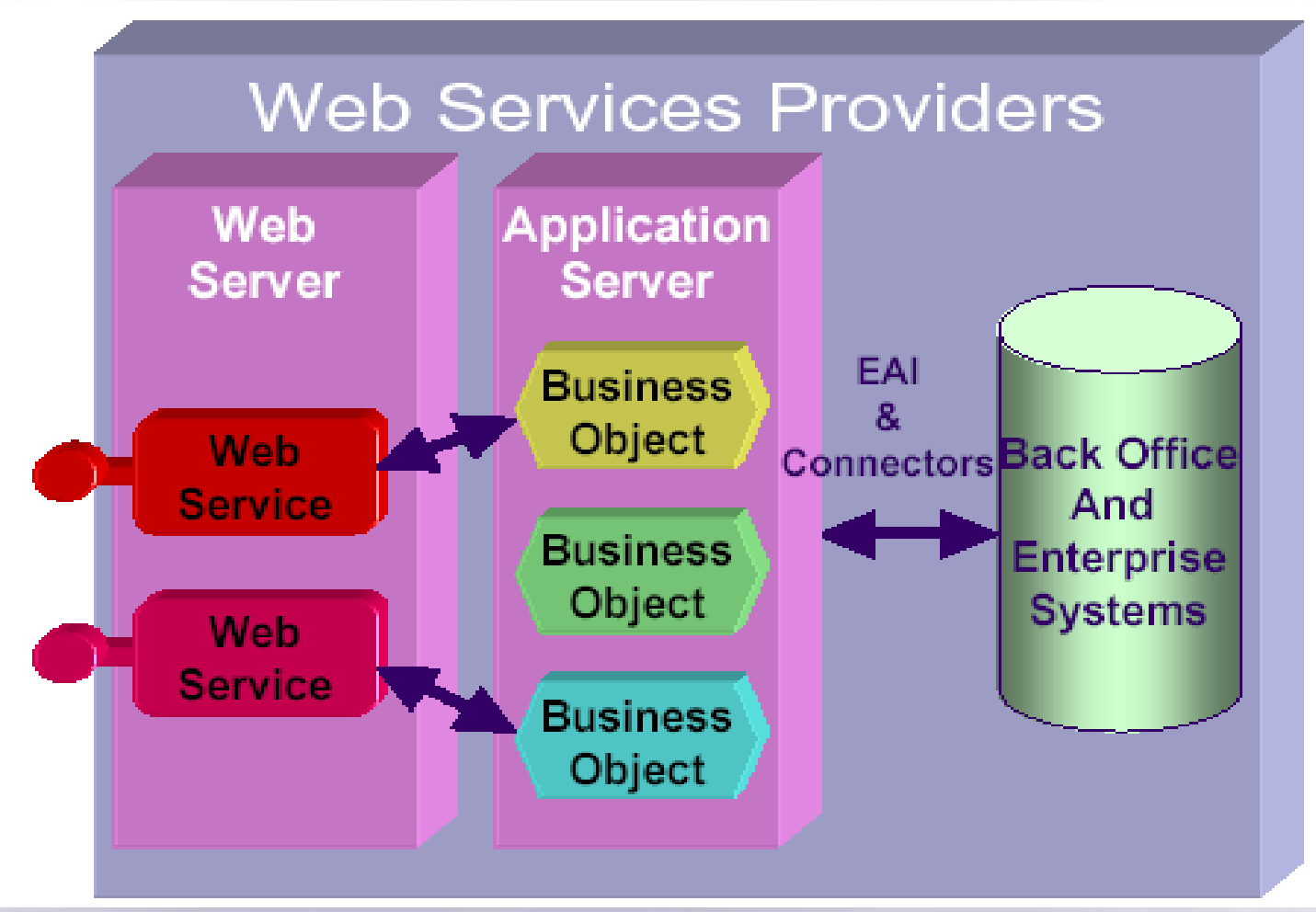
Génération WSDL



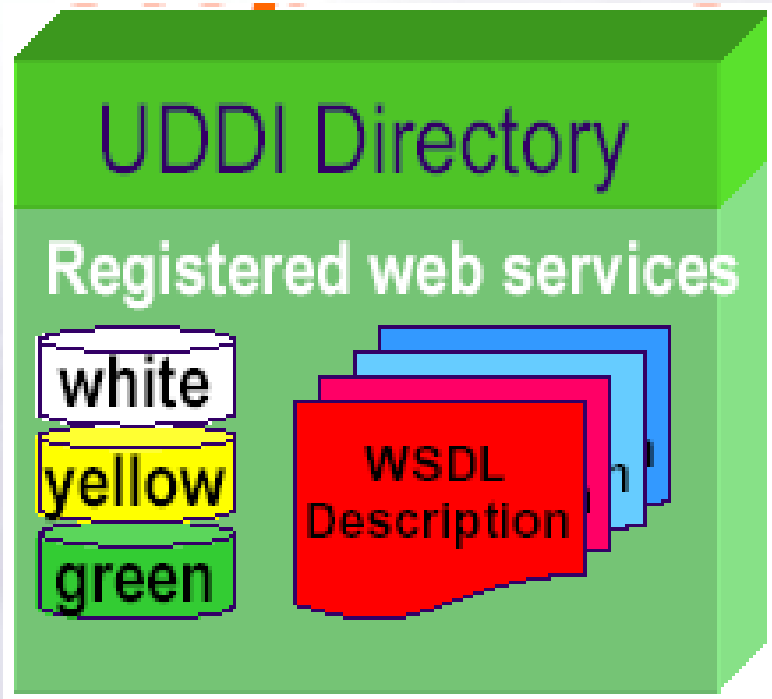
Recherche et découverte: UDDI

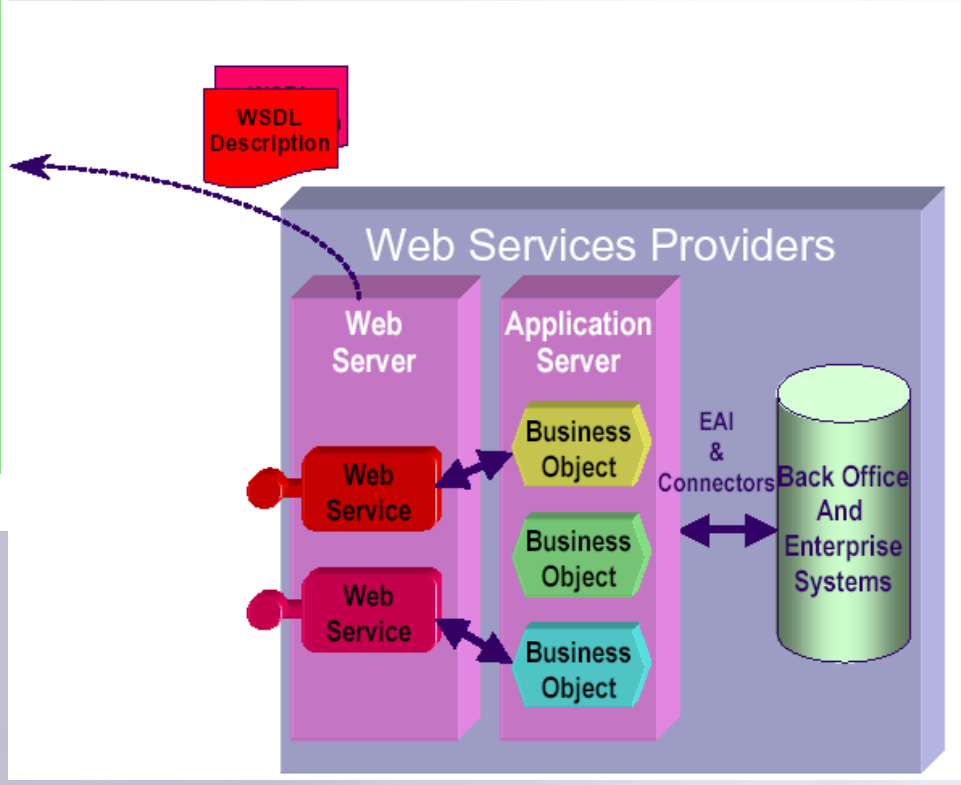
- Universal description, discovery and integration
- Annuaire mondial de services web
- Pages blanches
  - Nom et coordonnées du fournisseur + description générique du service
- Pages jaunes
  - Classement des services par catégories / domaines
- Pages vertes
  - WSDL + document d'utilisateur

