Architecture des réseaux et des systèmes informatiques répartis

Théo Pierron

Polytech 4A Apprentissage

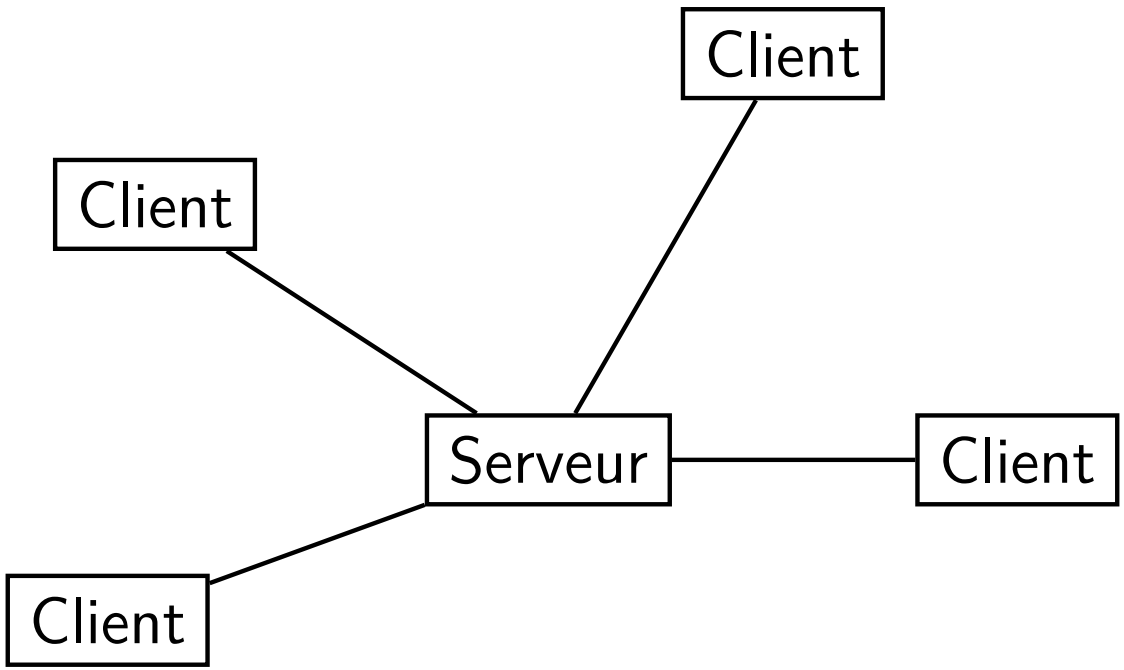
Modalités pratiques

- `theo.pierron@univ-lyon1.fr`
- `https://perso.liris.cnrs.fr/tpierron/ARSIR2021`
- Discord
- Correction des TPs : faites des groupes !
- Examen (50%) + TP (50%)

