La programmation réseau

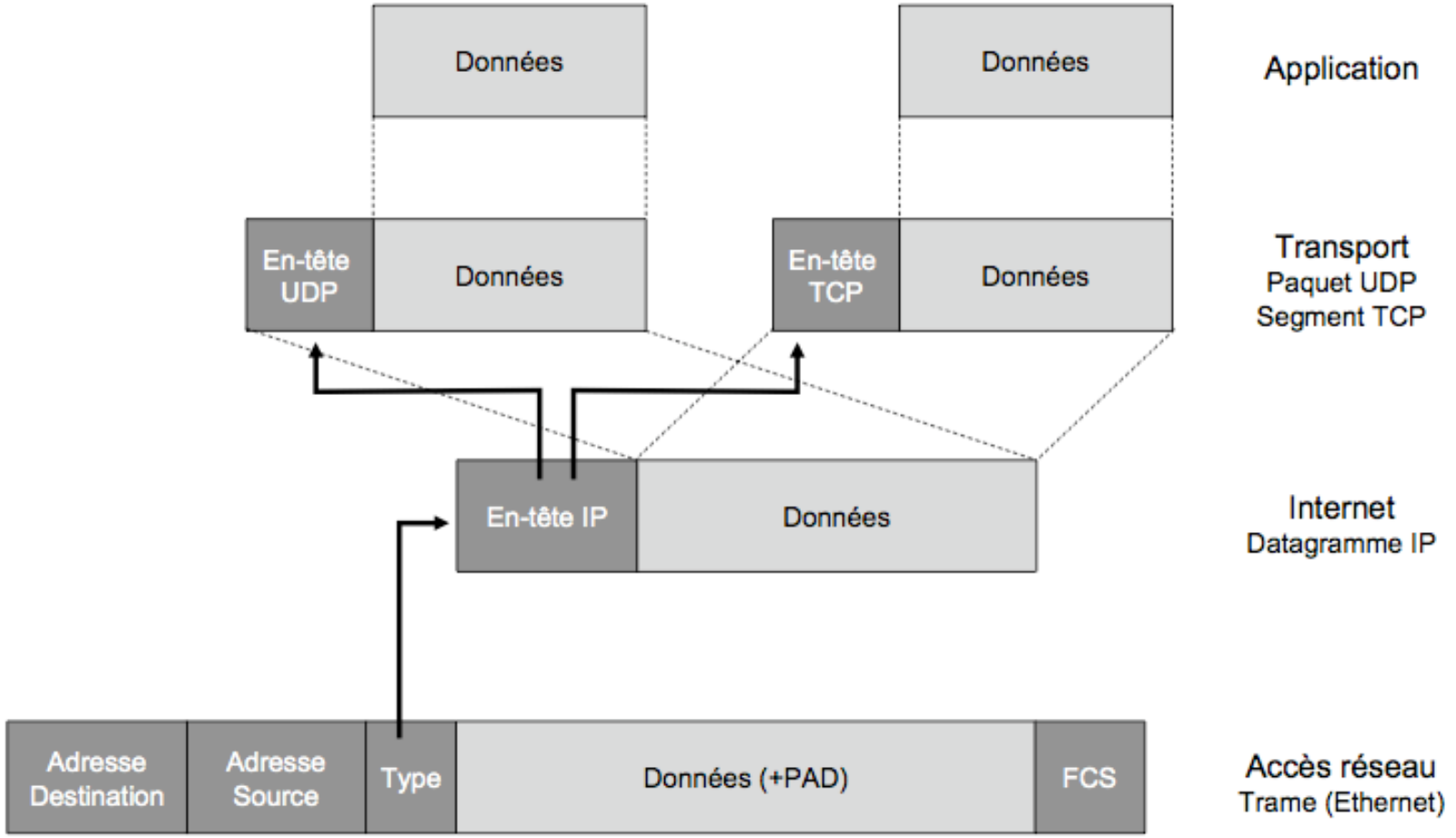
Java

Sommaire

La programmation réseau

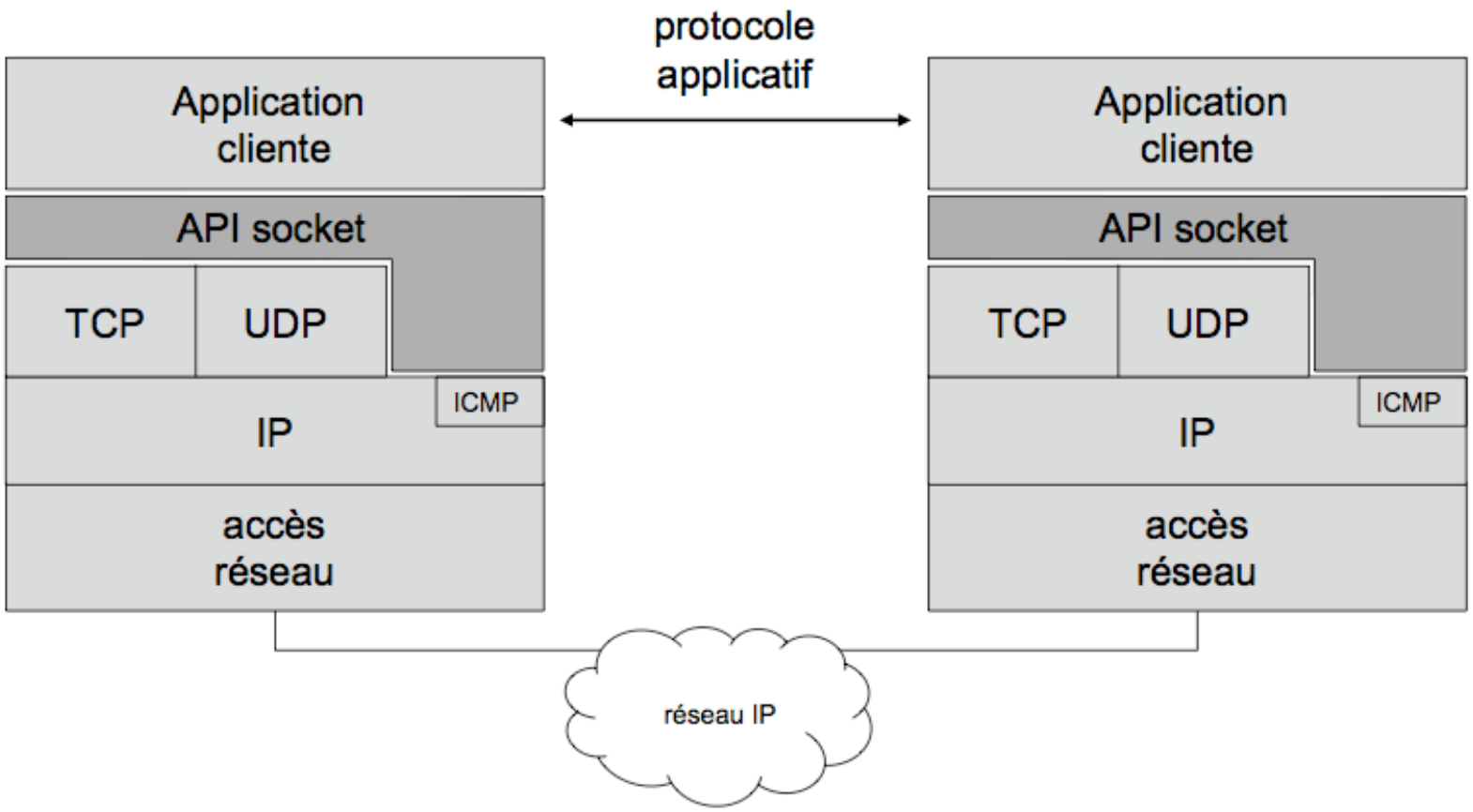
- *Rappel Réseau*
  - *Couches réseau et TCP-IP*
  - *Notion de Socket*
  - *Notion de Port*
- *La classe URL*
- *Les Sockets Java*
- *TCP/IP et les principaux services*
- *La classe InetAdress*
- *Processus de Sockets*
- *Exemple de process serveur*
- *Exemple de process client simple et multi-clients*
- *Datagram- Buffered stream- et Data Stream- sockets*

Rappel couches réseau et TCP/IP



Source Christine Bulfone : Le client/Serveu©r et l'API socket

Présentation de L'API Socket



Source Christine Bulfone : Le client/Serveu©r et l'API socket

La notion de « socket »

