

Construction d'applications réparties en mode message

Construction d'applications réparties en mode message

Construction d'applications réparties en mode message

Rappel sur le réseau

Construction d'applications réparties en mode message

Rappel sur le réseau
Internet

Construction d'applications réparties en mode message

Rappel sur le réseau
Internet
L'infrastructure

Construction d'applications réparties en mode message

Rappel sur le réseau

Internet

L'infrastructure

Notion de protocole

Construction d'applications réparties en mode message

Rappel sur le réseau

Internet

L'infrastructure

Notion de protocole

Mode C/S en mode message

Construction d'applications réparties en mode message

Rappel sur le réseau

Internet

L'infrastructure

Notion de protocole

Mode C/S en mode message

Interface de programmation TCP

Construction d'applications réparties en mode message

Rappel sur le réseau

- Internet

- L'infrastructure

- Notion de protocole

Mode C/S en mode message

Interface de programmation TCP

Interface de programmation UDP

Construction d'applications réparties en mode message

Rappel sur le réseau

- Internet

- L'infrastructure

- Notion de protocole

Mode C/S en mode message

Interface de programmation TCP

Interface de programmation UDP

Interface de programmation MultiCast

Construction d'applications réparties en mode message

Rappel sur le réseau

Internet

L'infrastructure

Notion de protocole

Mode C/S en mode message

Interface de programmation TCP

Interface de programmation UDP

Interface de programmation MultiCast

Les applis C/S Internet

Construction d'applications réparties en mode message

Rappel sur le réseau

- Internet

- L'infrastructure

- Notion de protocole

Mode C/S en mode message

Interface de programmation TCP

Interface de programmation UDP

Interface de programmation MultiCast

Les applis C/S Internet

Conclusion

Rappel sur le réseau : application répartie

Coopération d'un ensemble de logiciels s'exécutant sur plusieurs sites reliés par des réseaux de communication

Communication par message sur un réseau

Un message : une structure d'octets passés entre deux niveaux d'un système

- application, couche OSI, système d'exploitation
- l'entête permet l'interprétation du corps du message
- un ensemble de messages + règles d'échanges : protocole

Rappel sur le réseau : le modèle OSI

Rappel sur le réseau : le modèle OSI

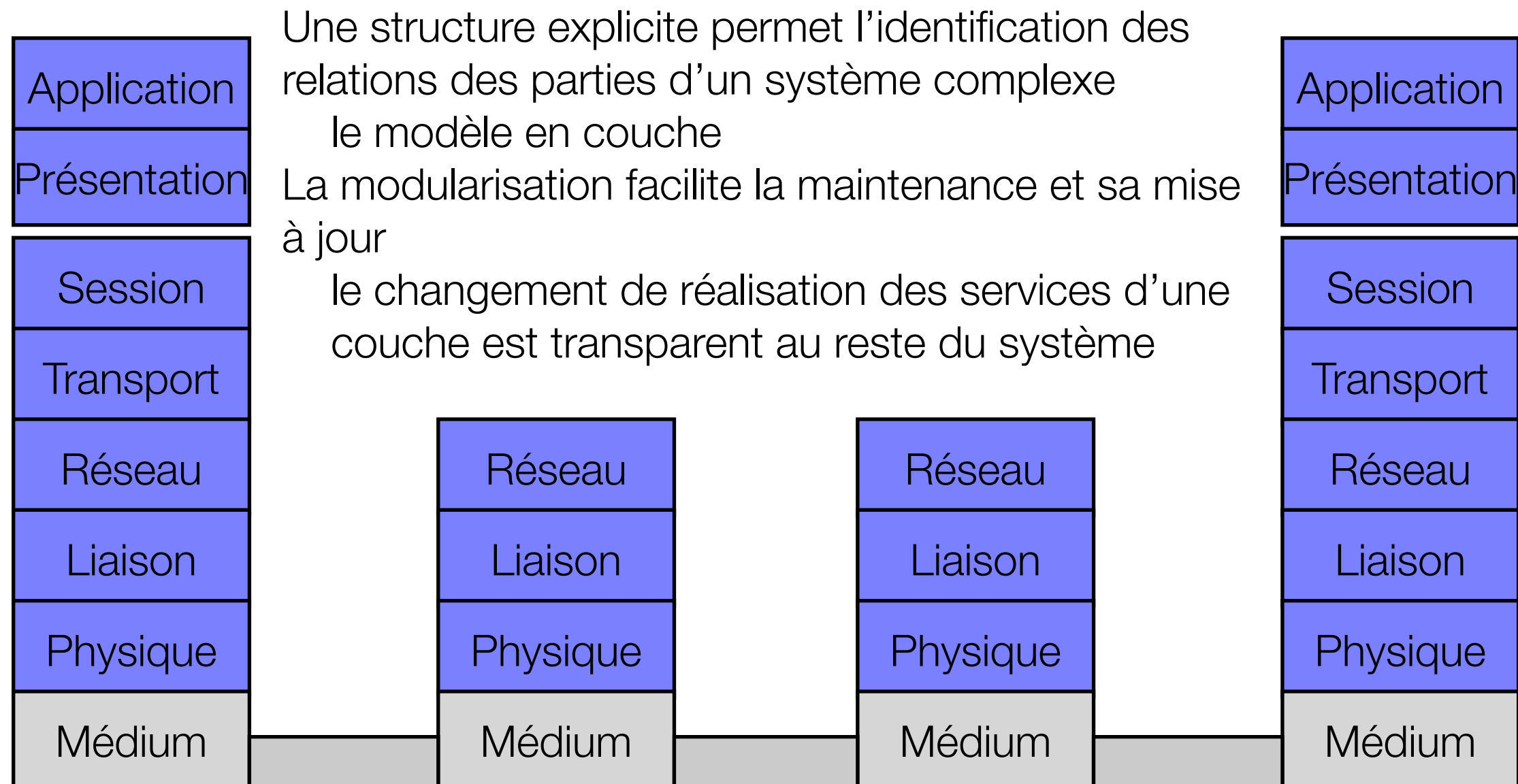
Une structure explicite permet l'identification des relations des parties d'un système complexe

le modèle en couche

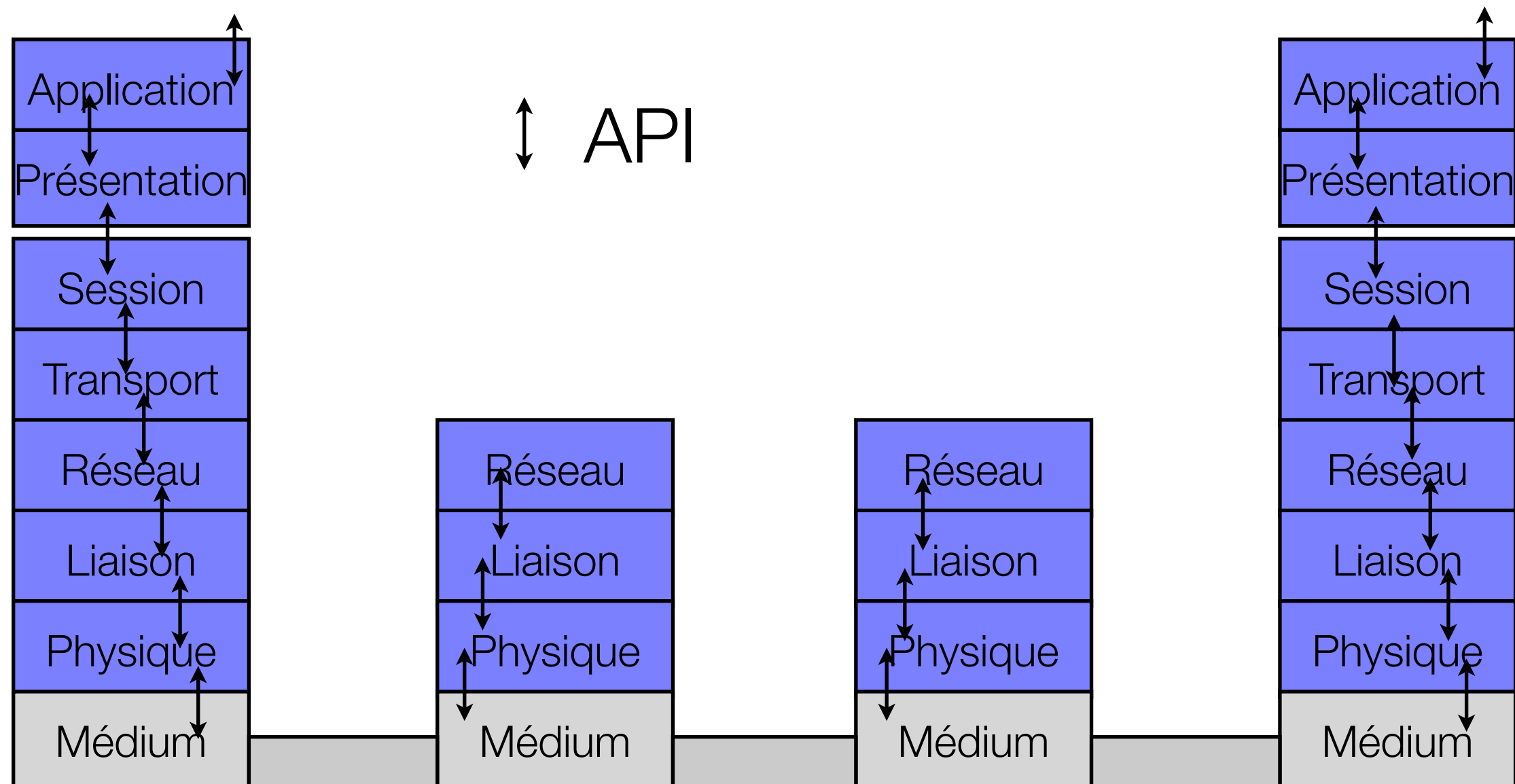
La modularisation facilite la maintenance et sa mise à jour

le changement de réalisation des services d'une couche est transparent au reste du système

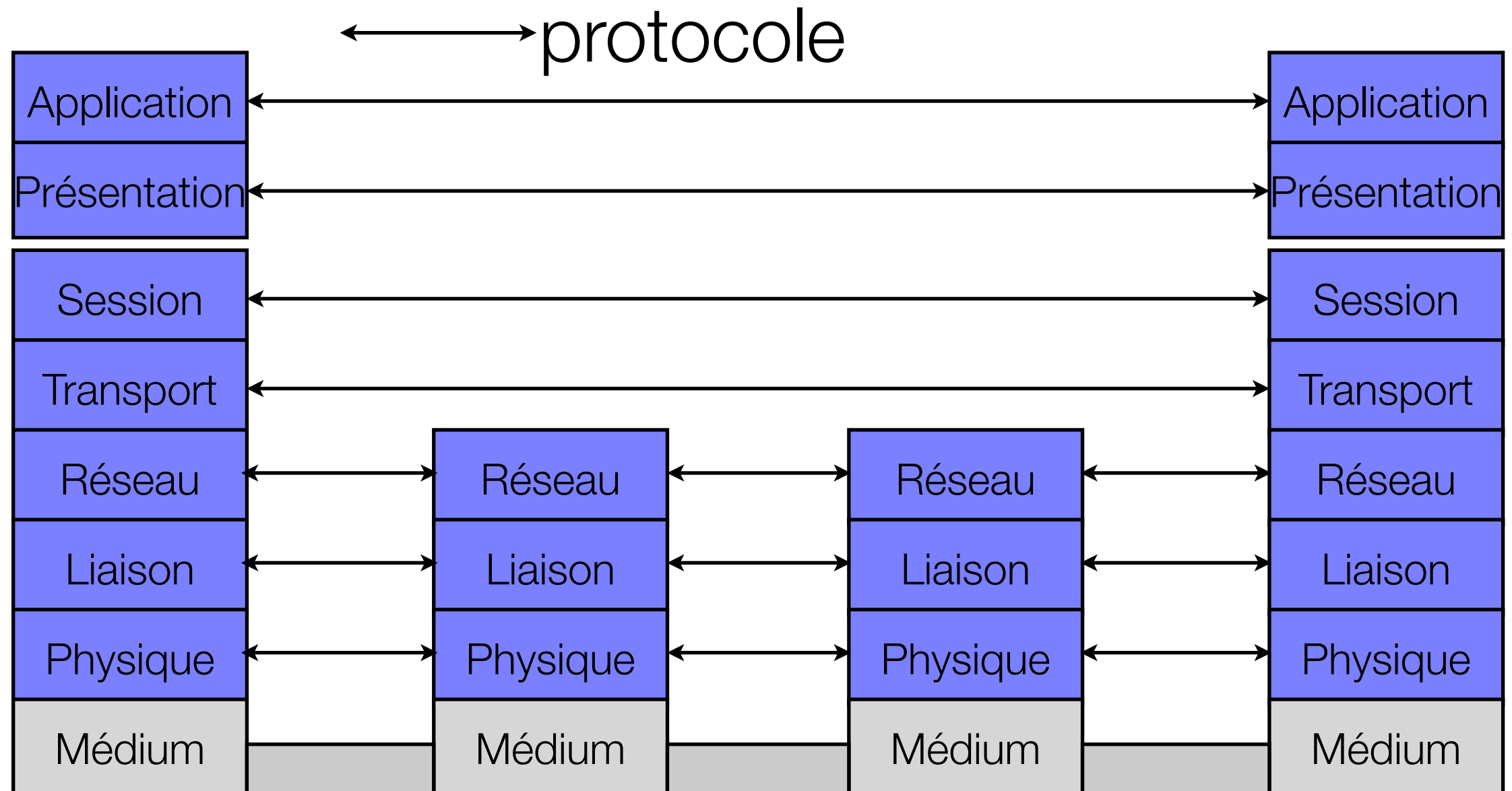
Rappel sur le réseau : le modèle OSI



Rappel sur le réseau : le modèle OSI



Rappel sur le réseau : le modèle OSI



Rappel sur le réseau : internet

des millions de machines :

- stations de travail, serveurs, portables, PDA...
- exécutent des applications réparties

des liaisons :

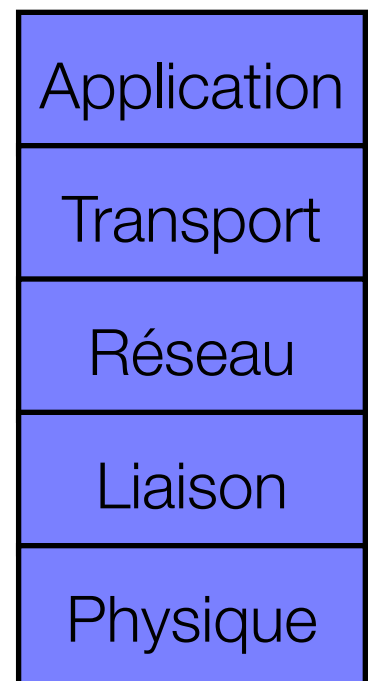
- filaire, radio, satellite

des routeurs :

- transmettent les paquets (datagrammes) à travers le réseau

Rappel sur le réseau : internet

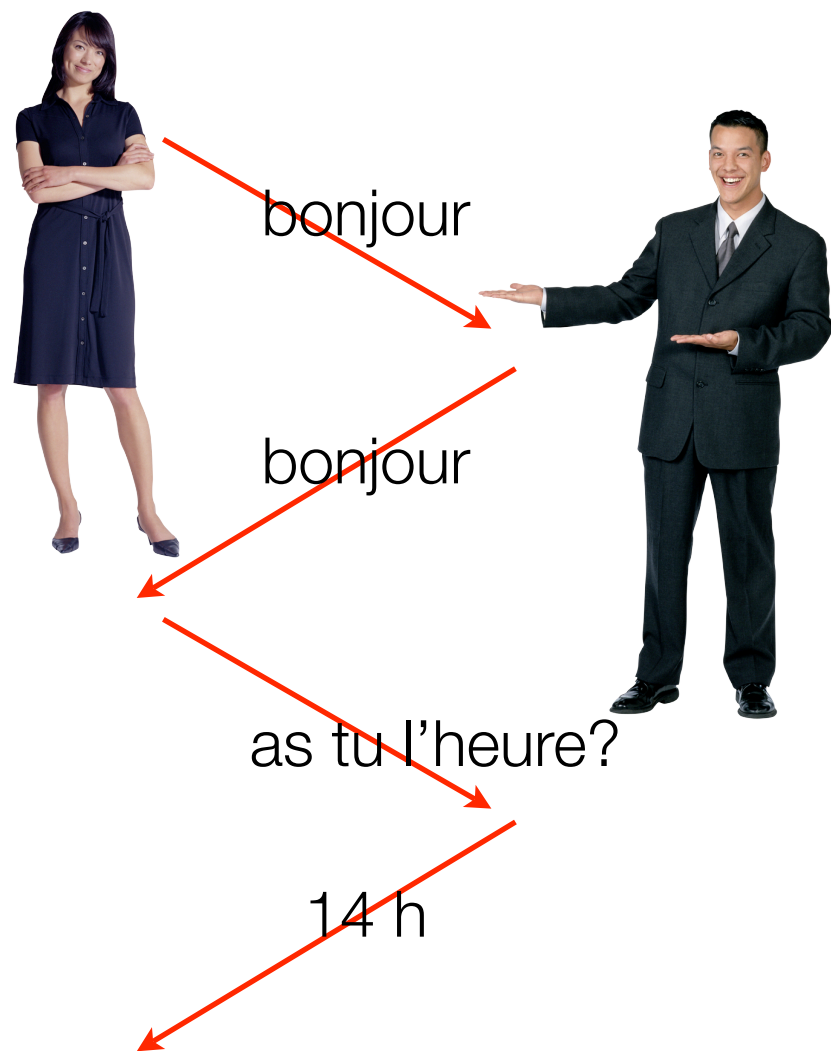
- Application : applications supportées par internet
http, smtp, ftp
- Transport : transfert de bout en bout
tcp, udp
- Réseau : routage des datagrammes de la source vers la destination
ip, protocoles de routage
- Liaison : transfert de données entre éléments voisins
ppp, ethernet
- Physique : bits sur le câble