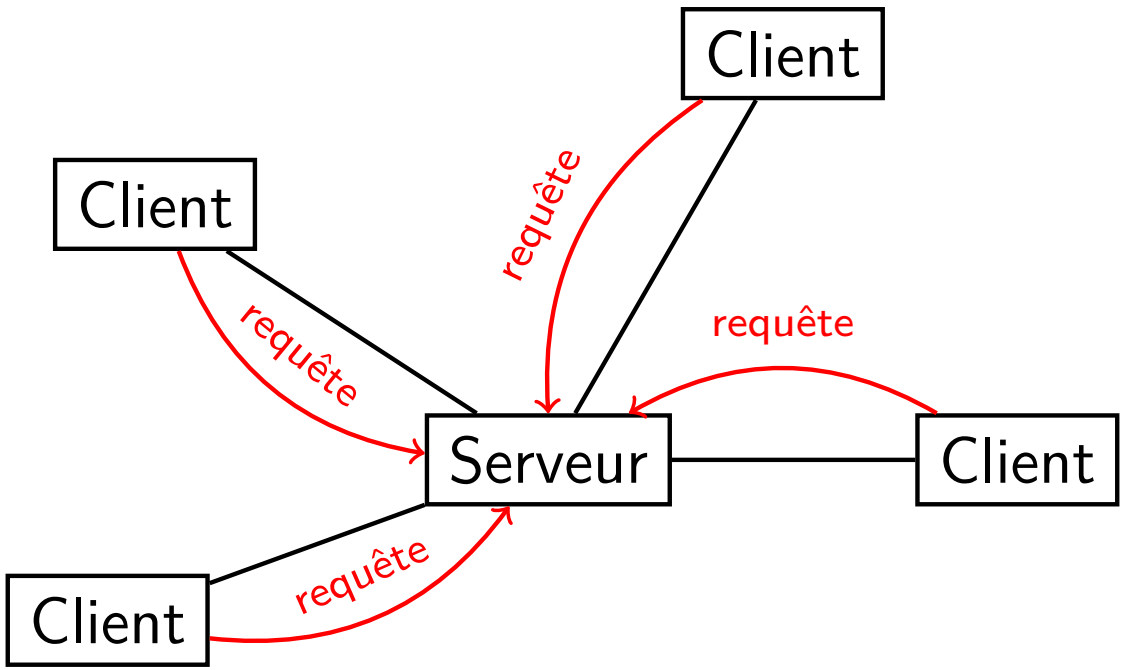
Architecture client/serveur

Principe général

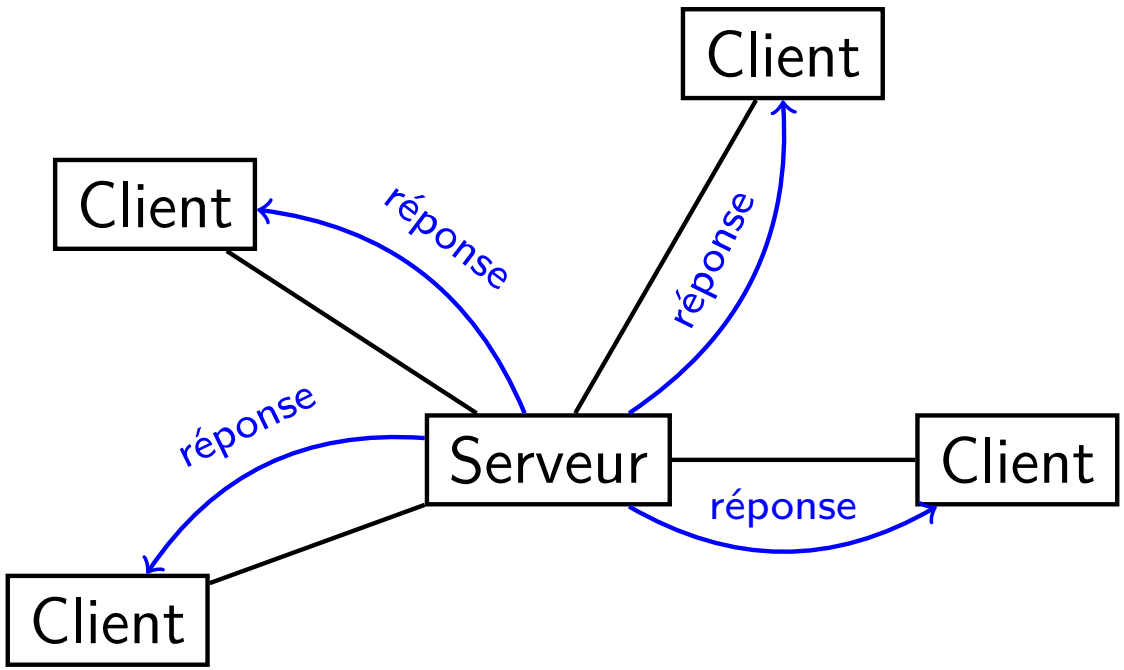
Principe général



Principe général



Principe général



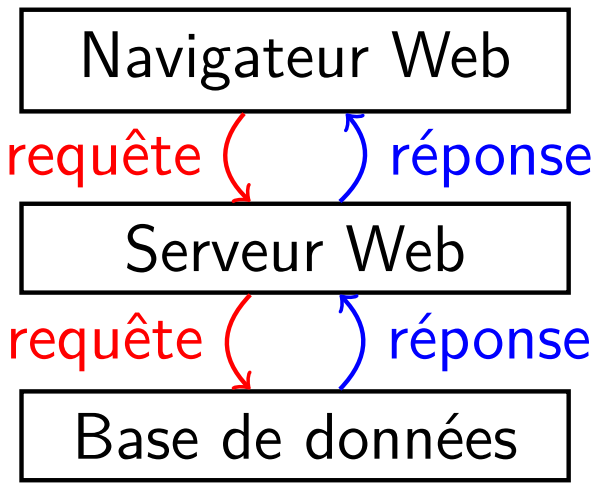
Exemples

Exemples

- Terminal à distance (telnet, ssh,...)
- Navigation Web (protocole HTTP)
- Transfert de fichiers (protocole FTP)
- Envoi/réception de mails (SMTP/POP/IMAP)