- *API (Application Program Interface) socket*
  - Mécanisme d'interface de programmation
  - permet aux programmes d'échanger des données
  - Les application ne voient les couches de communication qu'à travers l'API socket (abstraction)
  - n'implique pas forcément une communication par le réseau
  - le terme « socket » signifie douille, prise électrique femelle

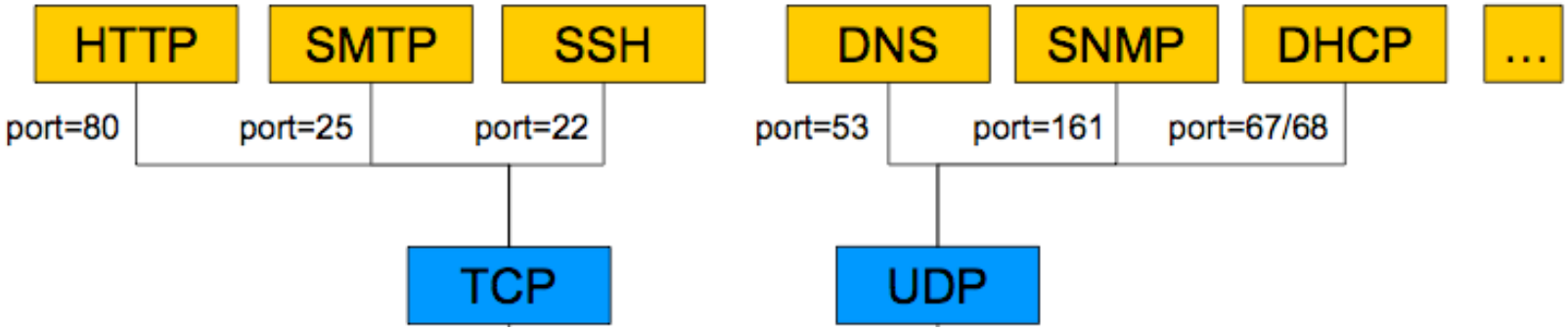
- *Une connexion est entièrement définie sur chaque machine par :*
  - *le type de protocole (UDP ou TCP)*
  - *l'adresse IP*
  - *le numéro de port associé*
    - *au processus serveur : port local sur lequel les connexions sont attendues*
    - *client : allocation dynamique par le système*