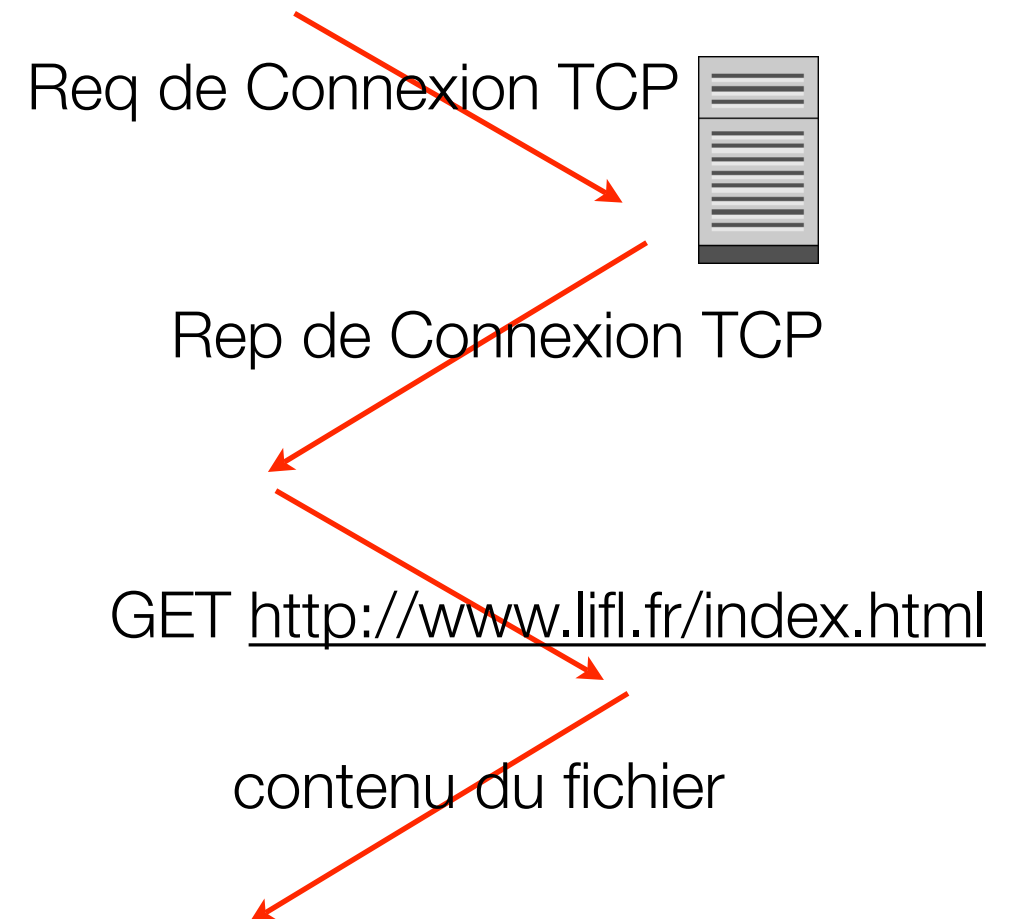
Rappel sur le réseau : l'infrastructure

- Protocoles
 - envoi et réception de messages
 - IP, TCP, UDP, HTTP, FTP, PPP
- Internet
 - Réseau de réseaux
 - Hiérarchique
 - Privé/public
- Standards
 - RFC : Request for Comments

Rappel sur le réseau : notion de protocole



temps



d'autres protocoles humains ?

Rappel sur le réseau : notion de protocole

- des protocoles humains
 - Quelle heure est-il ?
 - J'ai une question...
 - Introductions
- des protocoles réseaux
 - des machines plutôt que des humains
 - toutes les activités de communication d'internet sont gouvernées par des protocoles
- envois de messages spécifiques
- actions spécifiques exécutées lorsque des messages sont reçus,...

Rappel sur le réseau : notion de protocole

Un protocole définit :

- le format des messages
- l'ordonnancement des messages envoyés et reçus
- les actions à entreprendre à la réception d'un message donné

Rappel sur le réseau : notion de protocole

Différents modes d'organisation

- pour les extrémités du réseau
 - modèle client/serveur
WWW, email,...
 - modèle peer-to-peer : interaction symétrique
e.g.: Gnutella, KaZaA
- pour le coeur du réseau
 - commutation de circuits
téléphone
 - commutation de paquets
discrétisation - internet

Rappel sur le réseau : notion de protocole

Service orienté connexion

- Objectif : transférer des données de bout en bout
 - Connexion (handshaking) : initialisation et préparation du transfert de données à l'avance
 - Etat d'initialisation pour les deux hôtes communicants
- TCP – Transmission Control Protocol
Service orienté connexion d'internet (RFC 793)
 - transfert de données fiable, ordonné
 - perte : ack et retransmission
 - contrôle de flux
 - l'émetteur s'adapte à la vitesse du récepteur
 - contrôle de congestion
 - les émetteurs ralentissent lorsque le réseau est congestionné

Rappel sur le réseau : notion de protocole

Service orienté sans connexion

- Objectif : transférer des données de bout en bout
- UDP – User Datagram Protocol
Service orienté sans connexion d'internet (RFC 768)
 - transfert de données non fiable
 - pas de contrôle de flux
 - pas de contrôle de congestion
- Les applications utilisant TCP :
HTTP, FTP, Telnet, SMTP, ...
- Les applications utilisant UDP :
téléconférence, téléphone sur Internet, SNMP, ...

Client/Serveur en mode message

- protocole de niveau applicatif :
un élément du logiciel applicatif qui définit les messages échangés entre applications et les actions qui utilisent les services de communication fournis pas les protocoles de niveau bas (TCP, UDP)
- Client :
 - il initie le contact avec le serveur
 - il demande des services au serveur

Web: client intégré dans le navigateur, e-mail: dans le lecteur de mails
- Serveur :
 - il fournit les services demandés par le client

serveur Web envoie la page demandée, serveur mail délivre les e-mails

Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres

Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client

Serveur

Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres

Client

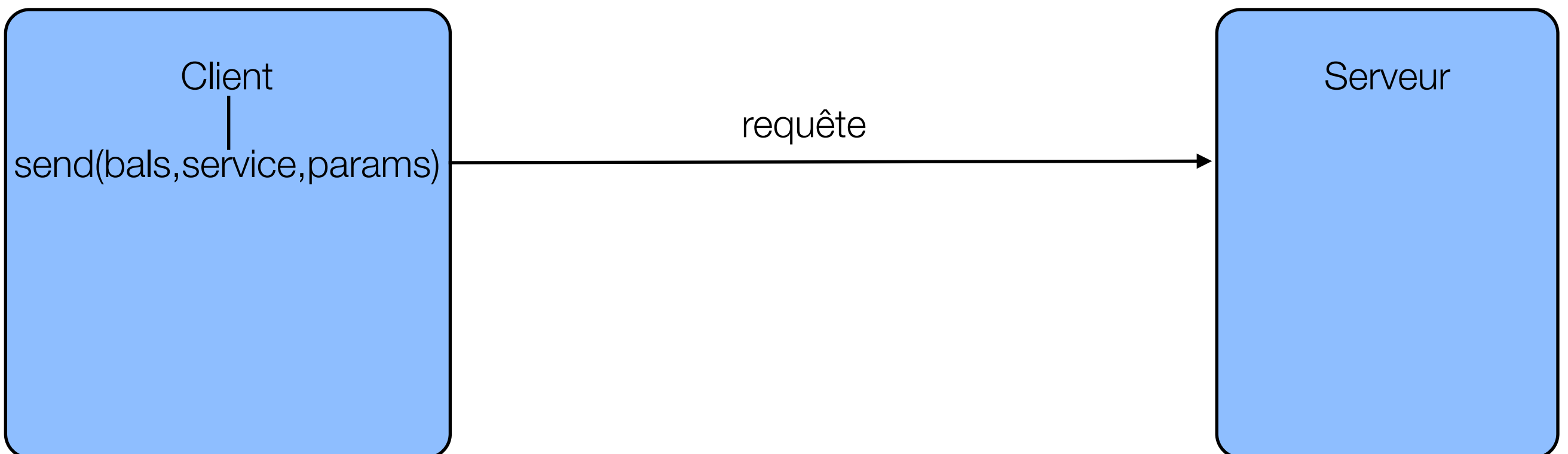
send(bals,service,params)

Serveur

Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

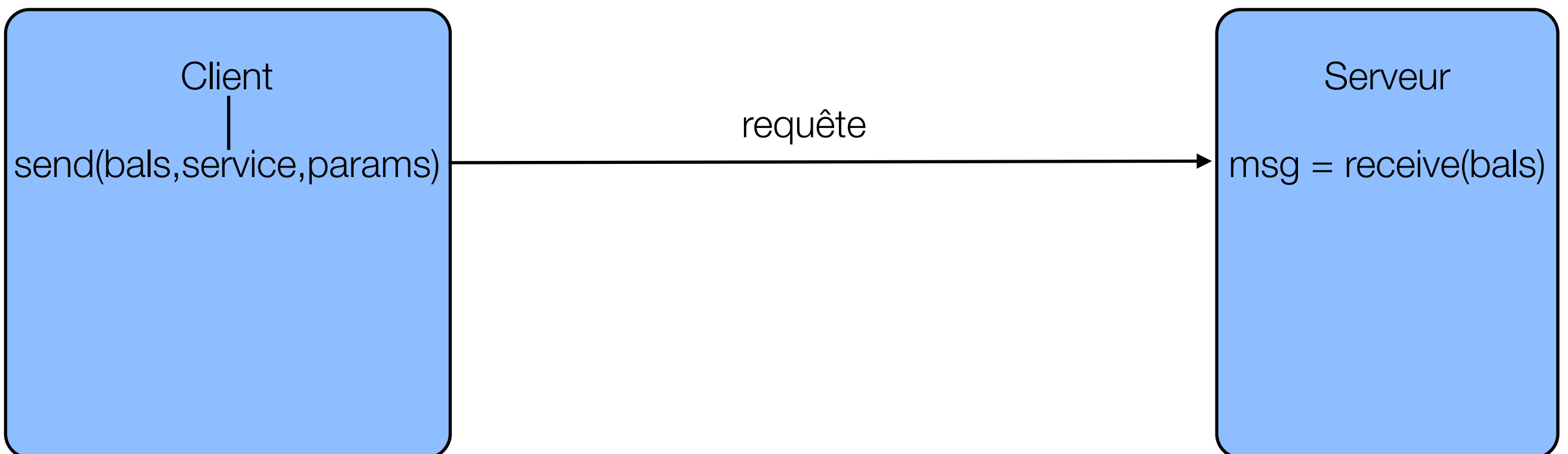
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

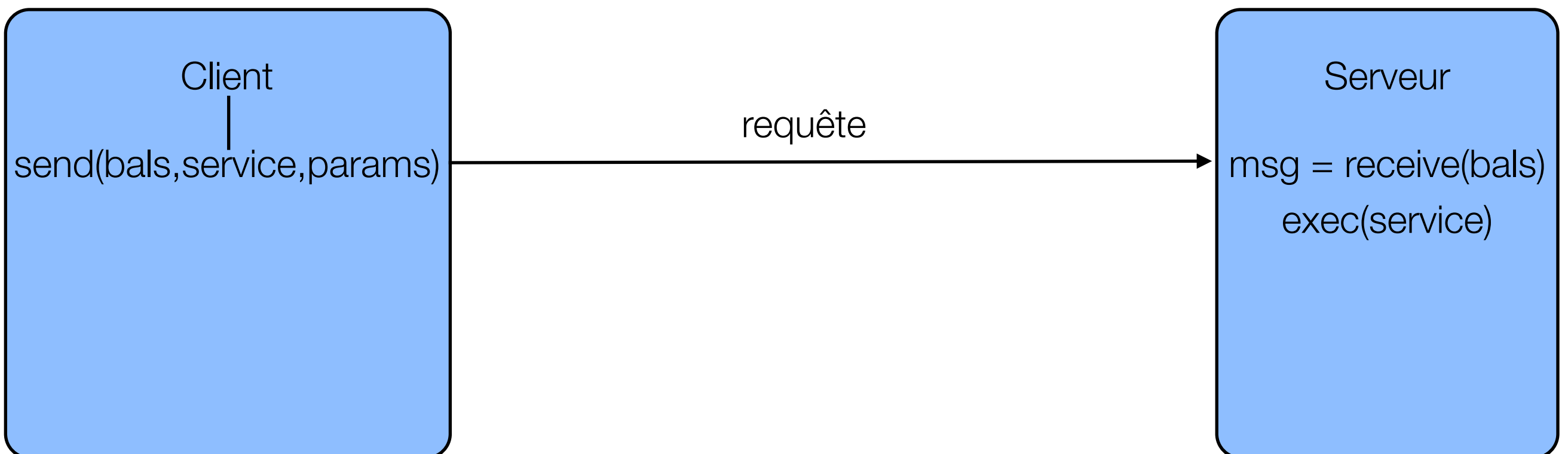
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

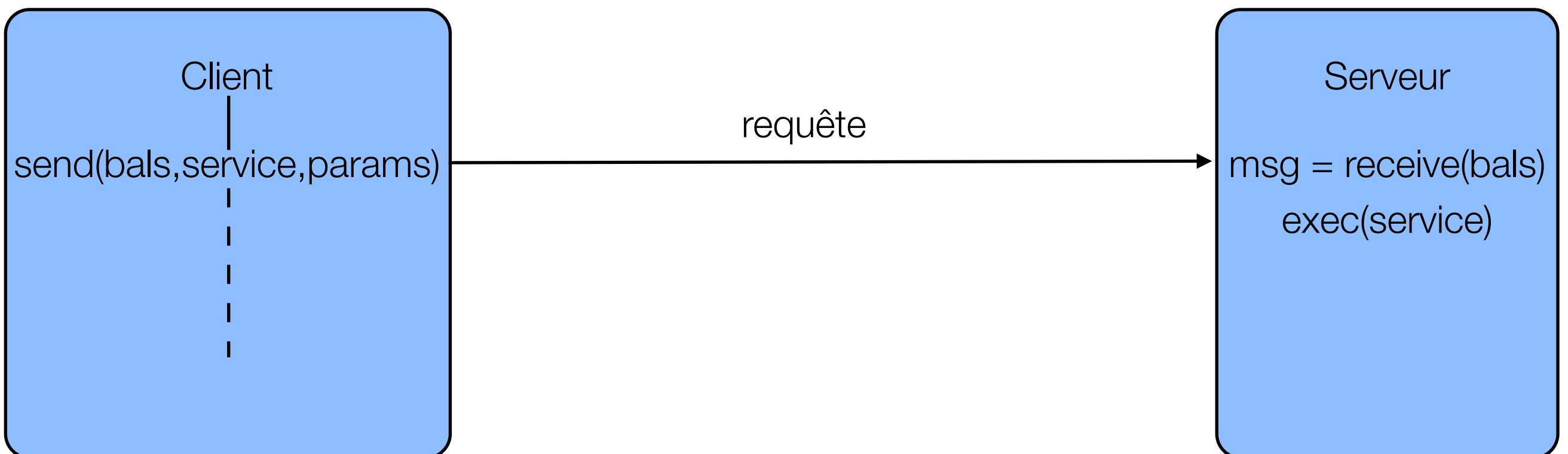
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