- Annuaires (DNS)
- Temps (NTP)
- Impression (CUPS)
- Bases de données
- IRC, Discord, Twitch ...

Architecture 3 niveaux

Le serveur peut être un client pour un autre serveur !



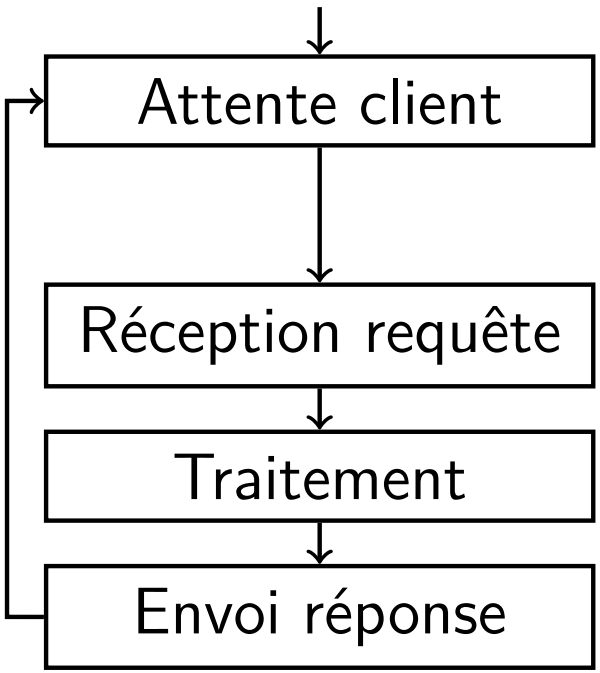
Deux problèmes

- Un serveur doit pouvoir gérer plusieurs clients.
  → Organisation logicielle ?
- Client et serveur peuvent être sur des machines différentes.
  → Communication ?