Source Christine Bulfone : Le client/Serveu©r et l'API socket

La notion de « port »

*Un service rendu par un programme serveur sur une machine est accessible par un **port***

- *Un port est identifié sur une machine par un nombre entier (16 bits)*
- *de 0 à1023 :*
  - *ports **réservés** assignés par l'IANA (Internet Assigned Numbers Authority)*
  - donnent accès aux services standard :



- > 1024
  - ports **utilisateurs** disponibles pour placer un service applicatif quelconque

Source Christine Bulfone : Le client/Serveu©r et l'API socket

java.net
La classe URL

- accès à une URL pour le Web
  `://Hote[:port]/cheminDacces`
- getFile(), getHost(), getPort(), getProtocol(), getRef() retournent les champs de l'URL
- Les données référencée par l'URL peuvent être téléchargée de 3 façons :
  - par un objet **`URLConnection`** créé par **`openConnection()`**
  - par un objet **`InputStream`** créé par **`openStream()`**
  - par l'appel à **`getContents()`** qui retourne directement le contenu
- pour plus de contrôle utiliser la classe **`URLConnection`**
  - getContent-Length-Type-Encoding...

```
/** la Classe java.net.URL :
* encaps. Uniform Resource Locator
*/
public final class URL
extends Java.lang.Object
implements serealizable
{   ...
    public final Object getContent()
      throws IOException;
    public URLConnection
      openConnection();
      throws IOException;
    public final InputStream
      openStream();
      throws IOException;
    public String toString();
}
```

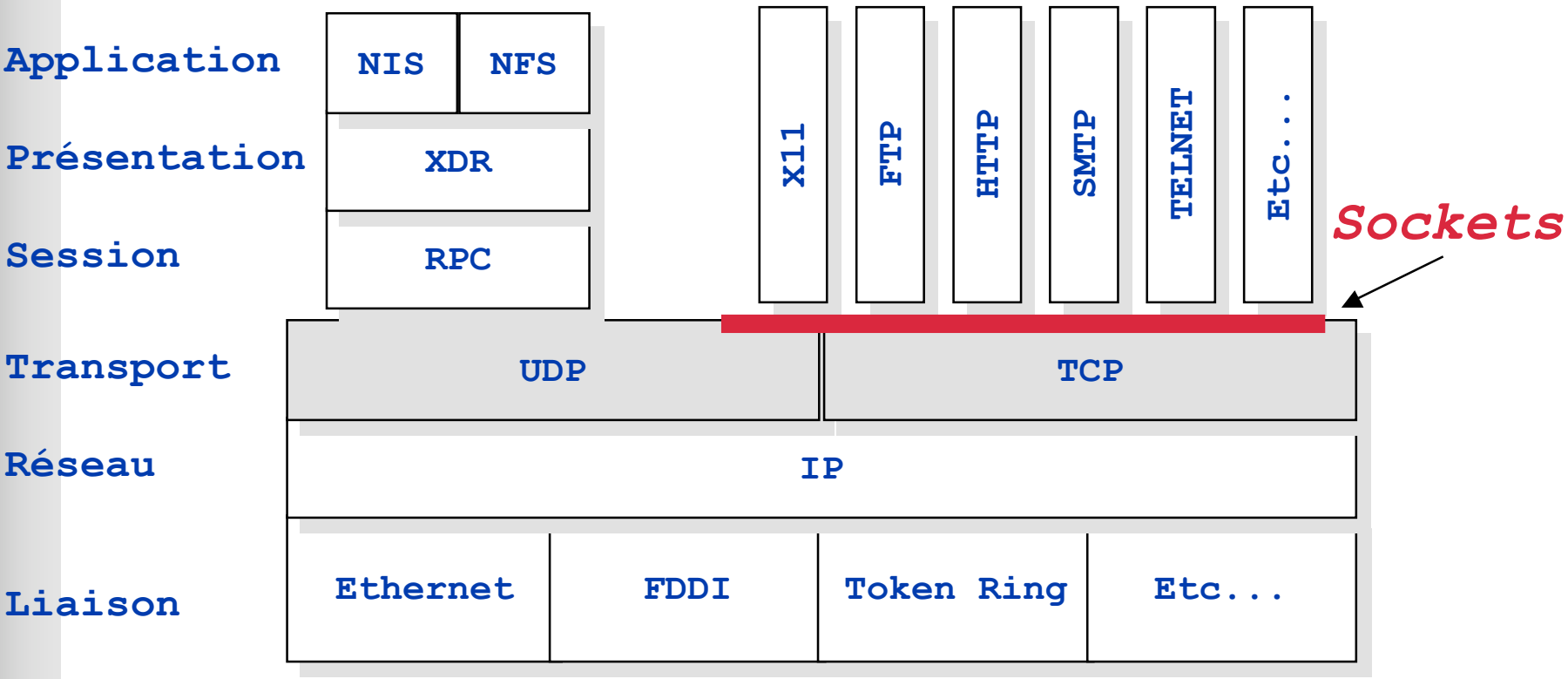
```
try
  URL lt = new URL ("http://...");
catch (MalformedURLException e) {
//...traitement d'erreur
}
```