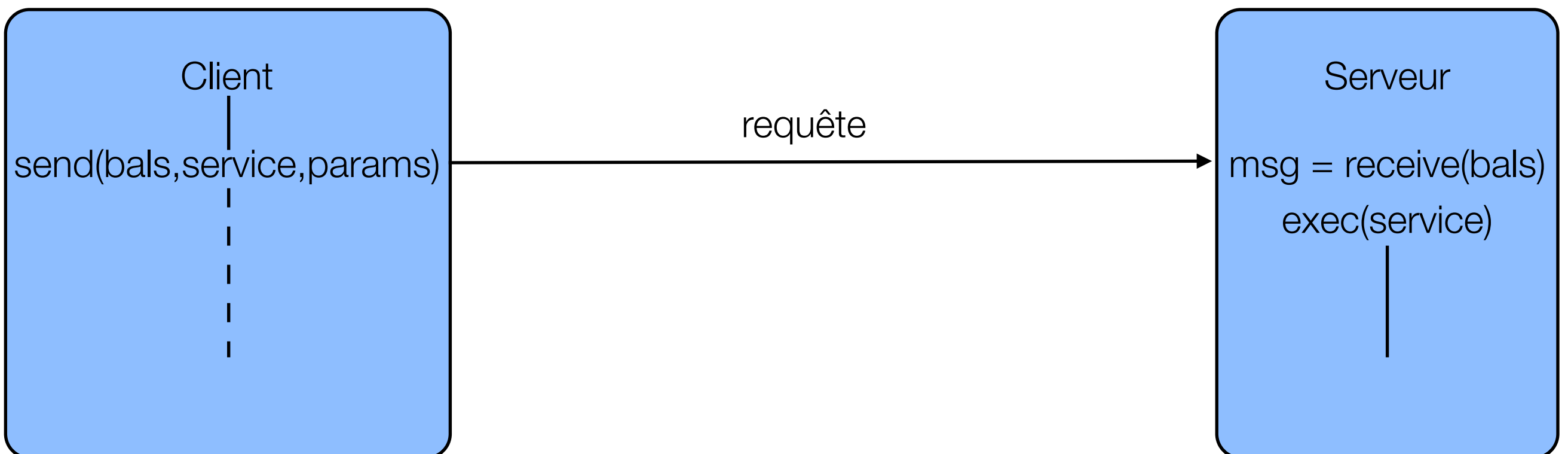
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

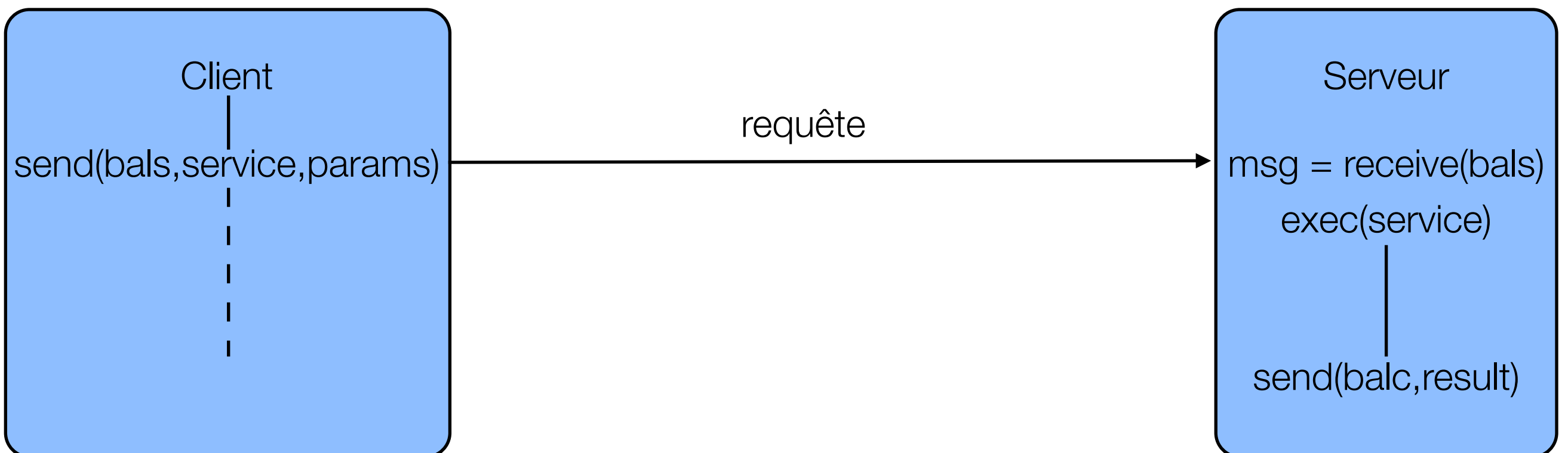
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

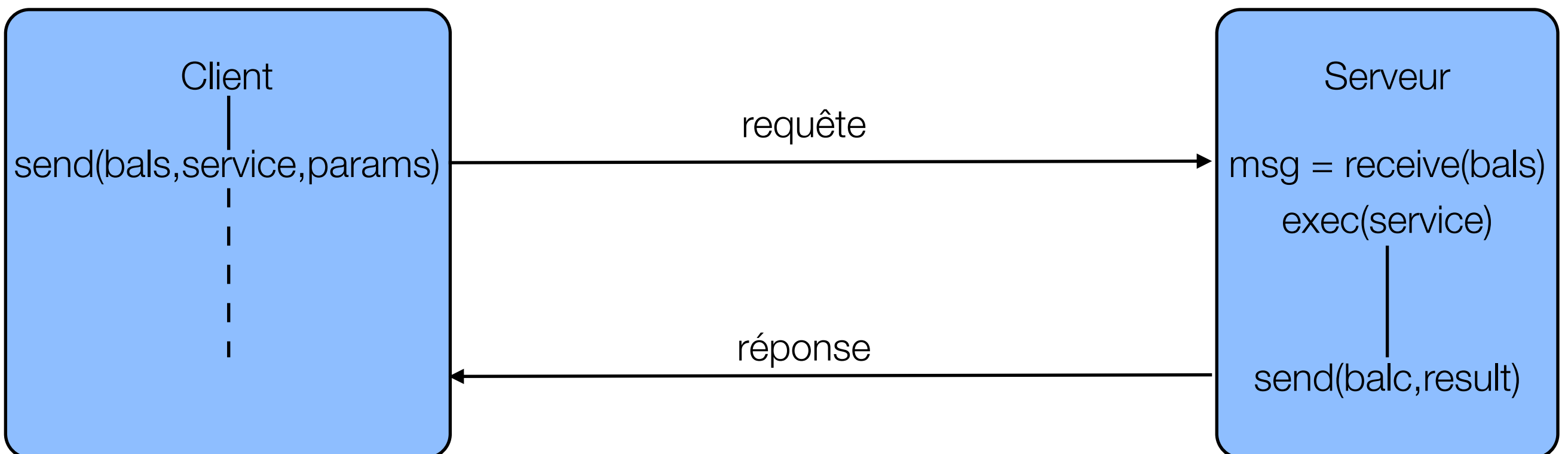
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

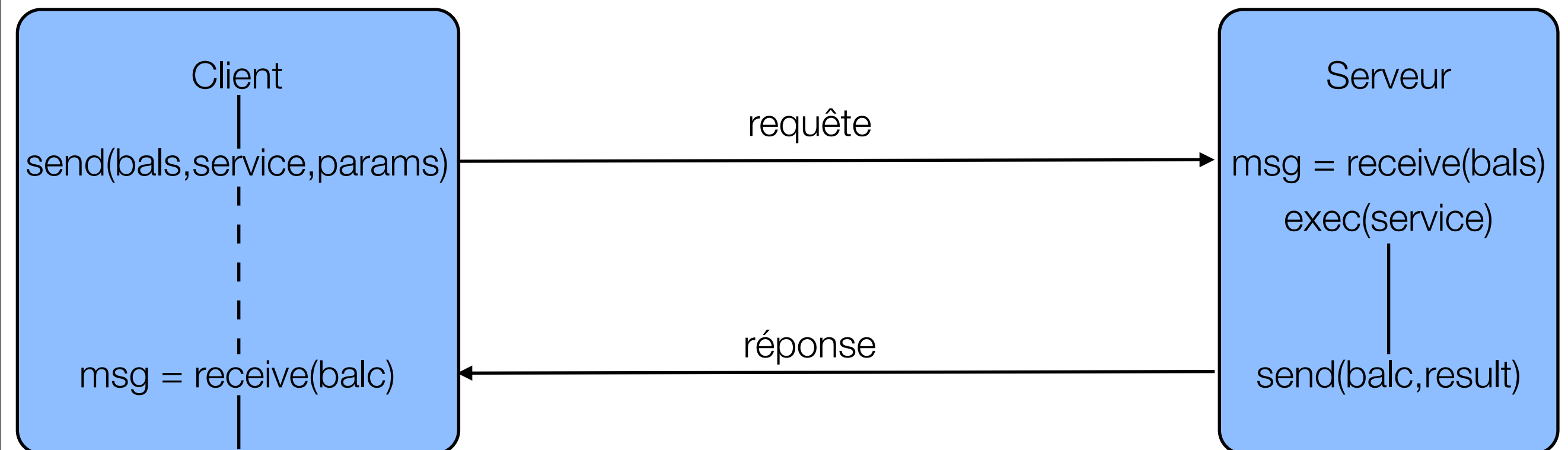
- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

Exemple de réalisation avec une communication par message ou communication asynchrone

- émission non bloquante
- réception bloquante
- communication par message typé
- communication sur les processus destinataires ou sur des ports, sur des boîtes à lettres



Client/Serveur en mode message

- API : Application Programming Interface
 - définit l'interface entre les couches application et transport
 - socket : API Internet
 - deux processus communiquent par envoi et réception de données sur des sockets
- Comment un processus identifie un autre processus pour communiquer ?
 - par l'adresse IP qui accueille l'hôte distant
 - par le numéro de port qui permet de déterminer le processus participant à la communication

Client/Serveur en mode message

Adressage Internet

chaque carte réseau est un point d'accès au réseau et possède une adresse physique Internet (@IP) unique

- une machine peut avoir plusieurs accès donc plusieurs @IP
- @IP = un entier 32 bits pour IPv4 (128 pour IPv6)

4 classes d'adressage

- A : 128 réseaux, 16 M de points d'accès par réseau
- B : 16 K réseaux, 64 K de points d'accès par réseau
- C : 2 M réseaux, 256 points d'accès par réseau
- D : adresse multicast IP

la correspondance adressage physique <---> adressage symbolique est gérée par une fédération de Domain Naming System

- 134.206.10.18 <-> www.lifl.fr
- 127.0.0.1 <-> localhost

Client/Serveur en mode message

la classe `java.net.InetAddress`

- Représentation des adresses réseaux Internet

`134.206.10.18`, `www.lifl.fr`

- Création

```
InetAddress host = InetAddress.getLocalHost();  
InetAddress host = InetAddress.getByName("www.lifl.fr");  
InetAddress host[] =  
InetAddress.getAllByName("www.lifl.fr");
```

- Propriétés

- Adresse symbolique : `String getHostName()`
- Adresse IP : `String.getHostAddress()`
- Adresse binaire : `byte[] getAddress()`
- Multicast ? : `boolean isMulticastAddress()`

Client/Serveur en mode message

Obtenir un objet java.net.InetAddress

```
import java.net.InetAddress;
try {
    InetAddress adresseLocale = InetAddress.getLocalHost();
} catch (java.net.UnknownHostException ex) {
    . . .
}
try {
    InetAddress adresse =
InetAddress.getByName("www.lifl.fr");
} catch (java.net.UnknownHostException ex) {
    . . .
}
try {
    InetAddress[] adresses =
InetAddress.getAllByName("www.lifl.fr");
} catch (java.net.UnknownHostException ex) { . . .
}
```

Client/Serveur en mode message

accès aux propriétés d'un java.net.InetAddress

```
public static void afficher(InetAddress adresse)
{
    String chaine = adresse.toString() ;

    String nom = adresse.getHostName() ;

    String ip = adresse.getHostAddress() ;

    byte[] binaire = adresse.getAddress() ;

    boolean multicast = adresse.isMulticastAddress() ;
}
```

Quel service de transport pour quelle application ?

- Perte de données
 - Quelques applications (e.g., audio) tolèrent des pertes de données
 - D'autres applications (e.g., transfert de fichiers, telnet) demandent un transfert de données fiable à 100%
- Délai de transmission
 - Quelques applications (e.g., téléphone sur Internet, jeux interactifs) demandent des délais courts pour être efficace
- Largeur de bande
 - Quelques applications (e.g., multimédia) demandent une bande de transmission minimum pour être efficace
 - D'autres ("applications élastiques") peuvent fonctionner avec la largeur qu'elles ont !

Besoins en services de transport des applications communes

Application	Perte données	Largeur de bande	Sensible au temps
transfert de fichiers	pas de perte	élastique	non
e-mail	pas de perte	élastique	non
web	pas de perte	élastique	non
audio/vidéo temps réel	perte tolérée	audio : 5kb-1Mb vidéo : 10kb-5Mb	100's msec
audio/vidéo stockage	perte tolérée	audio : 5kb-1Mb vidéo : 10kb-5Mb	quelques secondes
jeux interactifs	perte tolérée	quelques Kbps up	100's msec
applications financières	pas de perte	élastique	oui et non

Les protocoles transport des applications internet

Application	Protocole couche application	Protocole couche transport
e-mail	smtp (RFC 821)	TCP
Accès distant	telnet (RFC 854)	TCP
Web	http (RFC 2068)	TCP
Transfert de fichiers	ftp (RFC 959)	TCP
Flux multimedia	propriétaire	TCP ou UDP
SGF réparti	nfs (RFC 1813)	TCP ou UDP
Téléphone sur internet	sip (RFC 3261)	UDP

Interface de programmation TCP

API Socket

introduite en 1981 dans UNIX BSD4.1

les sockets sont créées, utilisées et détruites explicitement par les applications.

réalisation du paradigme client/serveur

deux types de service transport via l'API socket :

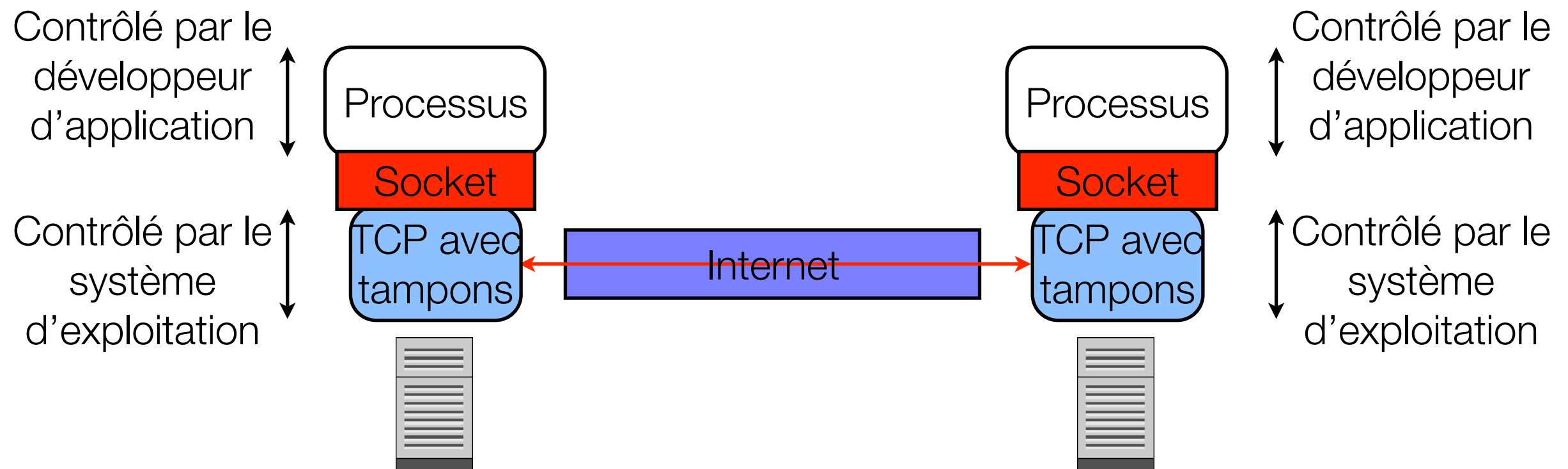
- datagramme non fiable
- fiable, orienté flux d'octet

Socket

interface locale à la machine, créée par l'application, contrôlée par l'OS (une "porte") dans laquelle le processus d'application peut envoyer et recevoir des messages vers/d'un autre processus d'application (distant ou local).