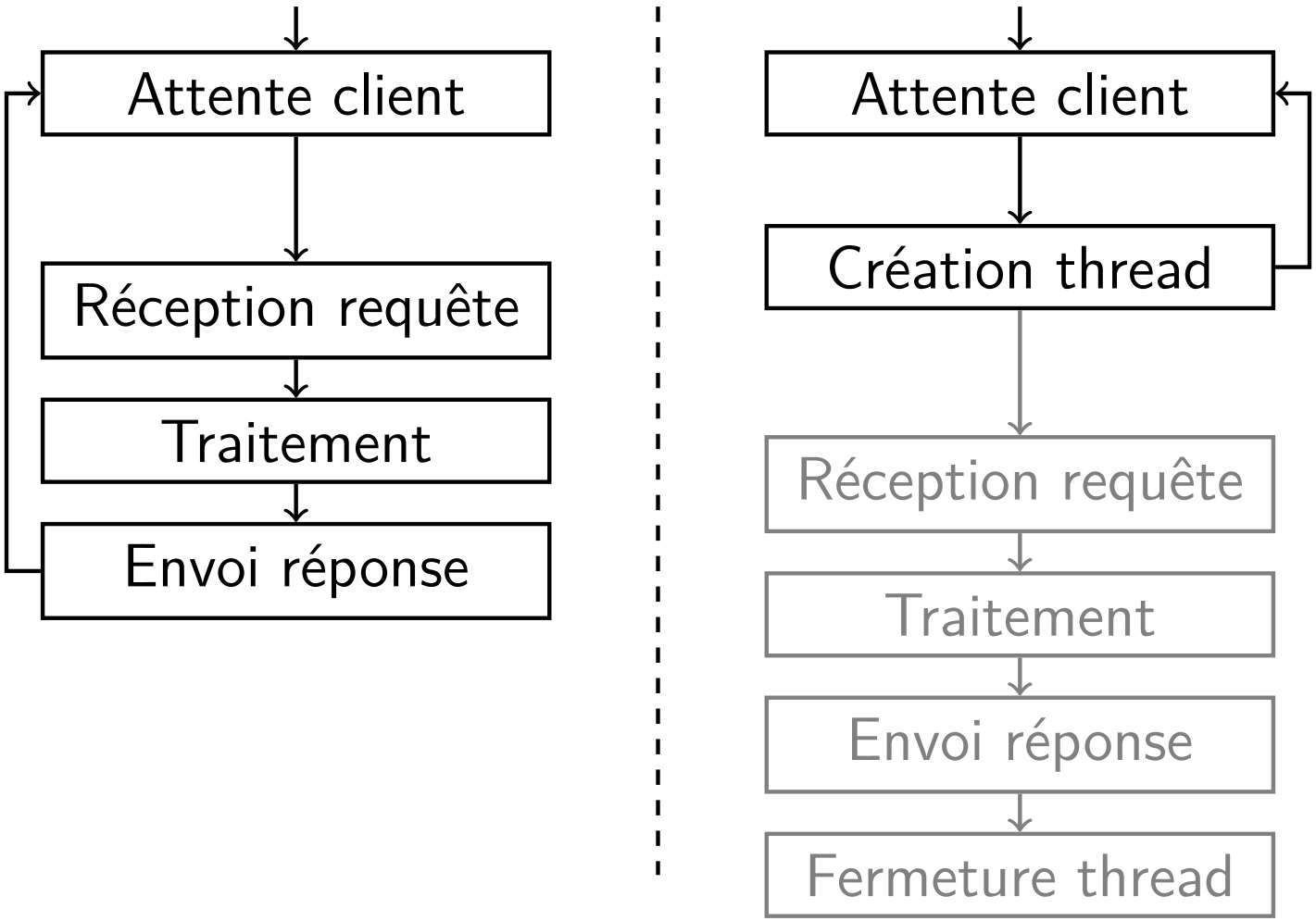
Organisation logicielle

- Client : envoie ses requêtes, reçoit les réponses.
- Serveur : doit gérer plusieurs clients.

- Client : envoie ses requêtes, reçoit les réponses.
- Serveur : doit gérer plusieurs clients.