java.net
Sockets Java

- Sockets:
  - communication peer-ro-peer
  - possède un nom et une adresse réseau
- Les types de sockets :
  - Datagram sockets
    - interface pour l'accès UDP (User Datagram Protocol)
    - transmissions de paquets indépendants "sans " garantie
  - Stream sockets
    - interface pour l'accès TCP (Transmission Control Protocol)
    - guarantie de transmission...
  - Raw sockets
    - interface au protocol IP (Internet Protocol) ou ICMP (Internet Control Message Protocol)
    - tests de nouveaux protocoles ou fcts évoluées

- Utilité des "Datagram sockets"
  - **multicast** :
    envoi de paquets à un groupe (ex. NetBios)
    - Il existe une classe (java 1.1) `MulticastSocket` dérivée de `DatagramSocket`
  - **broadcast** :
    envoi de paquets à l'ensemble des entités d'un réseau
  - **situation de découverte** :
    découvrir des entités du réseaux, nouveau service disponible
  - **messagerie non critique :**
    envoi de messages de contrôle d'activité non critiques "Je suis en vie"

java.net
TCP/IP et les principaux services



java.net
La classe InetAdress

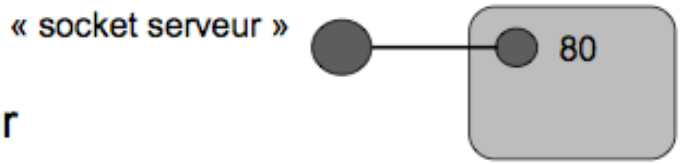
- faire plus que télécharger un objet référencé par une URL.
  - écrire un serveur par exemple...
- accès à une adresse sur TCP/IP
  - Net_id.Host_id.Port_ID
    206.24.45.100
    "java.sun.com"
- Cette classe permet de se connecter à un port d'un Host Internet et d'écrire/lire des données des classes java.io.
- Pas de constructeur
  - utilisation de getLocalHost(), getByName() ou getAllByName()
- tests sans le réseau...

```
//retourn le "localhost"
addr = InetAddress.getByName(null);
```

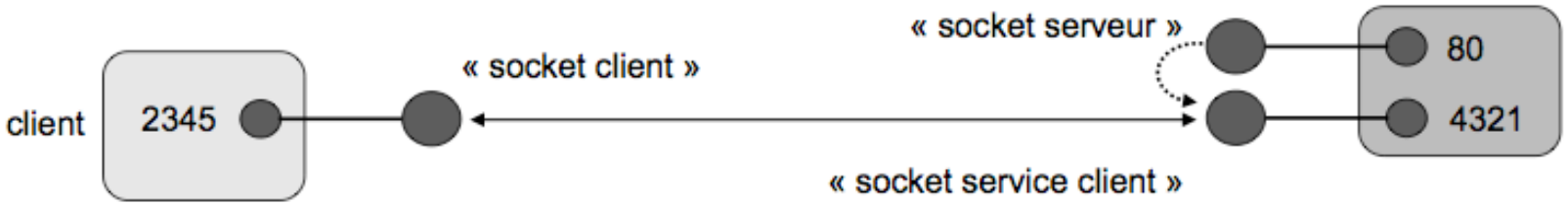
```
/** la Classe InetAdress:
* encapsulation d'une adresse IP
*/
public final class InetAdress
extends Java.lang.Object
{   // Methodes de classes
    public static InetAdress
      getByName(String host)
      throws UnknownHostException;
    // ttes les @IP associées à un Host
    public static InetAdress []
      getAllByName(String host)
      throws UnknownHostException;
    public static InetAdress
      getLocalHost(String host)
      throws UnknownHostException;
    // Méthodes d'instances
    public boolean equals (Object obj);
    public byte[] getAdress();
    public getHostName();
    public hashCode();
    public String toString();
}
```

Processus de Sockets

1. Le serveur crée une « socket serveur » (associée à un port) et se met en attente



2. Le client se connecte à la socket serveur
   - Deux sockets sont alors créés
     - Une « socket client » côté client
     - Une « socket service client » côté serveur
   - Ces sockets sont connectées entre elles



3. Le client et le serveur communiquent par les sockets
   - L'interface est celle des fichiers (read, write)
   - La socket serveur peut accepter de nouvelles connexions