Cycle de vie: 1 - Déploiement

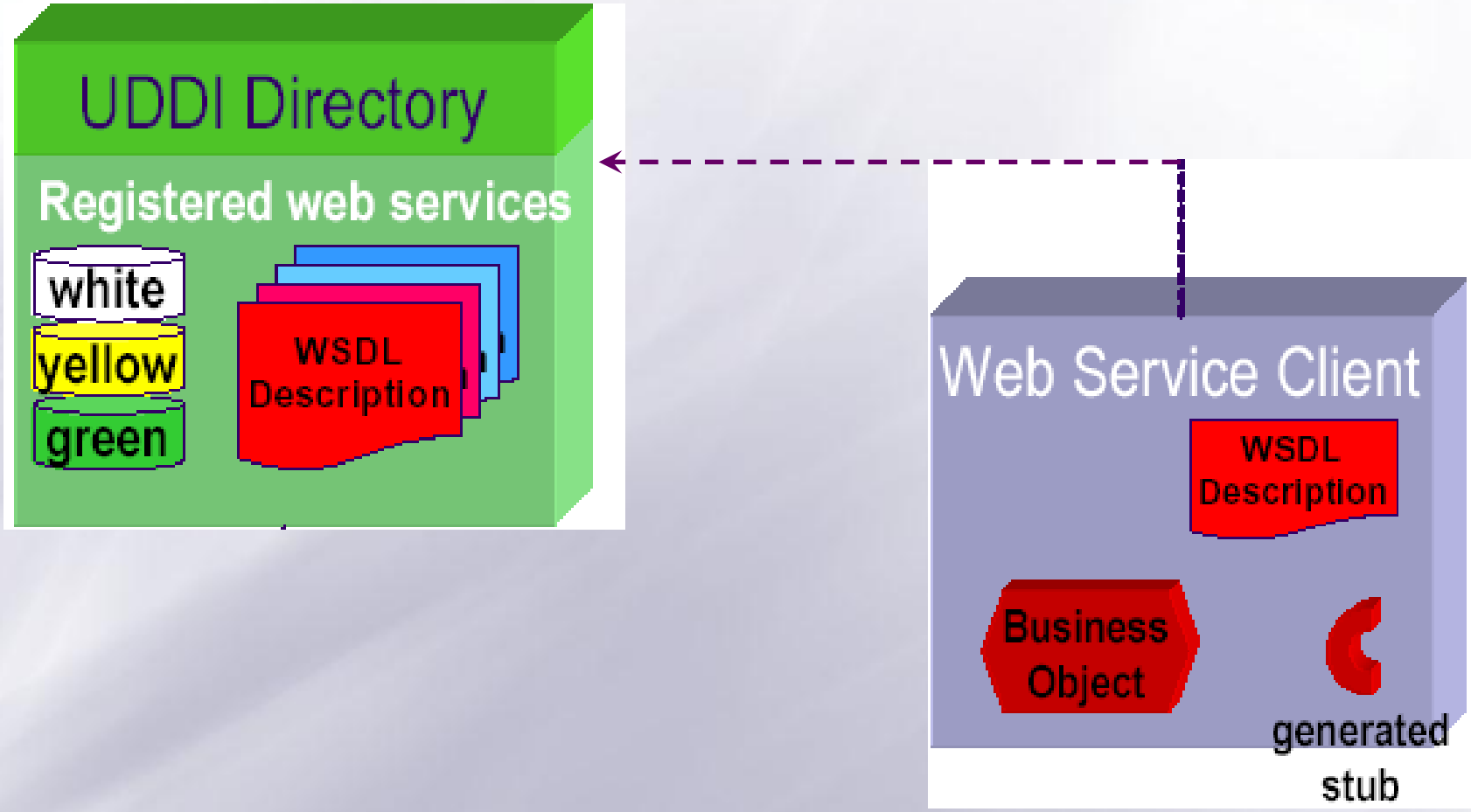


Cycle de vie: 2 - Enregistrement

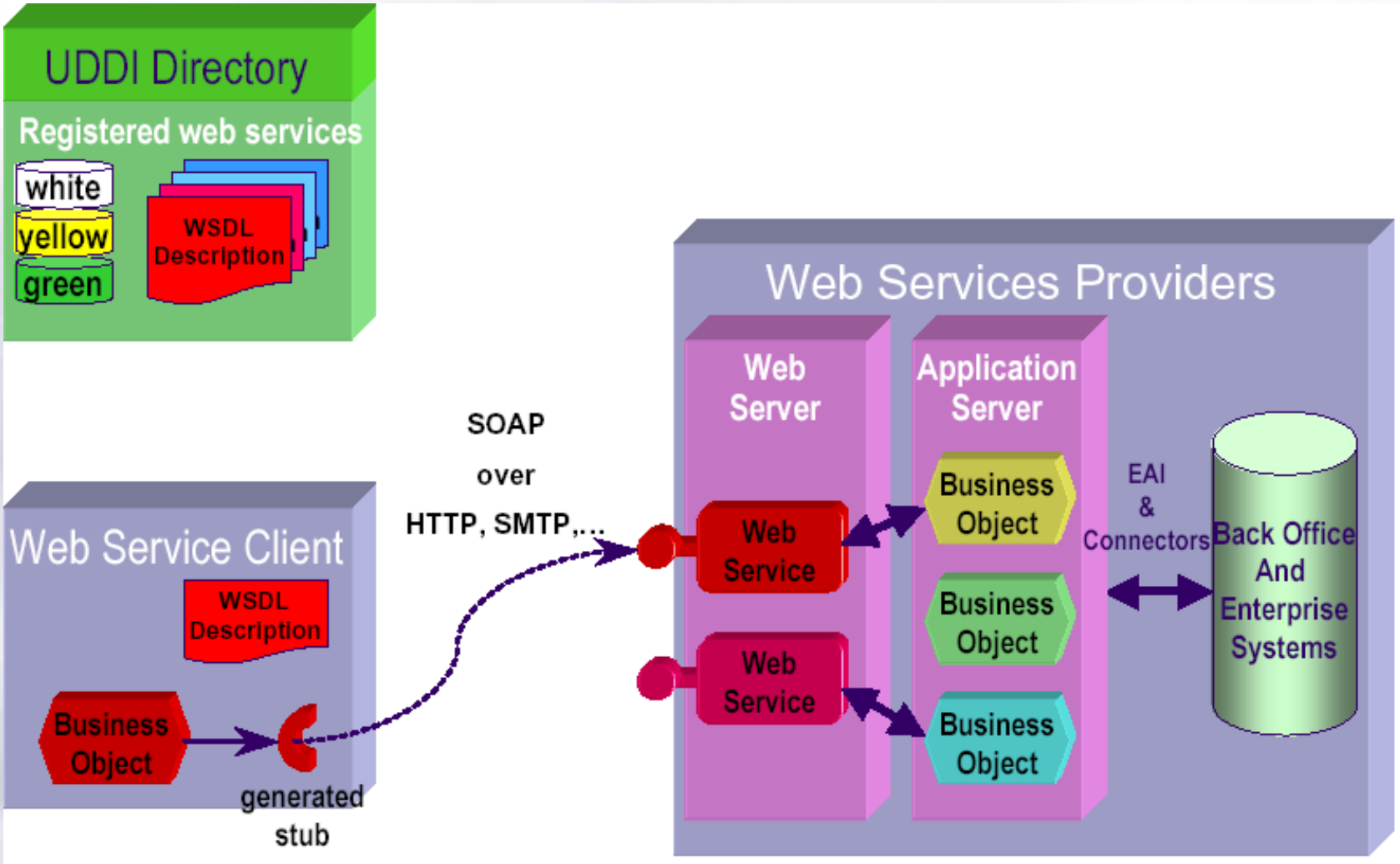




Cycle de vie: 3 - Recherche



Cycle de vie: 4 - Invocation



Exercice: Client Google (devenu indisponible)

- Récupérez l'archive tpGoogle.zip
- Utilisez le batch wsdl2java pour générer les classes nécessaires à l'invocation du service GoogleSerach (ce fichier utilise l'api axis 1.1)
- Essayez de comprendre le rôle de chaque classe en élaborant un schéma UML illustrant les différentes relations entre ces classes
- Développez un client java qui utilise ces classes pour appeller l'opération doGoogleSearch de ce service web (voir utilisationDoGoogleSearch.txt)

Exercice: Client IP2Geo

- IP2Geo est un service web qui permet de déterminer l'adresse physique (pays, ville…) correspondante à une adresse IP donnée
- Récupérez l'archive tpIP2Geo.zip disponible sur www.redcad.org/members/tarak.chaari/cours/qos/tpIP2Geo.zip
- Utilisez le batch wsdl2java pour générer les classes nécessaires à l'invocation du service IP2Geo (ce fichier utilise l'api axis 1.1)
- Essayez de comprendre le rôle de chaque classe générée
- Développez un client java qui utilise ces classes pour appeler l'opération resolveIP de ce service web (voir utilisationResolveIP.txt)

http://ws.cdyne.com/ip2geo/ip2geo.asmx

Exercice: Création d'un service Web sous Axis

http://www.redcad.org/members/tarak.chaari/cours/qos/TPWS.pdf

Support de cours

http://www.redcad.org/members/tarak.chaari/cours/cours_soa.pdf