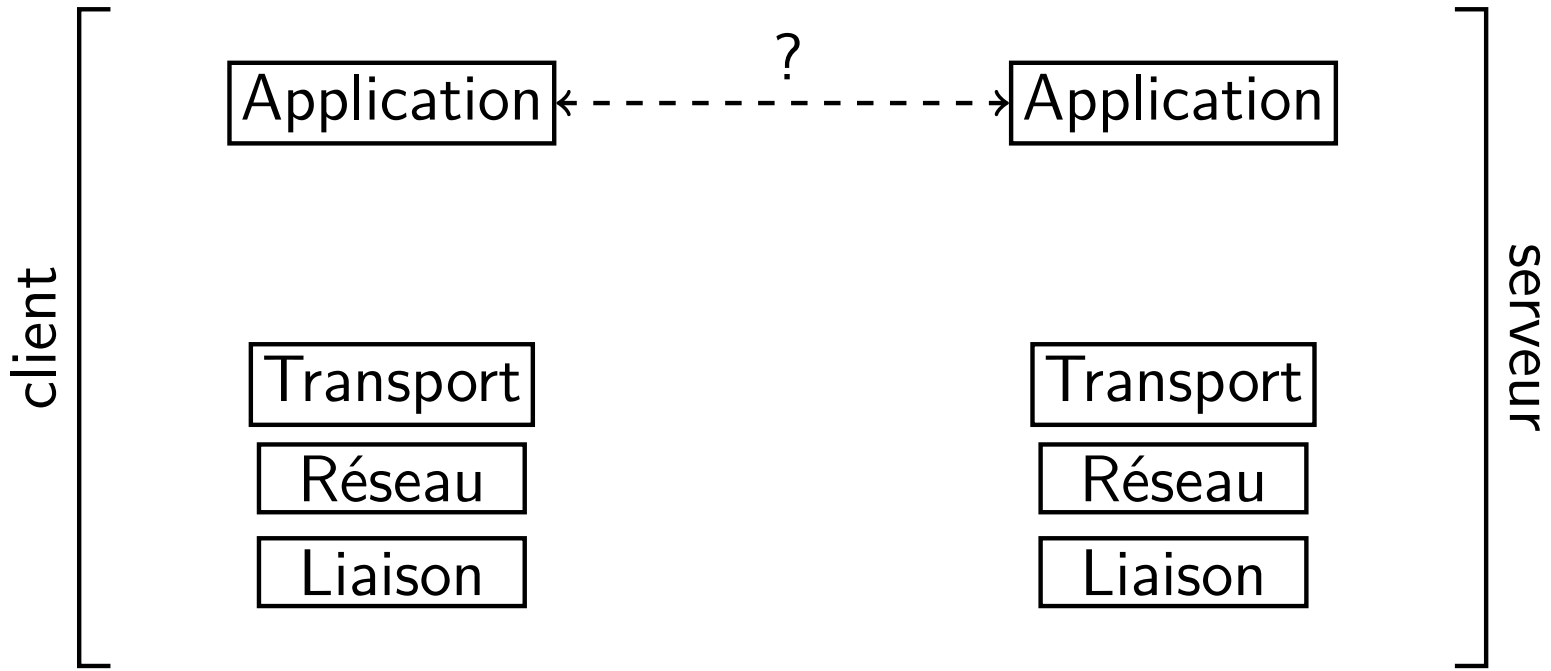


- Client : envoie ses requêtes, reçoit les réponses.
- Serveur : doit gérer plusieurs clients.