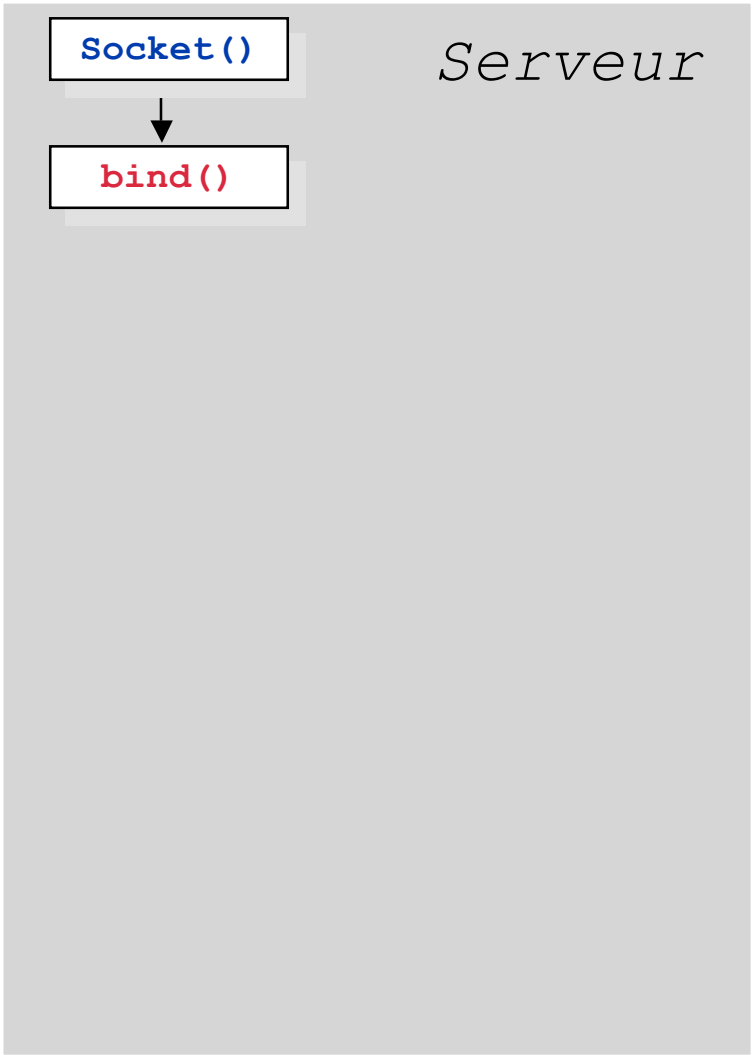
Source Christine Bulfone : Le client/Serveu©r et l'API socket

java.net
Processus de Sockets - 1

- creation et initialisation d'une "Socket" : appel de **`socket(...)`**

java.net
Processus de Sockets - 2

- *Établissement d'un service sur un port de la machine serveur pour être visible du réseau :* **`bind(...)`**