Interface de programmation TCP

Socket : une porte entre le processus d'application et le protocole de bout en bout (UDP ou TCP)



Interface de programmation TCP

Le client doit contacter le serveur :

Le processus du serveur doit avoir été lancé avant.

Le serveur doit avoir créé une socket d'accueil des clients.

Le client contacte le serveur :

création d'une socket-TCP locale en précisant l'adresse IP et le numéro de port du serveur

Quand le client TCP crée la socket :

le client TCP établit la connexion vers le serveur TCP

Quand le serveur TCP est contacté par le client :

il crée une nouvelle socket pour que le processus serveur puisse communiquer avec le client, permet au serveur de communiquer avec plusieurs clients

Interface de programmation TCP

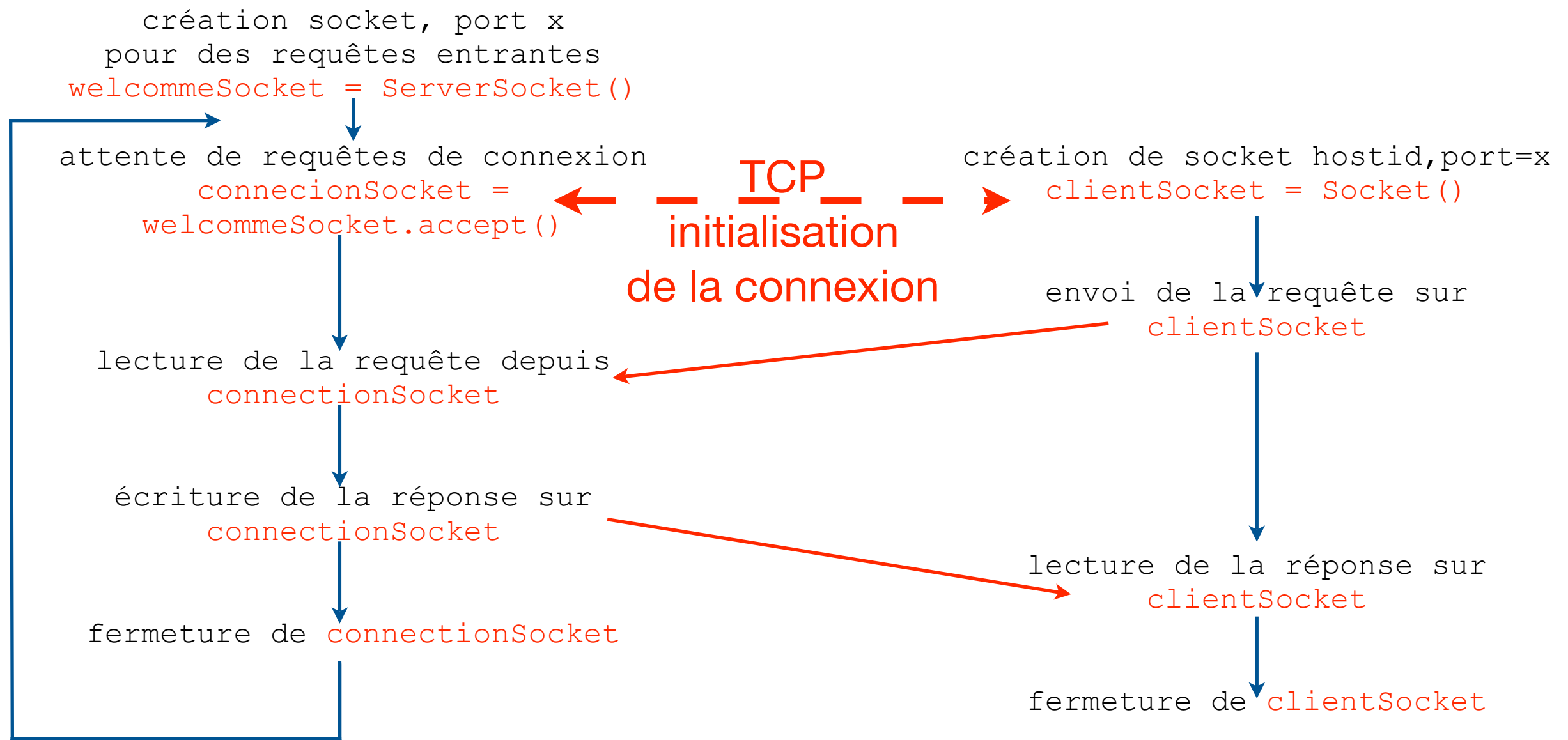
Exemple d'une application C/S :

- Le client lit une ligne sur l'entrée standard (inFromUser stream)
- et l'envoie vers le serveur via une socket (outToServer stream)
- Le serveur lit des lignes venant de la socket
- Le serveur convertit la ligne en majuscule et la renvoie vers le client
- Le client lit et affiche la ligne modifiée venant de la socket (inFromServer stream)

Interface de programmation TCP

Serveur sur hostid

Client



Interface de programmation TCP client Java

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

class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

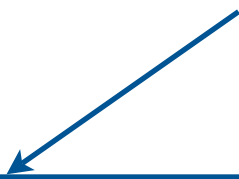
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
```

Interface de programmation TCP client Java

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

class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

Création  
du flux d'entrée


        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```

Interface de programmation TCP client Java

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

```
class TCPClient {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String sentence;
```

```
        String modifiedSentence;
```

```
        BufferedReader inFromUser =  
            new BufferedReader(new InputStreamReader(System.in));
```

```
        Socket clientSocket = new Socket("hostname", 6789);
```

```
        DataOutputStream outToServer =  
            new DataOutputStream(clientSocket.getOutputStream());
```

Création

du flux d'entrée

Création socket client

connexion avec le serveur

Interface de programmation TCP client Java

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

```
class TCPClient {
```

```
    public static void main(String argv[]) throws Exception
    {
```

```
        String sentence;
```

```
        String modifiedSentence;
```

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

```
        Socket clientSocket = new Socket("hostname", 6789);
```

```
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
```

Création

du flux d'entrée

Création socket client

connexion avec le serveur

Création du flux de sortie

attaché à la socket

Interface de programmation TCP client Java

```
BufferedReader inFromServer = new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: "+modifiedSentence);

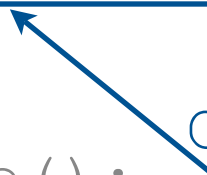
clientSocket.close();
}
}
```

Interface de programmation TCP client Java

```
BufferedReader inFromServer = new BufferedReader(new  
    InputStreamReader(clientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

Création du flux d'entrée
attaché à la socket



```
outToServer.writeBytes(sentence + '\n');
```

```
modifiedSentence = inFromServer.readLine();
```

```
System.out.println("FROM SERVER: "+modifiedSentence);
```

```
clientSocket.close();
```

```
}
```

```
}
```

Interface de programmation TCP client Java

```
BufferedReader inFromServer = new BufferedReader(new  
    InputStreamReader(clientSocket.getInputStream()));
```

envoi d'une ligne
vers le serveur

```
sentence = inFromUser.readLine();
```

Création du flux d'entrée
attaché à la socket

```
outToServer.writeBytes(sentence + '\n');
```

```
modifiedSentence = inFromServer.readLine();
```

```
System.out.println("FROM SERVER: "+modifiedSentence);
```

```
clientSocket.close();
```

```
}
```

```
}
```

Interface de programmation TCP client Java

```
BufferedReader inFromServer = new BufferedReader(new  
    InputStreamReader(clientSocket.getInputStream()));
```

envoi d'une ligne
vers le serveur

```
sentence = inFromUser.readLine();
```

Création du flux d'entrée
attaché à la socket

```
outToServer.writeBytes(sentence + '\n');
```

```
modifiedSentence = inFromServer.readLine();
```

lecture d'une ligne
venant du serveur

```
System.out.println("FROM SERVER: "+modifiedSentence);
```

```
clientSocket.close();
```

```
}
```

```
}
```

Interface de programmation TCP serveur Java

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

class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {

            Socket connectionSocket = welcomeSocket.accept();

            BufferedReader inFromClient = new BufferedReader(new
                InputStreamReader(connectionSocket.getInputStream()));
```

Interface de programmation TCP serveur Java

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

class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {

            Socket connectionSocket = welcomeSocket.accept();

            BufferedReader inFromClient = new BufferedReader(new
                InputStreamReader(connectionSocket.getInputStream()));
```

Création d'une socket
de connexion



Interface de programmation TCP serveur Java

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

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception {  
        String clientSentence;  
        String capitalizedSentence;
```

Création d'une socket
de connexion

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while(true) {
```

Attente sur la socket de connexion
d'un contact de client

```
            Socket connectionSocket = welcomeSocket.accept();
```

```
            BufferedReader inFromClient = new BufferedReader(new  
                InputStreamReader(connectionSocket.getInputStream()));
```

Interface de programmation TCP serveur Java

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

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception {  
        String clientSentence;  
        String capitalizedSentence;
```

Création d'une socket
de connexion

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while(true) {
```

Attente sur la socket de connexion
d'un contact de client

```
            Socket connectionSocket = welcomeSocket.accept();
```

```
            BufferedReader inFromClient = new BufferedReader(new  
                InputStreamReader(connectionSocket.getInputStream()));
```

Création d'un flux entrant attaché
à la socket


Interface de programmation TCP serveur Java

```
DataOutputStream outToClient = new  
    OutputStream(connectionSocket.getOutputStream());  
  
clientSentence = inFromClient.readLine();  
  
capitalizedSentence = clientSentence.toUpperCase() + '\n';  
outToClient.writeBytes(capitalizedSentence);  
  
}  
}  
}
```

Interface de programmation TCP serveur Java

```
DataOutputStream outToClient = new  
    OutputStream(connectionSocket.getOutputStream());
```

Création du flux de sortie
attaché à la socket



```
clientSentence = inFromClient.readLine();
```

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
outToClient.writeBytes(capitalizedSentence);
```

```
}
```


```
}
```

```
}
```

Interface de programmation TCP serveur Java

```
DataOutputStream outToClient = new  
    OutputStream(connectionSocket.getOutputStream());
```

Création du flux de sortie
attaché à la socket



```
clientSentence = inFromClient.readLine();
```

Lecture d'une ligne depuis
la socket



```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
outToClient.writeBytes(capitalizedSentence);
```

```
}
```


```
}
```

```
}
```

Interface de programmation TCP serveur Java

```
DataOutputStream outToClient = new  
    DataOutputStream(connectionSocket.getOutputStream());
```

Création du flux de sortie
attaché à la socket



```
clientSentence = inFromClient.readLine();
```

Lecture d'une ligne depuis
la socket



```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
outToClient.writeBytes(capitalizedSentence);
```

Ecriture d'une ligne sur
la socket



```
}
```

```
}
```

```
}
```

Interface de programmation TCP serveur Java

```
DataOutputStream outToClient = new  
    OutputStream(connectionSocket.getOutputStream());
```

Création du flux de sortie
attaché à la socket

```
clientSentence = inFromClient.readLine();
```

Lecture d'une ligne depuis
la socket

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
outToClient.writeBytes(capitalizedSentence);
```

Ecriture d'une ligne sur
la socket

```
}  
}  
}
```

Fin de la boucle while, retour
et attente d'une autre
connexion cliente

Interface de programmation TCP

la classe `java.net.ServerSocket`

Pour les serveurs attendant des connexions clientes

Différents constructeurs :

```
ServerSocket s = new ServerSocket(8000);  
ServerSocket s = new ServerSocket(port, tailleFile);  
    si port=0 alors le système choisit automatiquement le port
```

Propriétés :

```
adresse IP : InetAddress getAddress();  
port : int getLocalPort();  
timeout : setSoTimeout, getSoTimeout
```

Méthodes :

```
attendre une connexion : Socket accept();  
fermer la socket : void close();  
affichage : String toString():  
    ServerSocket[addr=, port=, localport=8000]
```

Interface de programmation TCP

la classe `java.net.Socket`

Utilisée durant les communications TCP/IP

Différents constructeurs pour établir la connexion :

```
Socket s = new Socket("www.lifl.fr", 80);  
Socket s = new Socket(InetAddress, 8000);
```

Propriétés :

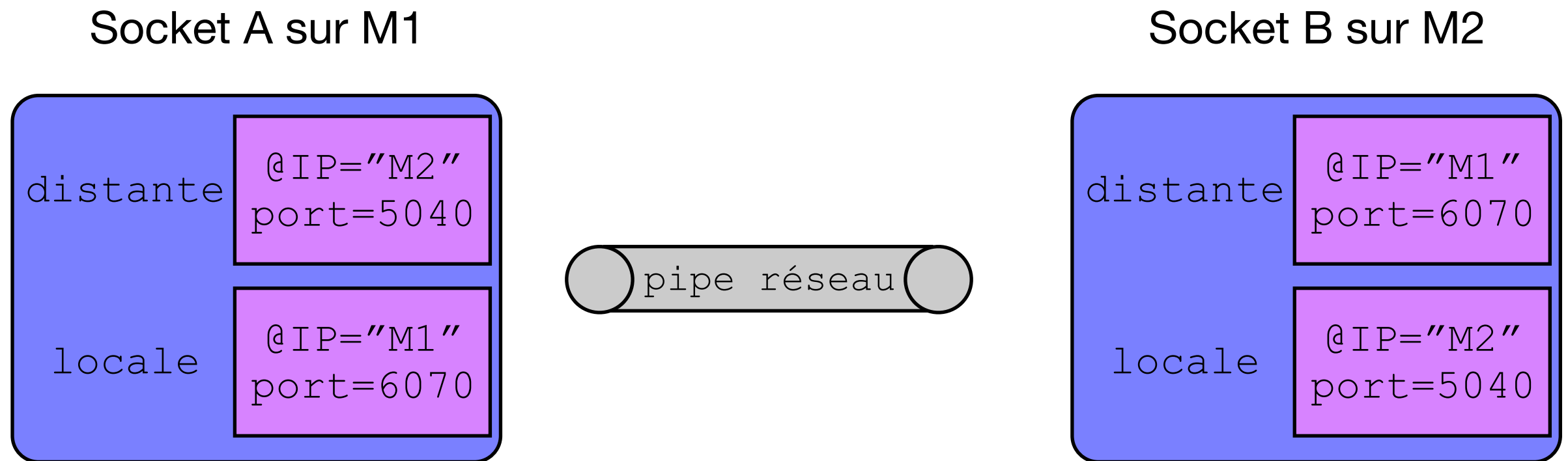
```
adresse IP : InetAddress getInetAddress(), getLocalAddress()  
port : int getPort(), getLocalPort()  
flux in : InputStream getInputStream()  
flux out : OutputStream getOutputStream()  
options set/get : TcpNoDelay, SoLinger, SoTimeout
```

Méthodes :

```
fermeture de la socket : void close()  
affichage : String toString()  
Socket[addr=www.lifl.fr/134.206.10.x,port=80,localport=50000]
```

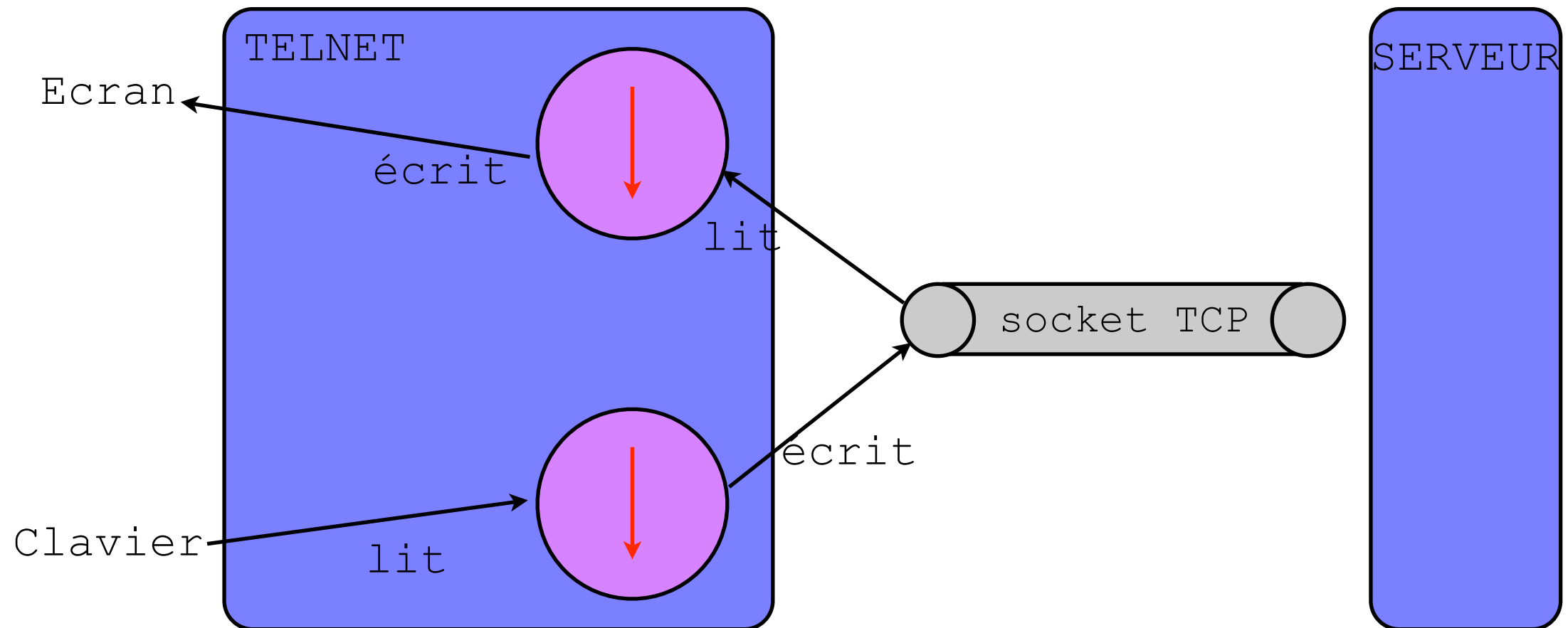
Interface de programmation TCP

la relation entre deux sockets connectées



<code>A.getInetAddress()</code>	<code>==</code>	<code>B.getLocalInetAddress()</code>
<code>A.getPort()</code>	<code>==</code>	<code>B.getLocalPort()</code>
<code>A.getLocalInetAddress()</code>	<code>==</code>	<code>B.getInetAddress()</code>
<code>A.getLocalPort()</code>	<code>==</code>	<code>B.getPort()</code>

Interface de programmation TCP l'application Telnet



Telnet : terminal texte pour dialoguer avec tout serveur TCP
utilisant un protocole ASCII

Interface de programmation TCP

quelques mots sur la classe `java.lang.Thread`

Permet de lancer des flots d'exécution simultanés au sein d'un même processus

- calcul parallèle, animation, . . .
- attente sur entrée/sortie bloquante

Méthodes

- démarrage : `void start();`
- arrêt : `void interrupt();`
- attente fin : `void join();`
- traitement : `void run();`
- . . .