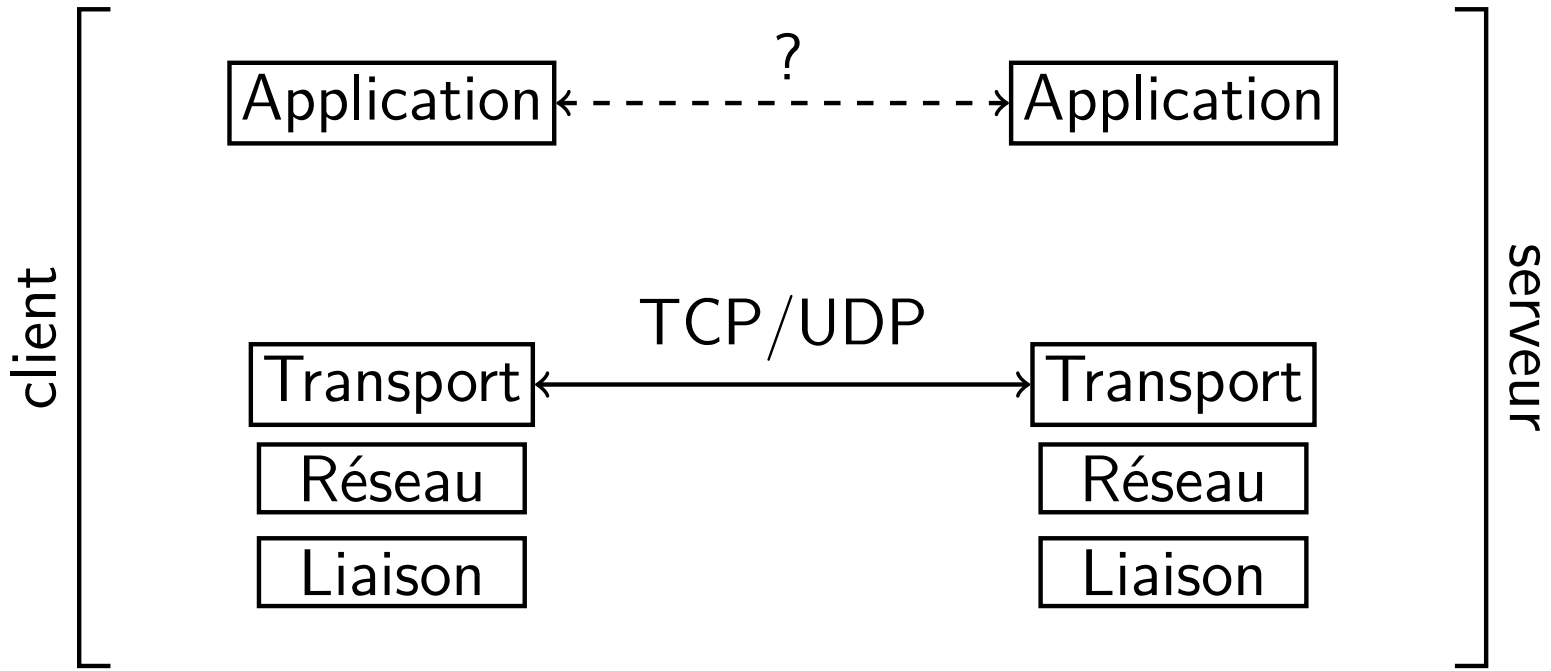


Communication ?