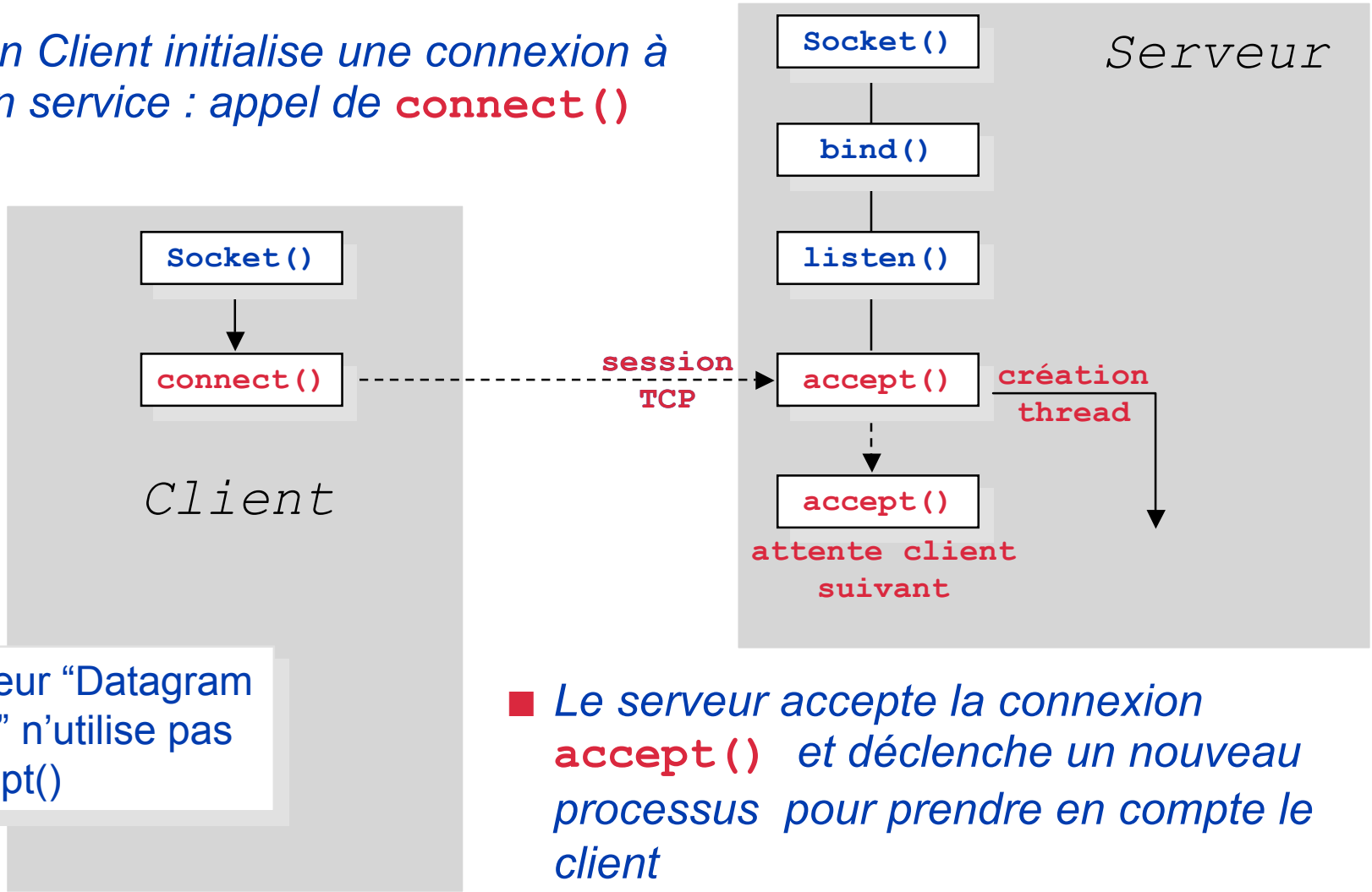
java.net
Processus de Sockets - 3

- *attente de connexions, le serveur est prêt : appel de* **`listen(...)`**

un serveur “Datagram sockets” n’utilise pas de listen() : il n’y a pas de déroulement de sessions

java.net
Processus de Sockets - 4

- *Un Client initialise une connexion à un service : appel de* **`connect()`**

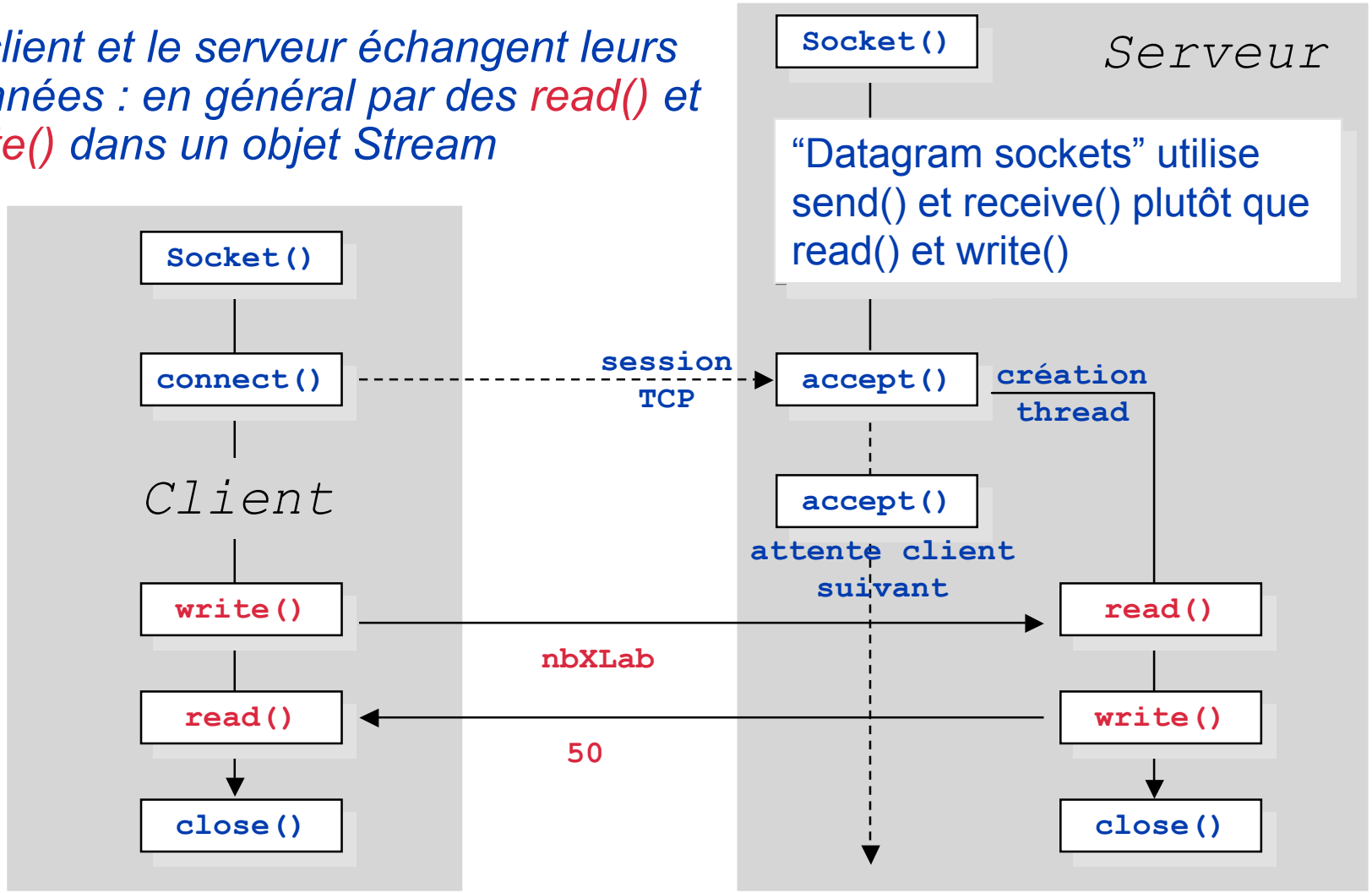


un serveur “Datagram sockets” n’utilise pas de accept()