Doit être héritée ou instanciée sur 1 objet implantant l'interface `java.lang.Runnable`

Interface de programmation TCP

le thread de redirection de flux

```
public class RedirigerFlux extends Thread {
    protected java.io.BufferedReader fluxLecture = null;
    protected java.io.PrintStream fluxEcriture = null;

    public RedirigerFlux (java.io.InputStream fl,
                          java.io.OutputStream fe) {
        fluxLecture = new java.io.BufferedReader(
            new java.io.InputStreamReader(fl));
        fluxEcriture = new java.io.PrintStream(fe);
        super.start();
    }

    public void run() {
        try { String s;
            while ( (s = fluxLecture.readLine()) != null )
                fluxEcriture.println(s);
        } catch (java.io.IOException err) {}
    }
}
```

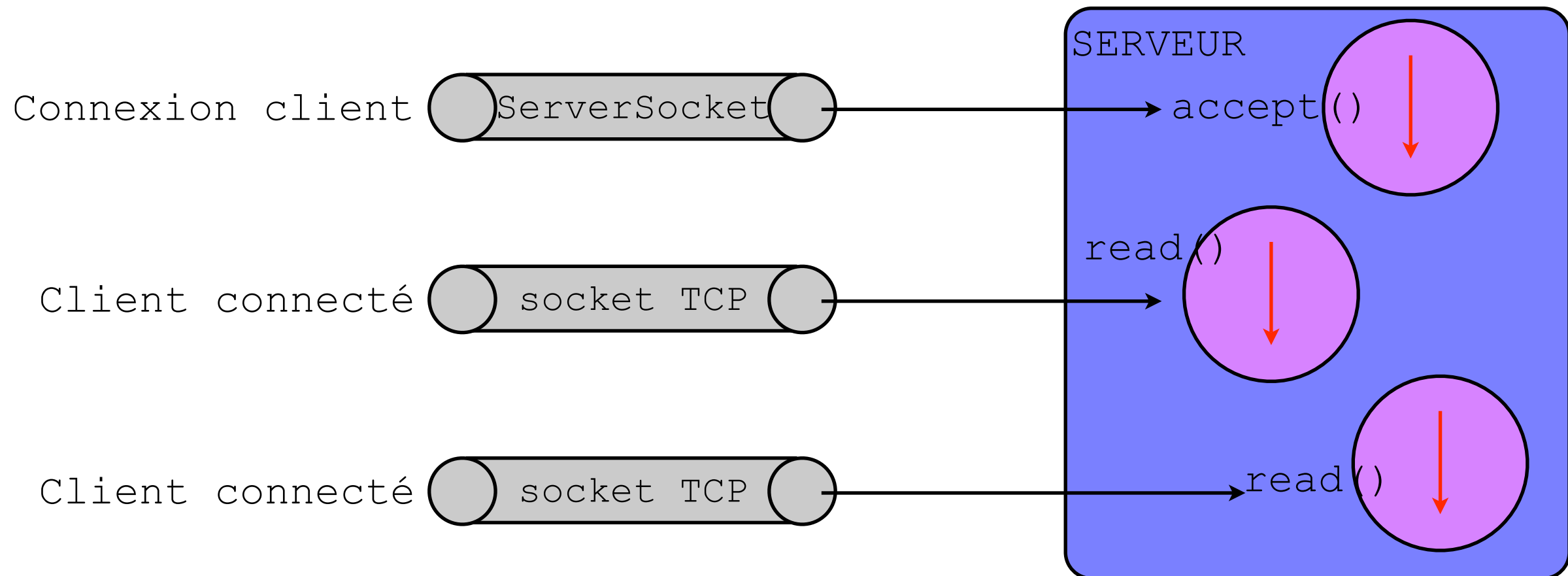
Interface de programmation TCP

l'application Telnet en Java

```
public class Telnet {  
    public static void main(String args[]) throws Exception  
    {  
        java.net.Socket socket = new java.net.Socket(machine, port);  
        // Création du thread redirigeant la socket vers l'écran.  
        RedirigerFlux socketVersEcran =  
            new RedirigerFlux(socket.getInputStream(), System.out);  
        // Création du thread redirigeant le clavier vers la socket.  
        RedirigerFlux clavierVersSocket =  
            new RedirigerFlux(System.in, socket.getOutputStream());  
        // Attendre la fin de lecture sur la socket.  
        socketVersEcran.join();  
        // Arrêter de lire le clavier.  
        clavierVersSocket.interrupt();  
    }  
}
```

Interface de programmation TCP

un serveur TCP/IP multi-clients



Les méthodes `accept()` et `read()` sont bloquantes : besoin de threads dans le serveur pour gérer simultanément plusieurs clients

Interface de programmation TCP

un serveur TCP/IP multi-threadé

```
import java.net.*;

public class ServeurMultiThreade {

    public static void main(String args[]) throws Exception
    {

        ServerSocket serveur = new ServerSocket(#port) ;

        while (true) {
            System.out.println(serveur + " en attente !");
            Socket client = serveur.accept() ;
            new GestionnaireClient(client) ;
        }

        serveur.close() ;
    }
}
```

Interface de programmation TCP

le thread gestionnaire des clients

```
public class GestionnaireClient extends Thread
{
    protected java.net.Socket client;

    public GestionnaireClient(java.net.Socket s)
    {
        client = s;
        super.start();
    }

    public void run()
    {
        try {
            // Traiter les requêtes clientes . . . puis
            client.close();
        } catch(Exception exc) { . . . }
    }
}
```

Interface de programmation UDP

UDP: pas de connexion entre client et serveur

Pas d'acquittement

L'émetteur indique systématiquement l'adresse IP et le no de port

Le serveur doit extraire du message l'adresse IP et le no port de l'émetteur

Les données peuvent être reçues dans le désordre ou perdues

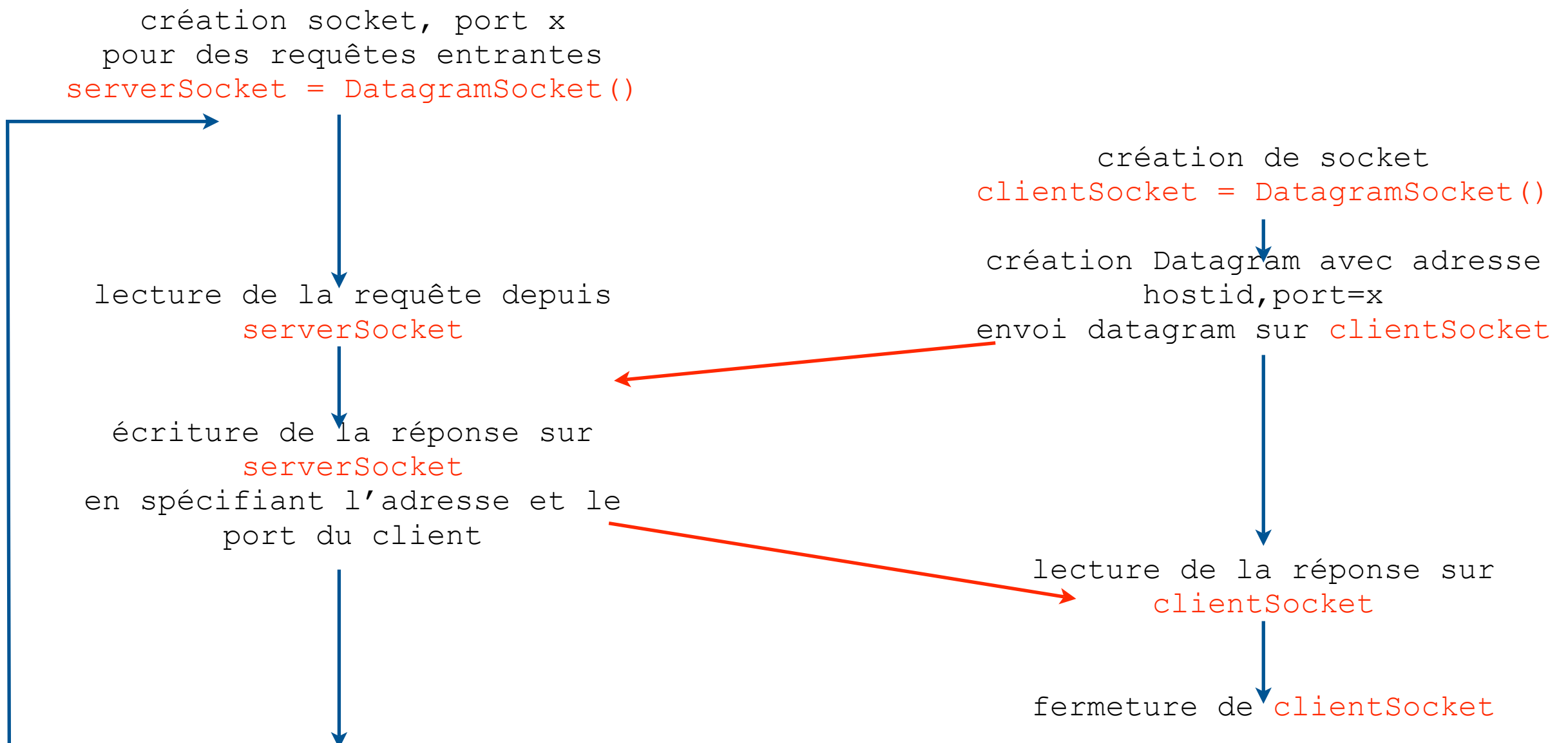
Point de vue application

UDP fournit un transfert non fiable de groupes d'octets ("datagrammes") entre client et serveur

Interface de programmation UDP

Serveur sur hostid

Client



Interface de programmation UDP client Java

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

class UDPClient {
    public static void main(String args[]) throws Exception {

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress = InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

Interface de programmation UDP client Java

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

class UDPClient {
    public static void main(String args[]) throws Exception {

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress = InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
    }
}
```

Création du flux entrant



Interface de programmation UDP client Java

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

Création du flux
entrant

```
class UDPClient {
    public static void main(String args[]) throws Exception {
```

Création
d'une
socket
client

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

```
        DatagramSocket clientSocket = new DatagramSocket();
```

```
        InetAddress IPAddress = InetAddress.getByName("hostname");
```

```
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
```

```
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

Interface de programmation UDP client Java

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

Création du flux entrant

```
class UDPClient {
    public static void main(String args[]) throws Exception {
```

Création
d'une
socket
client

```
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
```

```
        DatagramSocket clientSocket = new DatagramSocket();
```

```
        InetAddress IPAddress = InetAddress.getByName("hostname");
```

```
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
```

Traduction du nom hôte
en adresse IP

```
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

Interface de programmation UDP client Java

```
DatagramPacket sendPacket=  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, 9876);  
  
clientSocket.send(sendPacket);  
  
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
  
clientSocket.receive(receivePacket);  
  
String modifiedSentence =  
    new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

Interface de programmation UDP client Java

Création datagramme avec
données,
longueur, adr. IP, port



```
DatagramPacket sendPacket=  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, 9876);
```

```
clientSocket.send(sendPacket);
```

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);
```

```
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());
```

```
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();
```

```
}
```

```
}
```

Interface de programmation UDP client Java

Création datagramme avec
données,
longueur, adr. IP, port

```
DatagramPacket sendPacket=  
    new DatagramPacket(sendData, sendData.length,  
        IPAddress, 9876);
```

```
clientSocket.send(sendPacket);
```

Envoi datagramme vers le serveur

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());
```

```
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();
```

```
}  
}
```


Interface de programmation UDP client Java

Création datagramme avec
données,
longueur, adr. IP, port

```
DatagramPacket sendPacket=  
    new DatagramPacket(sendData, sendData.length,  
        IPAddress, 9876);
```

```
clientSocket.send(sendPacket);
```

Envoi datagramme vers le serveur

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

Lecture datagramme depuis le serveur

Interface de programmation UDP serveur java

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

class UDPServer {
    public static void main(String args[]) throws Exception {

        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true) {

            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);

            serverSocket.receive(receivePacket);
```

Interface de programmation UDP serveur java

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

```
class UDPServer {
    public static void main(String args[]) throws Exception {
```

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

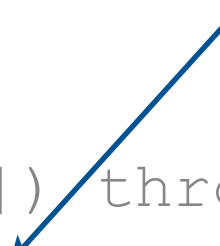
```
        byte[] receiveData = new byte[1024];
        byte[] sendData    = new byte[1024];
```

```
        while(true) {
```

```
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
```

```
            serverSocket.receive(receivePacket);
```

Création
Socket datagramme
Sur le port 9876



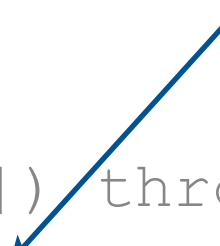
Interface de programmation UDP

serveur java

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

```
class UDPServer {
    public static void main(String args[]) throws Exception {
```

Création
Socket datagramme
Sur le port 9876



```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

```
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
```

Création espace pour
recevoir datagramme



```
        while(true) {
```

```
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
```

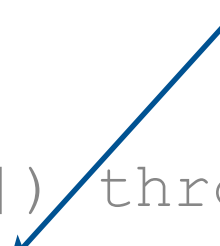
```
            serverSocket.receive(receivePacket);
```

Interface de programmation UDP serveur java

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

```
class UDPServer {
    public static void main(String args[]) throws Exception {
```

Création
Socket datagramme
Sur le port 9876



```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

```
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
```

Création espace pour
recevoir datagramme



```
        while(true) {
```

```
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
```

```
            serverSocket.receive(receivePacket);
```

Réception du
datagramme



Interface de programmation UDP serveur java

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();
sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length,
                       IPAddress, port);

    serverSocket.send(sendPacket);
}
}
}
```

Interface de programmation UDP serveur java

Adresse IP et numéro
de port de
l'émetteur



```
String sentence = new String(receivePacket.getData());
```

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();
```

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, port);
```

```
serverSocket.send(sendPacket);
```

```
}
```

```
}
```

```
}
```

Interface de programmation UDP serveur java

Adresse IP et numéro
de port de
l'émetteur

```
String sentence = new String(receivePacket.getData());
```

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();
```

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, port);
```

```
serverSocket.send(sendPacket);
```

```
}
```

```
}
```

```
}
```