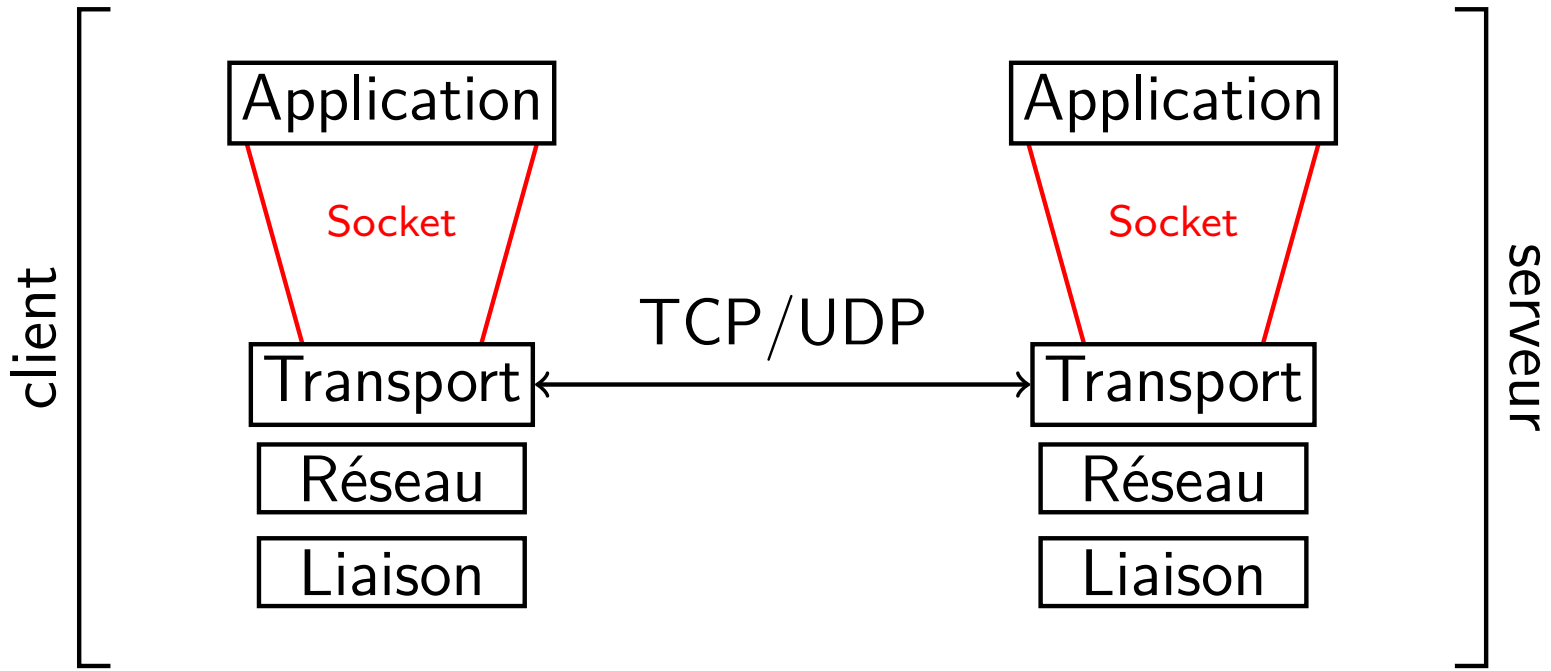
Sockets


client
Application
?
Application
Transport
Réseau
Liaison
Transport
Réseau
Liaison
serveur