- *Le serveur accepte la connexion* **`accept()`** *et déclenche un nouveau processus pour prendre en compte le client*

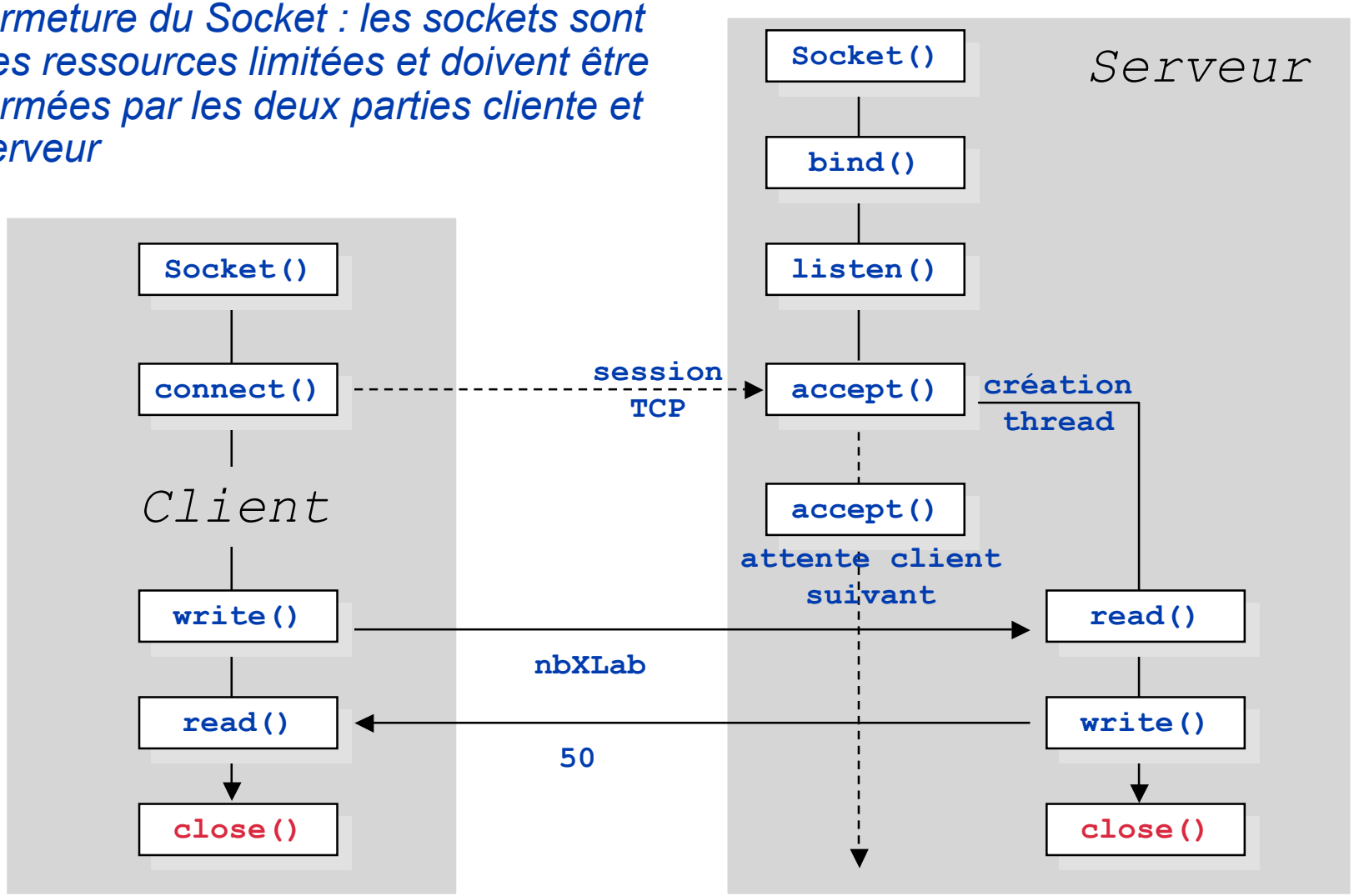
java.net
Processus de Sockets - 5

- *le client et le serveur échangent leurs données : en général par des read() et write() dans un objet Stream*