Création datagramme pour
envoi vers le client

Interface de programmation UDP serveur java

Adresse IP et numéro
de port de
l'émetteur

```
String sentence = new String(receivePacket.getData());
```

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();
```

Envoi du
datagramme

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, port);
```

```
serverSocket.send(sendPacket);
```

Création datagramme pour
envoi vers le client

Interface de programmation UDP serveur java

Adresse IP et numéro
de port de
l'émetteur

```
String sentence = new String(receivePacket.getData());
```

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();
```

Envoi du
datagramme

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, port);
```

```
serverSocket.send(sendPacket);
```

Création datagramme pour
envoi vers le client

Fin du while,
retour et attente d'un
autre datagramme

Interface de programmation UDP

La classe `java.net.DatagramPacket`

Représente un paquet UDP

tampon de données, longueur, adresse IP, port

Constructeurs

`DatagramPacket(buffer, taille)`

`DatagramPacket(buffer, taille, InetAddress, port)`

Propriétés

Buffer : `byte[]` `getData()`

Taille buffer : `int` `getLength()`

Adresse IP : `InetAddress` `getAddress()`

Port IP : `int` `getPort()`

Modification : `setData`, `setLength`, `setAddress`, `setPort`

Interface de programmation UDP

La classe `java.net.DatagramPacket`

Un émetteur crée un `DatagramPacket` avec les données, la longueur, l'adresse et le numéro de port du destinataire

```
byte[] tampon = new byte[8096];  
ia = InetAddress.getByName("www.lifl.fr");  
DatagramPacket dp = new DatagramPacket(tampon, tampon.length, ia, 1000);
```

Un récepteur alloue le tampon pour recevoir le paquet et le passe au constructeur de `DatagramPacket`

```
byte[] tampon = new byte[8096];  
dp = new DatagramPacket(tampon, tampon.length);
```

A la réception, le `DatagramPacket` contient les données, l'adresse et le port de l'émetteur

Le tampon peut être réutilisé pour plusieurs envois / réceptions

Interface de programmation UDP

la classe `java.net.DatagramSocket`

Une prise pour pouvoir communiquer en UDP
peut être connectée à une socket paire

Constructeurs

socket anonyme : `DatagramSocket()`

socket avec port : `DatagramSocket(1000)`, `DatagramSocket(1000, ip)`

Propriétés

l'adresse locale : `InetAddress getLocalAddress()`

l'adresse distante : `InetAddress getAddress()`

le port local : `int getLocalPort()`

le port distant : `int getPort()`

Méthodes

connexion : `void connect(InetAddress ia, int port)`

émission : `void send(DatagramPacket dp)`

réception : `void receive(DatagramPacket dp)`

fermeture : `void close()`

Interface de programmation Multicast IP applications

Il s'agit de délivrer un message à un ensemble de destinataires.

C'est plus efficace que l'envoi d'un message à chaque destinataire :
pris en charge par les routeurs

Les destinataires sont identifiés par une unique adresse de groupe
indépendante de la localisation des destinataires

adressage logique des groupes : RFC 966 et 1112

classe D : 224.x.x.x - 239.x.x.x

mais l'adresse est choisie à la main !

Les membres du groupe peuvent changer à tout moment : primitives
d'adhésion et de sortie d'un groupe

Interface de programmation Multicast IP

quelques exemples d'applications

Transmission de vidéo sur Internet : MBONE

Jeux répartis multi-utilisateurs temps réel

Systèmes de fichiers distribués
sans connaître la localisation physique
réplication, tolérances aux pannes

Services de localisation de ressources
rechercher une ressource WWW
le serveur le plus proche répond

Bref beaucoup d'applications intéressantes !

Interface de programmation Multicast IP

la communication en Multicast IP

Récepteur

création socket multicast, port x
pour réception de messages

```
s = MulticastSocket(x)  
s.joinnGroup(@IP)
```

lecture du message depuis s

```
p = DatagramPacket(buf)  
s.receive(p)
```

traiter p

Emetteur

création de socket

```
s = MulticastSocket()
```

création Datagram avec adresse
ip, port=x et données

```
p = DatagramPacket(data, @ip, x)
```

envoi datagram sur s

```
s.send(p)
```

fermeture de la socket

```
clientSocket
```


Interface de programmation Multicast IP serveur java

```
import java.net.*;

public class ServeurMulticastIP {

    public static void main (String args[]) throws Exception {

        InetAddress mcast = InetAddress.getByName("225.1.1.1");

        MulticastSocket ms = new MulticastSocket(8000);

        ms.joinGroup (mcast);
        while (true) {
            DatagramPacket msg = new DatagramPacket(
                                                new byte[512], 512);
            ms.receive(msg);
            System.out.println(msg.getAddress() + ":" +
                               msg.getPort() + " a envoyé " +
                               new String(msg.getData()));
        } } }
```

Interface de programmation Multicast IP client java

```
import java.net.*;
public class ClientMulticastIP {

    public static void main (String args[]) throws Exception {
        String message = "Hello world!";

        MulticastSocket ms = new MulticastSocket();

        InetAddress mcast = InetAddress.getByName("225.1.1.1");

        ms.setTimeToLive(1);

        DatagramPacket dp = new DatagramPacket(
            message.getBytes(), message.length(), mcast, 8000);

        for (int i=0; i<10; i++) ms.send(dp);

        ms.close();
    }
}
```

Interface de programmation Multicast IP

la classe `java.net.MulticastSocket`

Prise pour communiquer en Multicast IP

Sous classe de `java.net.DatagramSocket`

Nouvelles méthodes

joindre un groupe : `void joinGroup (InetAddress mcast);`

quitter un groupe : `void leaveGroup (InetAddress mcast);`

émission : `void send(DatagramPacket dp);`

Diffusion contrôlable :

champ time-to-live (TTL)

`Void setTimeToLive(int ttl) et int getTimeToLive()`

TTL = nombre de réseaux à franchir

TTL = 0 : même machine

TTL = 1 : même sous-réseau

TTL = 2..254 : selon topologie

TTL = 255 : aucune restriction !

Synthèse sur les sockets TCP/IP

API simple et fiable

peu de primitives orientées communication

La programmation des clients est différente de celle des serveurs.

asymétrie des primitives de communication

De nombreuses questions à se poser avant d'utiliser les primitives !

Rien pour la structuration de l'application !

Seulement approprié pour des applications réparties simples (1 client vers 1 serveur) ou comme brique de base pour des couches middleware plus évoluées

par ex. : WWW, FTP, email, news, . . .

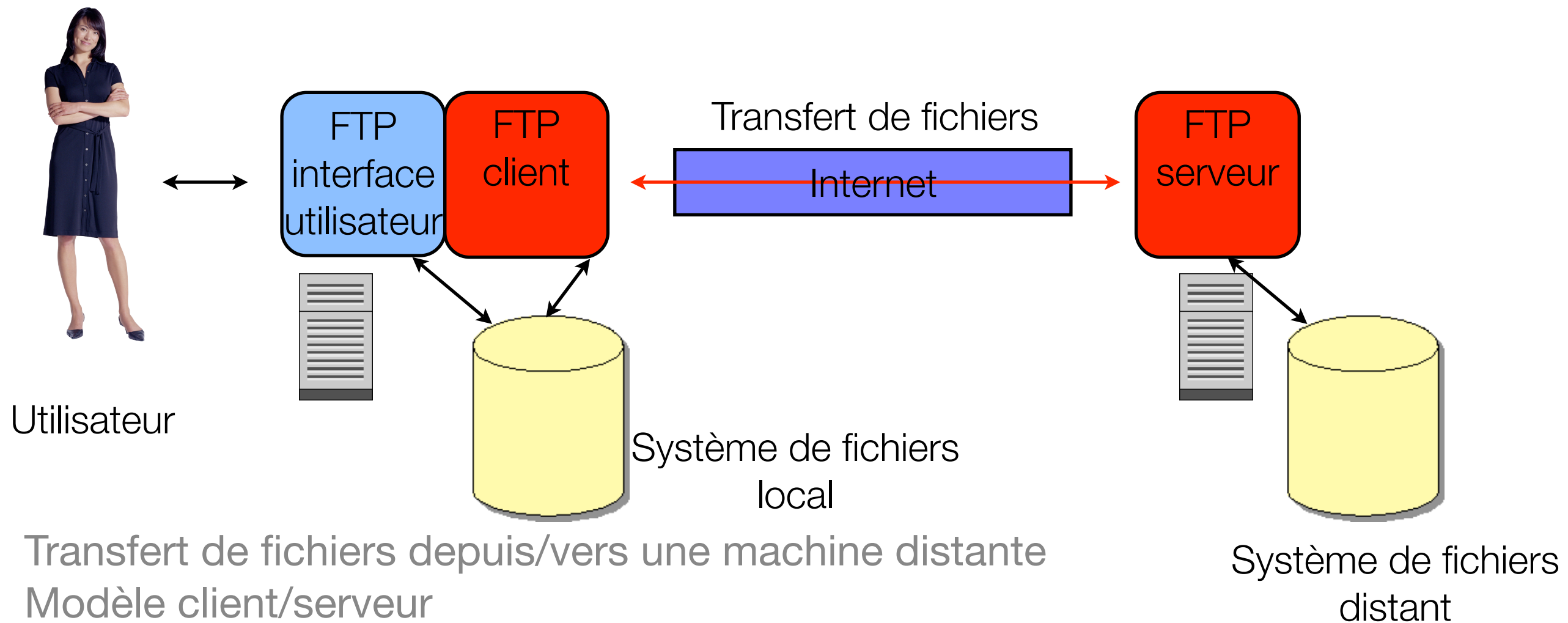
Les applications client/serveur d'internet

FTP : File Transfer Protocol

SMTP : Simple Mail Transfer Protocol

DNS : Domain Name System

FTP : le protocole de transfert de fichiers



Transfert de fichiers depuis/vers une machine distante

Modèle client/serveur

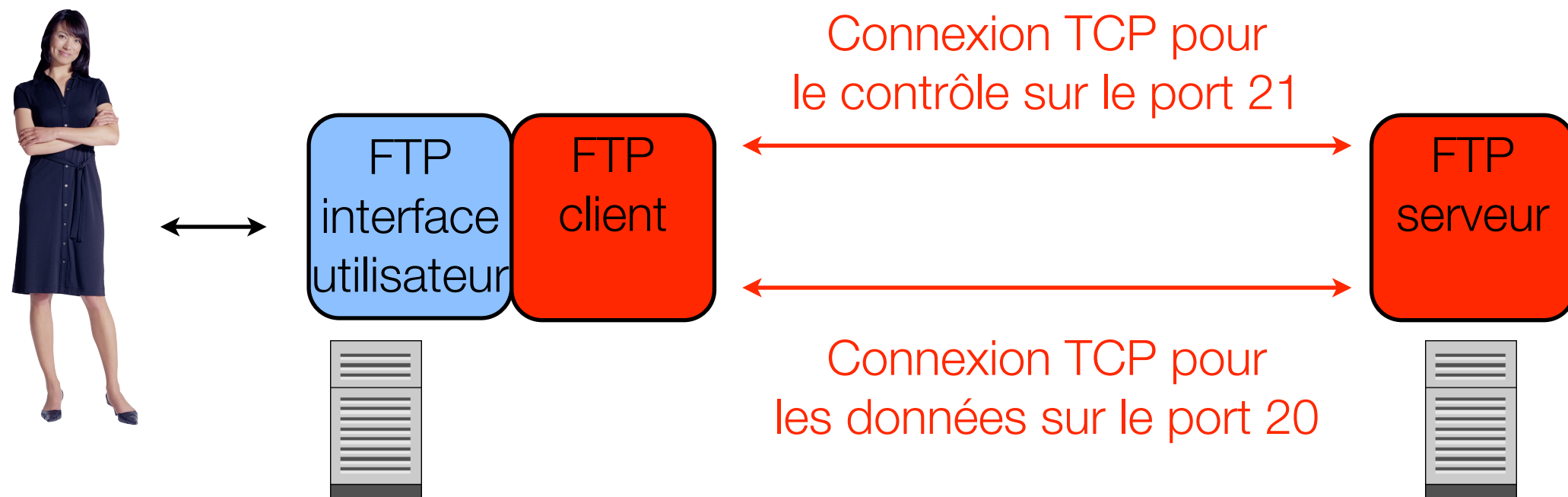
client : initie le transfert

serveur : hôte distant

ftp : RFC 959

ftp : écoute sur le port 21

FTP : le protocole de transfert de fichiers



Le contrôle et les données sont sur des connexions séparées.

Le client contacte le serveur ftp sur le port 21.

Deux canaux sont ouverts en parallèle :

canal de contrôle : échange de commandes/réponses entre le client et le serveur

canal de données : transfert de données depuis/vers le serveur

Le serveur maintient **un état** vis-à-vis du client : répertoire courant, authentification, mode de transfert, ...

FTP : le protocole de transfert de fichiers

Les commandes

Elles sont envoyées en ASCII sur le canal de contrôle.

USER username

PASS password

LIST

liste des fichiers du répertoire courant sur le serveur

RETR filename

demande d'un fichier

STOR filename

stocke un fichier sur la machine distante

Les réponses

Elles sont composées d'un code d'état suivi de texte et envoyées sur le canal de contrôle.

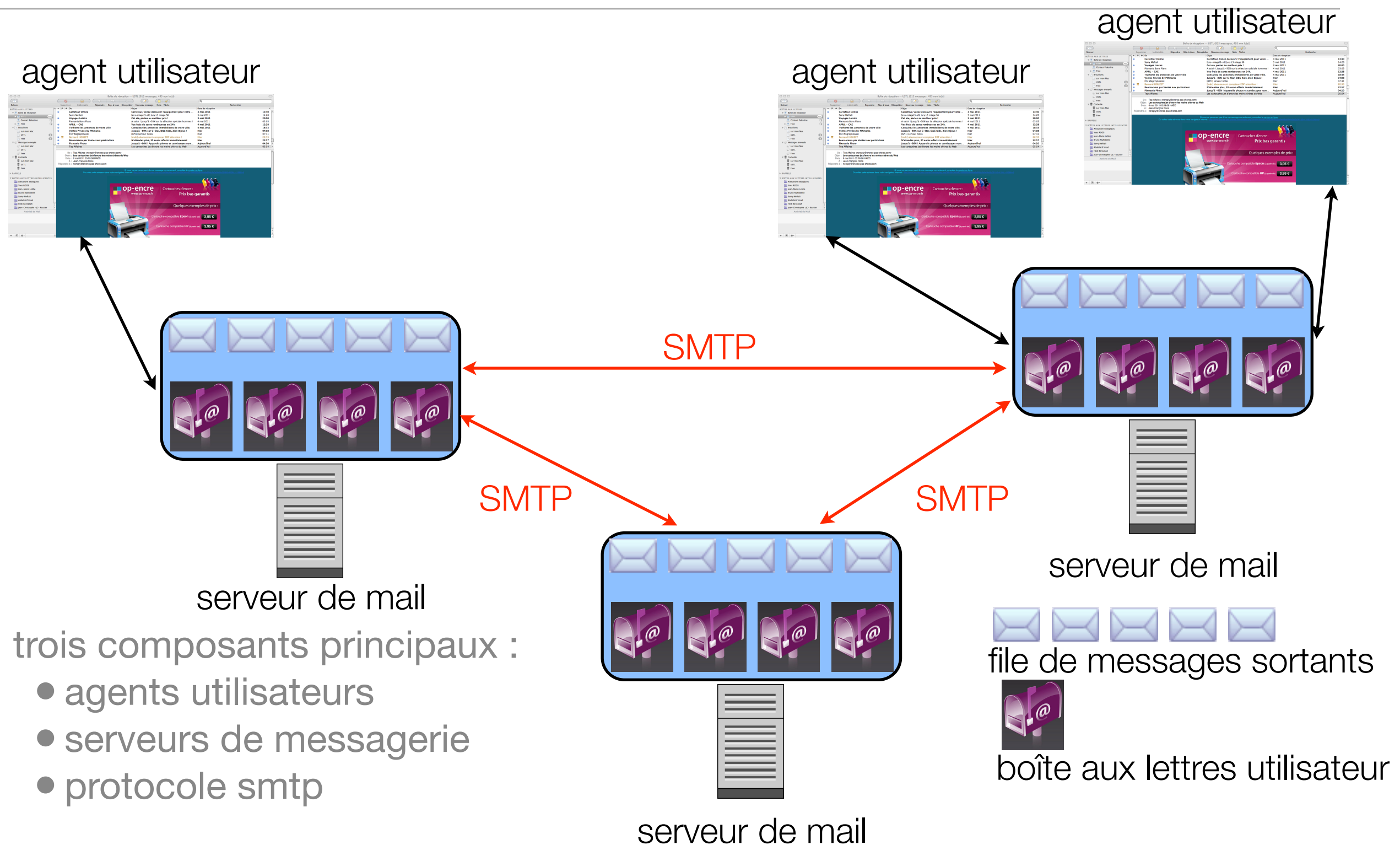
331 Username OK, password required

125 data connection already open; transfer starting

425 Can't open data connection

452 Error writing file

Courrier électronique



trois composants principaux :

- agents utilisateurs
- serveurs de messagerie
- protocole smtp

Courrier électronique

L'agent utilisateur

c'est un lecteur de message

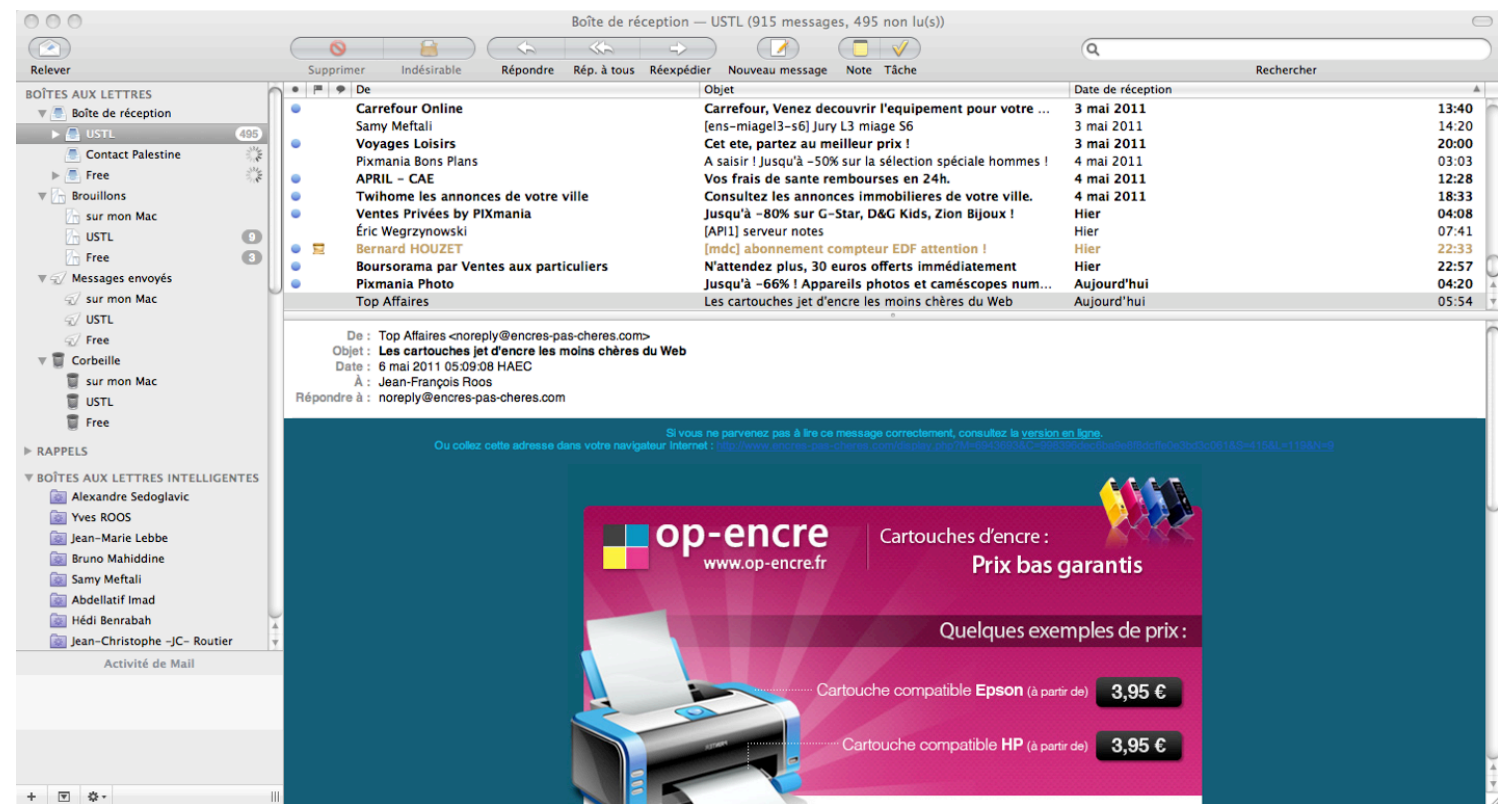
il permet la composition, la lecture et l'édition de messages électroniques

e.g :

Mail

Outlook

Thunderbird



Gère les messages entrants et sortants stockés sur le serveur

Courrier électronique

Les serveurs de messagerie



BAL contient les messages entrants (à lire) pour l'utilisateur



File de messages (messages sortants) à envoyer

Protocole smtp entre les serveurs de messagerie pour l'envoi des messages

Courrier électronique

Le protocole smtp

Il utilise tcp pour transférer des messages entre serveurs de messagerie.
Le transfert est direct du serveur émetteur vers le serveur récepteur.

trois phases :

- connexion

- transfert de messages

- déconnexion

L'interaction est sous forme commande/réponse :

- commande** : texte ASCII

- réponse** : code état et phrase

Les messages doivent être en 7-bit ASCII

smtp : RFC 821

smtp : écoute sur le port 25

Courrier électronique

Une simple conversation SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:   How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Courrier électronique

Le format des messages

RFC 821 (SMTP) : définit le protocole pour l'échange de messages

RFC 822 : standard définissant le format des messages :

entête, e.g.,

To:

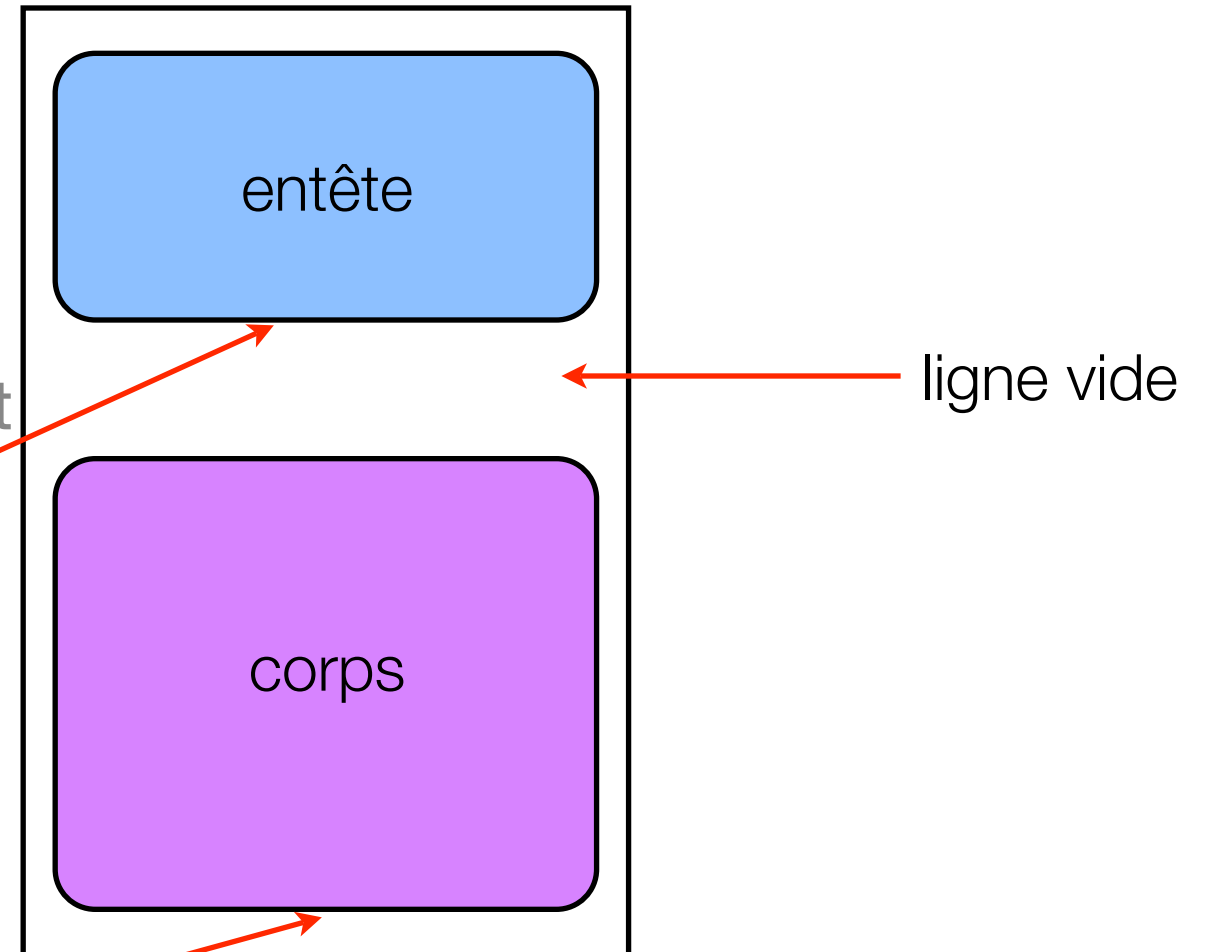
From:

Subject:

différents des commandes smtp!

corps

le "message", caractères ASCII 7 bits seulement



Courrier électronique

Le format des messages : extension multimédia

MIME : RFC 2045, 2056

Lignes supplémentaires dans l'entête du message pour déclarer que le type du contenu est de type MIME