Processus de Sockets - 6

- *fermeture du Socket : les sockets sont des ressources limitées et doivent être fermées par les deux parties cliente et serveur*



java.net
Exemple de serveur

```
/** un serveur tres simple qui renvoie ce
* qu'il reçoit : serveur miroir
*/
import java.io.*;
import java.net.*;
public class ServerMiroir {
// Choisir un no de port non entre 1-1024
  static final int port = 8080;
  public static void main(String[] args ) {
  try {
    ServerSocket s = new ServerSocket(port);
    System.out.println("Serv Started: " + s);
    // Blocks until a connection occurs:
    Socket socket = s.accept();
    System.out.println(
      "Connect. accepted, socket: "+ socket);
    BufferedReader in =
      new BufferedReader(
        new InputStreamReader(
                socket.getInputStream()));
    PrintWriter out =
      new PrintWriter(
         new OutputStreamWriter(
            socket.getOutputStream()));
    while (true) {
      String str = in.readLine();
      if (str.equals("END")) break;
      System.out.println
         ("Echoing: " + str);
      out.println(str);
      out.flush(); // Ne pas oublier!
    }
    System.out.println("closing...");
    socket.close();
  } catch(Exception e) {
      e.printStackTrace();
  }
 }
}
```

java.net
Exemple de process serveur

```
/** un serveur tres simple qui renvoie ce
* qu'il reçoit : serveur miroir
*/
import java.io.*;
import java.net.*;
public class ServerMiroir {
// Choisir un no de port entre 1-10
  static final int port = 8080;
  public static void main(String[] a
    s = new ServerSocket(port);
    intln("Serv Started: " + s);
    il a connection occurs:
    Socket socket = s.accept();
    System.out.println(
      "socket: "+ socket);
    BufferedReader in =
      new BufferedReader(
        new InputStreamReader(
                socket.getInputStream()));
    PrintWriter out =
      new PrintWriter(
        new OutputStreamWriter(
          socke
    while (true)
      String str
      if (str.eq
      System.out
        ("Echoing
      out.println(str);
      out.flush(); // Ne pas oublier!
    }
    System.out.println("closing...");
    socket.close();
  } catch(Exception e) {
    e.printStackTrace();
  }
```

Pas d'@ IP : exécuté sur la machine courante

Représente la connection

sinon le buffer n'est pas écrit sur le réseau tant qu'il n'est pas plein

appel automatique à toString()
socket: [addr=207.0.0.1, port=1077, localport=8080]

java.net
Exemple de process Client

```
/** Un client tres simple qui envoie n
* lignes au serveur et lit n ligne
*/
import java.io.*;
import java.net.*;
public class ClientMiroir {
  static final int port = 8080;
  public static void main(String args[]) {
  try {
  // Passing null to getByName() produces the
  // special "Local Loopback" IP address, for
  // testing on one machine w/o a network:
  InetAddress addr =
    InetAddress.getByName(null);
  // On aurait pu utiliser l'adresse ou
  // le nom de façon équivalente
  // InetAddress.getByName("127.0.0.1");
  // InetAddress.getByName("localhost");
  System.out.println("addr = " + addr);
  Socket socket = new Socket(addr, port);
  System.out.println("socket = " + socket);
    DataInputStream in =
      new DataInputStream(
        new BufferedInputStream(
          socket.getInputStream()));
    PrintStream out =
      new PrintStream(
        new BufferedOutputStream(
          socket.getOutputStream()));
    for(int i = 0; i < 10; i ++) {
      out.println("howdy " + i);
      out.flush(); // Ne pas oublier
      String str = in.readLine();
      System.out.println(str);
    }
    out.println("END");
    out.flush();  // Ne pas oublier!
    socket.close();
  } catch(Exception e) {
    e.printStackTrace(); }
  }
}
```

java.net
Gestion de plusieurs clients pour le serveur

```
import java.io.*; import java.net.*;
class ServerMiroir extends Thread { private
  Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  ServerMiroir (Socket s) {
    socket = s;
    try { in = new BufferedReader( new ...);
          out = new PrintWriter( new...);
    } catch(IOException e) {...}
    start(); // Calls run()
  }
  public void run() {
    try {
      while (true) { //lire s = in.readLine}
      System.out.println("closing...");
      in.close();  out.close();
      socket.close();
    } catch (IOException e) { ... }
  }
}
```

```
public class MultiServerMiroir {
  static final int port = 8080;
  public static void main(String[] args )
  {
  try {
    ServerSocket s = new
        ServerSocket(port);
    System.out.println("Serv. Started");
      while(true) {
        // Bloque jusqu'a connexion
        Socket socket = s.accept();
        new ServerMiroir(socket);
      }
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Datagram- Buffered stream- et Data Stream- sockets

	Datagram	Buf.Stream	DataStream	CORBA
fiabilité de la communication	Non	Oui	Oui	Oui
Paramètres typés	Non	Non	Oui	Oui
performances	très bonne 1,8 ms	bonne 2 ms	mauvaise 301 ms	bonne 3,2 ms
Sécurité	Oui	Oui	Oui	Oui
Desciptions d'Interface	Non	Non	Non	Oui

Les sockets fournissent un moyen de bas niveau pour la programmation réseau

La programmation d'une appli. Client/Serveur doit y plutôt s'appuyer sur CORBA IIOP et Java RMI