Version MIME

Méthode utilisée pour coder les données

Type de données multimédia/sous-type

Déclaration de paramètres

Données encodées

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.....base64 encoded data
```

Courrier électronique

Content-Type: paramètres type/subtype

Texte

Exemples de sous-types plain, html

Image

Exemples de sous-types jpeg, gif

Audio

Exemples de sous-types : basic (8-bite mu-law encoded), 32kadpcm (32 kbps coding)

Video

Exemples de sous-types mpeg, quicktime

Application

Autres données qui doivent être traitées directement par un lecteur
Exemples de sous-types msword, octet-stream

Courrier électronique

exemple multipart

From: alice@crepes.fr

To: bob@hamburger.edu

Subject: Picture of yummy crepe.

MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=98766789

--98766789

Content-Transfer-Encoding: quoted-printable

Content-Type: text/plain

Dear Bob,

Please find a picture of a crepe.

--98766789

Content-Transfer-Encoding: base64

Content-Type: image/jpeg

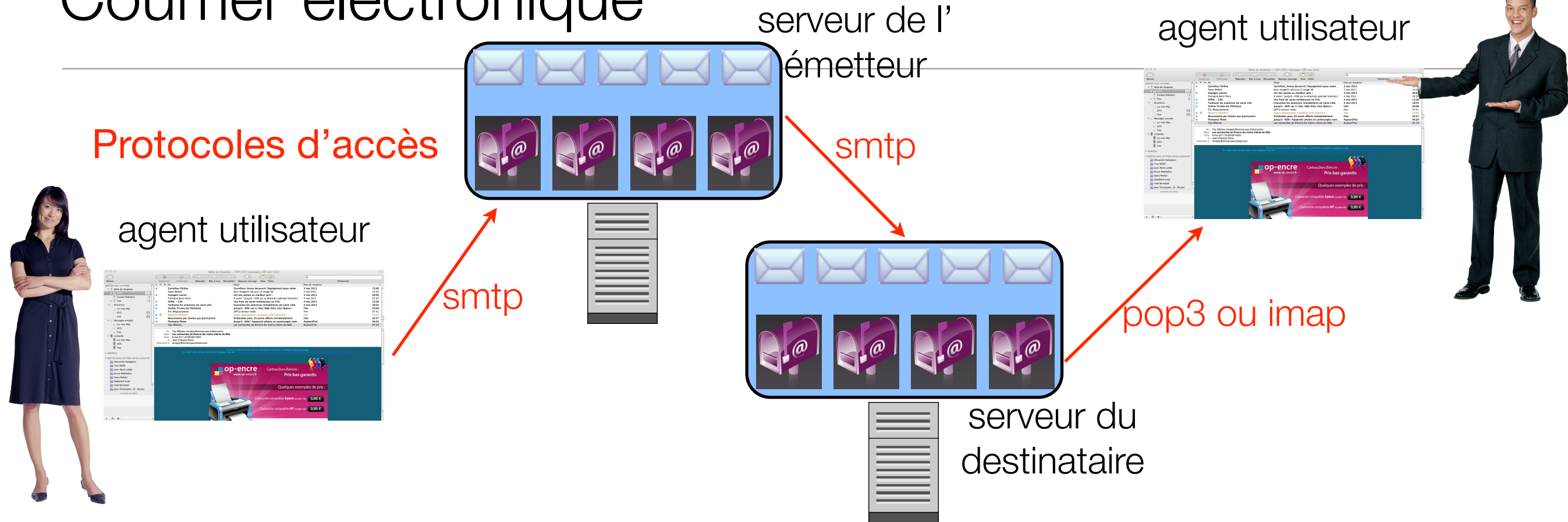
base64 encoded data

.....

.....base64 encoded data

--98766789--

Courrier électronique



SMTP: envoie les messages au serveur du destinataire

Les protocoles d'accès : aller chercher les messages sur le serveur

- POP: Post Office Protocol [RFC 1939]
identification (agent <--> serveur) et chargement
- IMAP: Internet Mail Access Protocol [RFC 1730]
plus de caractéristiques
manipulation des messages stockés sur le serveur
- HTTP: webmail, zimbra,...

Courrier électronique

Le protocole POP3

Phase d'autorisation

Commandes client :

user: nom

pass: password

Réponses serveur

+OK

-ERR

Phase de transaction

client:

list: liste les no de messages

retr: recherche message par no

dele: efface

quit

S: +OK POP3 server ready

C: user alice

S: +OK

C: pass hungry

S: +OK user successfully logged

C: list

S: 1 498

S: 2 912

S: .

C: retr 1

S: <message 1 contents>

S: .

C: dele 1

C: retr 2

S: <message 1 contents>

S: .

C: dele 2

C: quit

S: +OK POP3 server signing off

Domain Name System

Une personne possède plusieurs identificateurs : no secu, nom, no passeport

Hôtes, routeurs internet :

Adresse IP (32 bit) – utilisée pour acheminer les datagrammes

nom symbolique, e.g., chomolungma.lifl.fr – utilisé par les humains

Q: correspondance entre noms et adresse IP ?

Domain Name System :

base de données répartie réalisée par une hiérarchie de serveurs de noms

protocole de la couche application : hôte, routeurs, serveurs de noms

doivent résoudre les noms (traduction nom/adresse) pour communiquer

c'est une fonction cœur de l'internet

Domain Name System

Pourquoi ne pas centraliser le DNS?

- point sensible (panne)
- volume du trafic
- base de données centralisée distante
- maintenance
- pas de passage à l'échelle!

Pas de serveurs qui connaissent toutes les traductions nom/IP

Serveur de noms local

- Chaque domaine a un serveur de noms local (dit par défaut)
- Une requête DNS est dirigée en premier vers le serveur local

Serveur de noms faisant autorité

- Pour un hôte: celui qui stocke l'adresse et le nom de l'hôte
- Peut exécuter la traduction nom/adresse pour ce nom d'hôte

Domain Name System

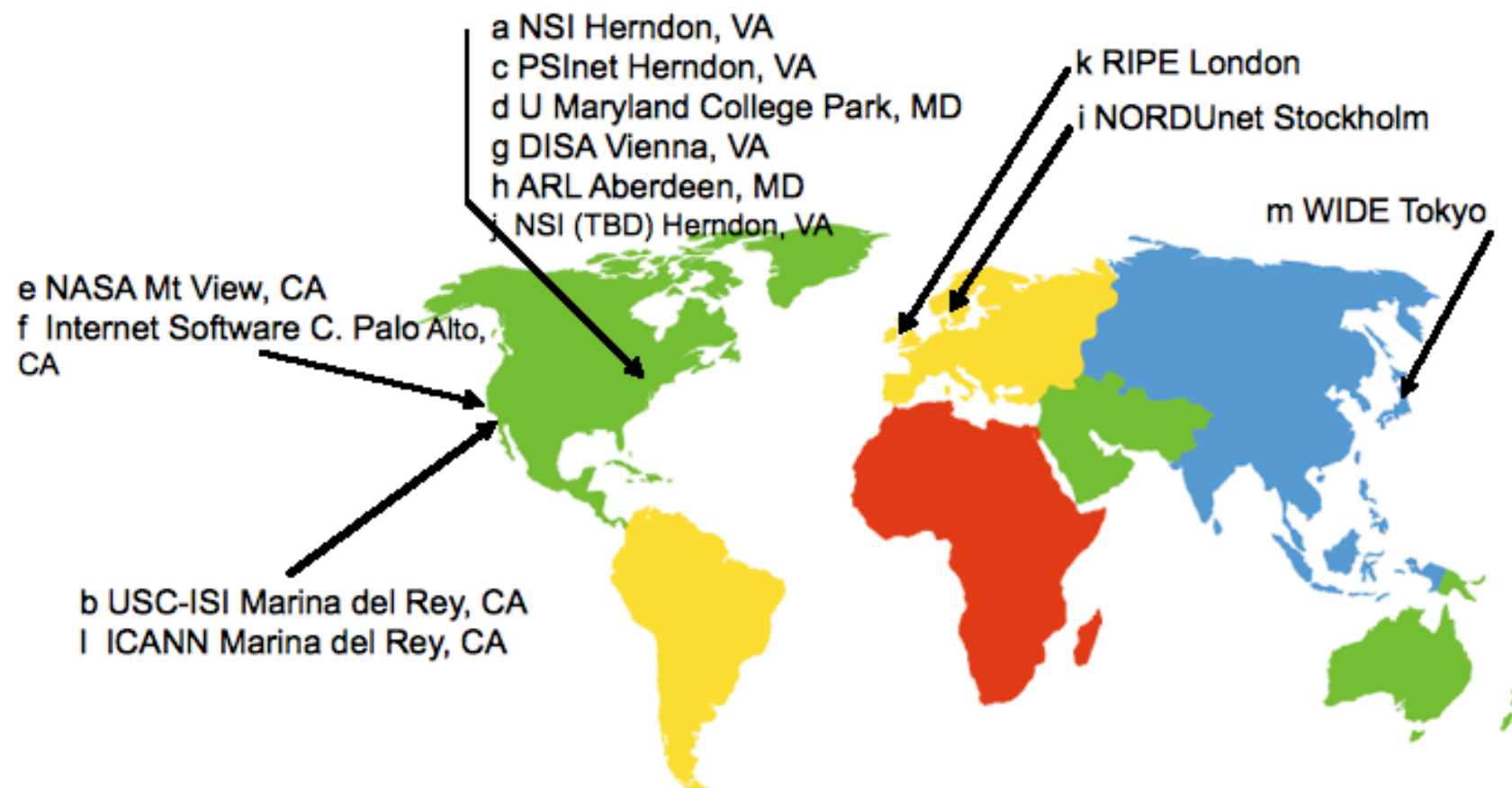
Les serveurs racines DNS

contactés par le serveur de noms local qui ne peut pas résoudre un nom

Serveur racine:

Contacte le serveur de noms faisant autorité si la traduction n'est pas connue

Fait la traduction



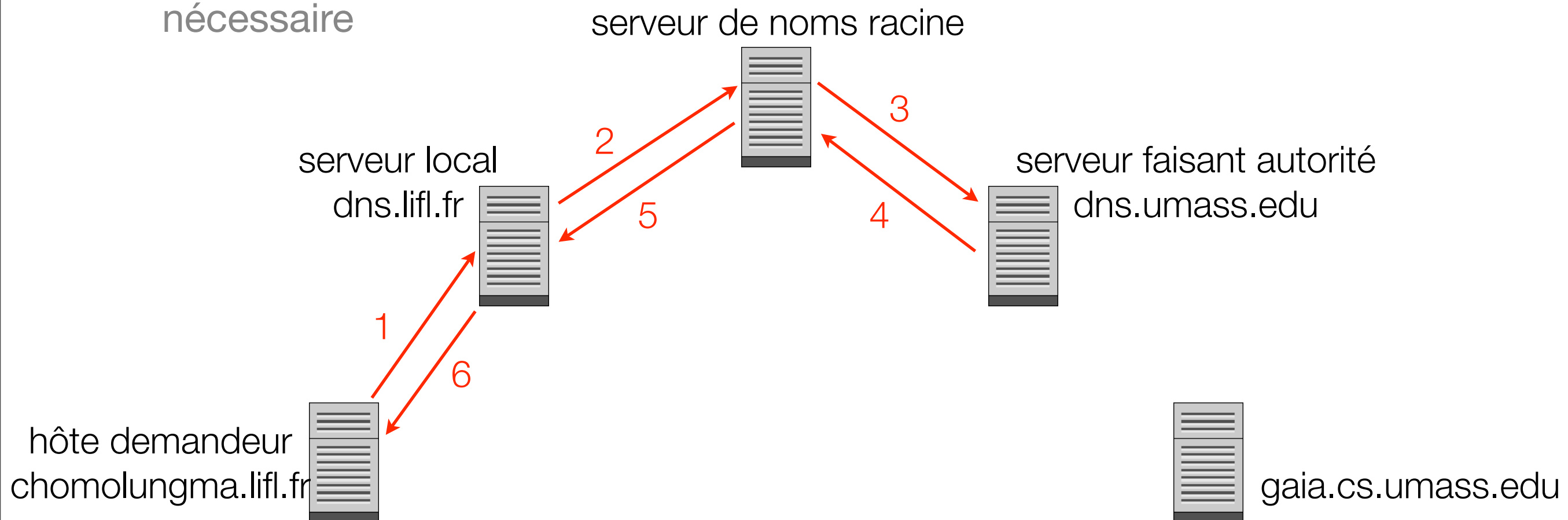
13 serveurs racines
dans le monde

Domain Name System

Un exemple simple DNS

chomolungma.lifl.fr veut l'@ IP de gaia.cs.umass.edu

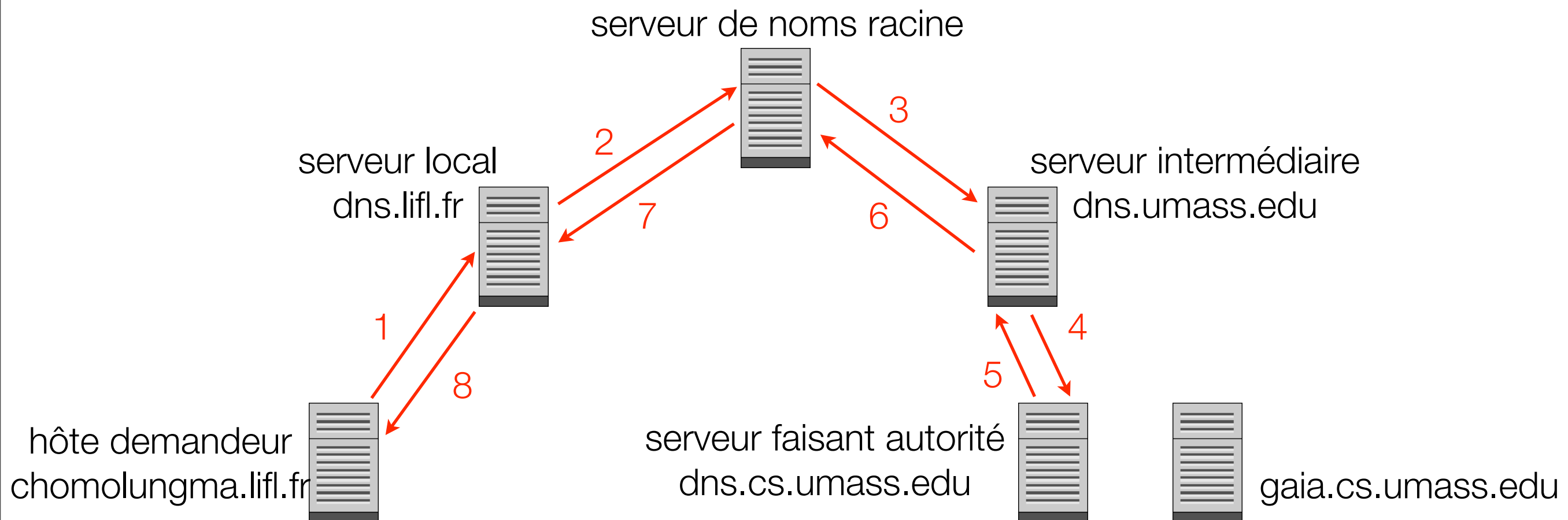
1. contacte son serveur DNS local dns.lifl.fr
2. dns.lifl.fr contacte le serveur racine, si nécessaire
3. Le serveur racine contacte le serveur faisant autorité, dns.umass.edu, si nécessaire



Domain Name System

Un exemple DNS

Serveur racine : peut ne pas connaître le serveur de noms faisant autorité
Peut connaître un serveur de noms intermédiaire qu'il contacte pour trouver le serveur de noms faisant autorité

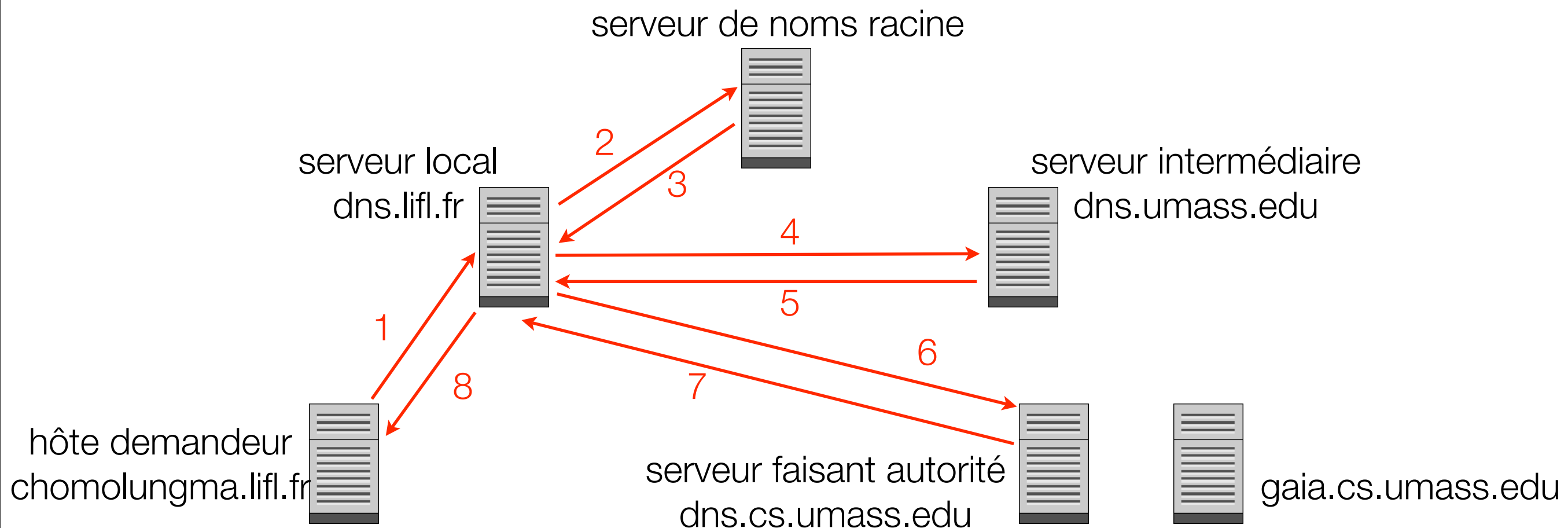


Domain Name System

Requête récursive : déplace la résolution de nom vers le serveur contacté

Charge ? Etat ?

Requête itérative : le serveur contacté donne en réponse le serveur suivant à contacter, “je ne connais pas ce nom, mais essayez avec ce serveur”



Domain Name System

Cache et mise à jour des données

Dès qu'un serveur de noms apprend une traduction @IP/nom, il la stocke dans son cache

Les entrées du cache sont associées à un timer et disparaissent après un certain temps

Les mécanismes de mises à jour/notification sont définies dans le RFC 2136

Domain Name System

Les enregistrements DNS

DNS : BD répartie qui gère des ressources (RR)

Format RR : (name, value, type, ttl)

Type=A

name est un nom d'hôte

value est une @ IP

Type=NS

name est un nom de domaine (e.g. foo.com)

value est une @ IP du serveur faisant autorité pour ce domaine

Type=CNAME

name est le nom alias pour certains noms canoniques

e.g. www.ibm.com est en réalité servereast.backup2.ibm.com

value est le nom canonique

Type=MX

value et le nom du serveur de messagerie associé à name

Domain Name System

Le protocole DNS : les requêtes et les réponses ont le même format

Entête msg

identification: 16 bits # pour requête,
les réponses aux requêtes utilisent
le même #

flags:

requête ou réponse

souhaite récursion

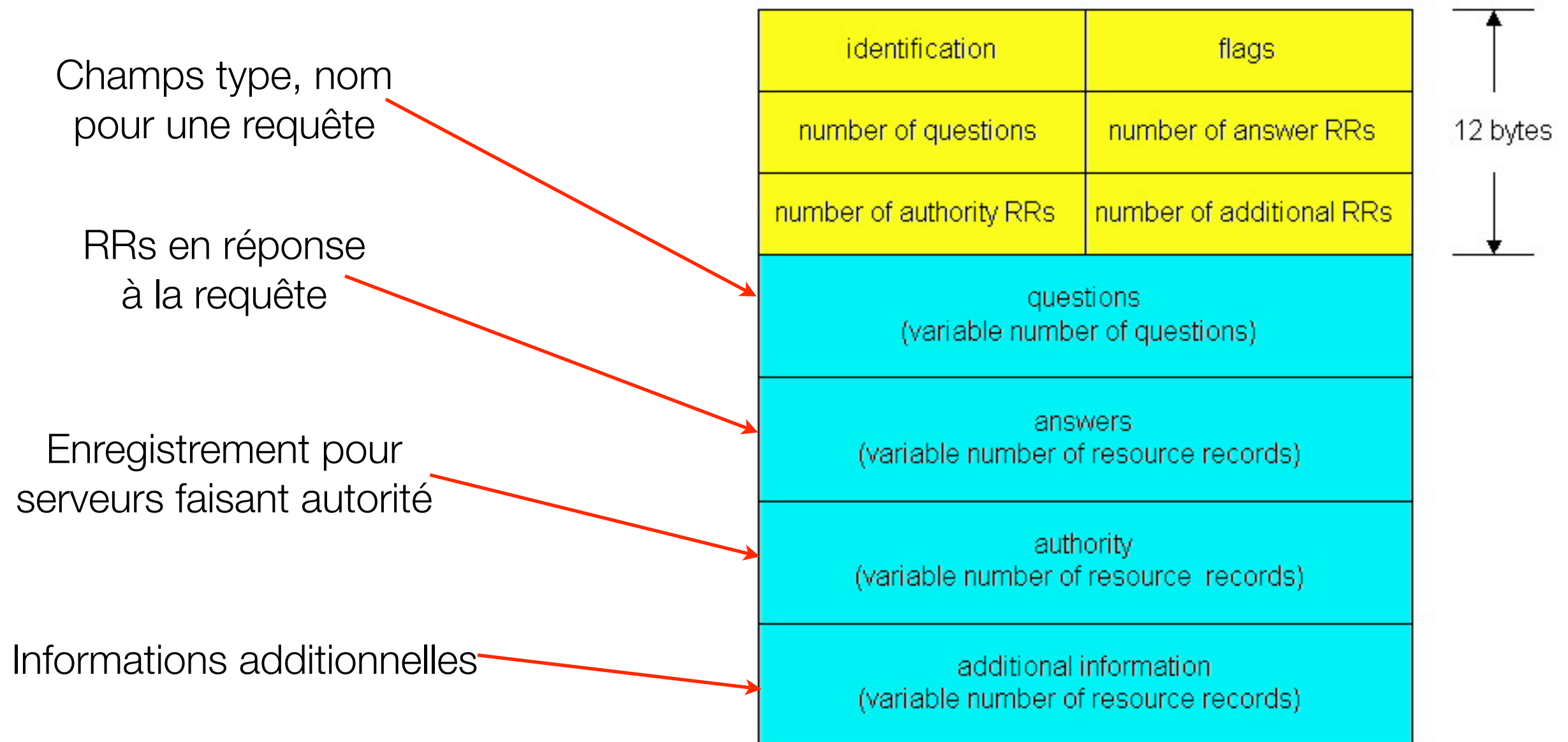
recursion disponible

réponse par serveur faisant autorité

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

↑
12 bytes
↓

Domain Name System



Conclusion

Ce qu'il faut retenir

Échange requête /réponse

- Un client demande des infos ou un service

- Le serveur répond avec des données et un code d'état

Les formats de message:

- entête : champs donnant des informations sur les données

- données : informations devant être communiquées

Messages de contrôle vs données

- dans ou hors bande

Centralisé vs. décentralisé

Avec ou sans état

Transfert de messages fiable ou non fiable

Sécurité: authentification