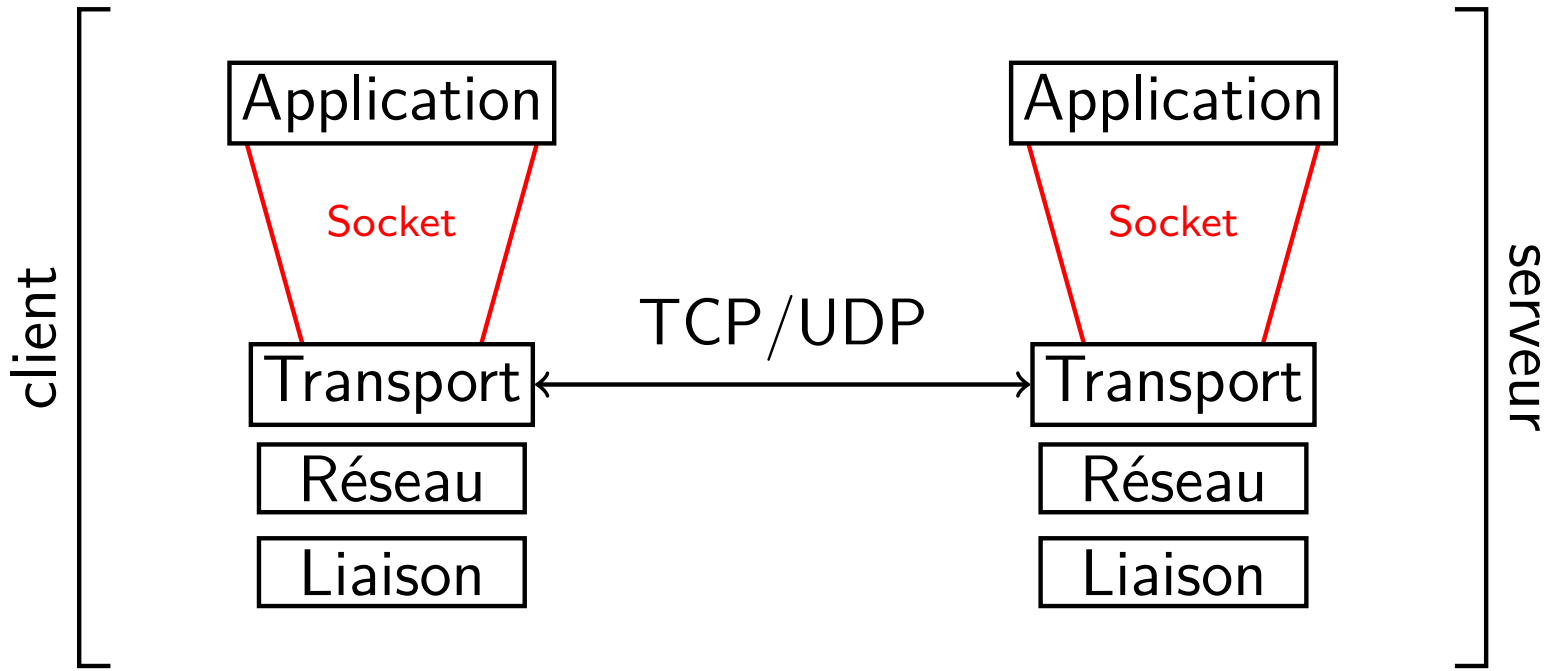
Sockets


client
Application
?
Application
TCP/UDP
Transport
Transport
Réseau
Réseau
Liaison
Liaison
serveur

Sockets


client
Application
Socket
Transport
Réseau
Liaison
TCP/UDP
Application
Socket
Transport
Réseau
Liaison
serveur

Sockets



- Socket = interface de communication entre deux couples (adresse IP, port), basée sur TCP ou UDP
- (Presque) pas besoin de gérer les paquets à la main
- Communication $\approx$ lecture/écriture dans un fichier

Protocoles de communication

UDP	TCP
(User Datagram Protocol)	(Transmission Control Protocol)

Protocoles de communication

UDP (User Datagram Protocol)	TCP (Transmission Control Protocol)
• rapide • pas de garantie • adapté aux échanges brefs ou à la diffusion	• plus lent • garanties • adapté aux échanges de plusieurs messages

Protocoles de communication

UDP (User Datagram Protocol)	TCP (Transmission Control Protocol)
• rapide • pas de garantie • adapté aux échanges brefs ou à la diffusion	• plus lent • garanties • adapté aux échanges de plusieurs messages

	séquentiel	parallèle
UDP	communication et exécution rapides	Peu utilisé
TCP	Peu utilisé	Transport fiable, adapté aux échanges plus longs

Architecture séquentielle UDP

Deux objets :

- `DatagramSocket` : le canal de communication
- `DatagramPacket` : le paquet à envoyer/recevoir

DatagramSocket

Méthodes :

- `send(DatagramPacket packet)`
- `receive(DatagramPacket packet)`

DatagramSocket

Méthodes :

- `send(DatagramPacket packet)`
- `receive(DatagramPacket packet)`

Constructeurs :

- `DatagramSocket(int port, InetAddress adresse)`
- `DatagramSocket(int port)`
- `DatagramSocket()`

DatagramPacket

Attributs :

- `InetAddress Address` : adresse de l'émetteur/destinataire
- `int Port` : port de l'émetteur/destinataire
- `byte[] Data` : contenu

DatagramPacket

Attributs :

- `InetAddress Address` : adresse de l'émetteur/destinataire
- `int Port` : port de l'émetteur/destinataire
- `byte[] Data` : contenu

Constructeurs :

- Réception : `DatagramPacket(byte[] data, int length)`
- Envoi : `DatagramPacket(byte[] data, int length, InetAddress destination, int port)`

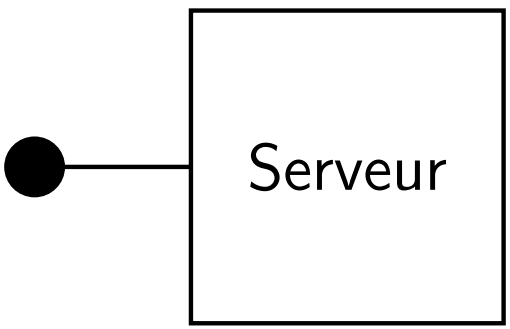
Exemple

Remarques

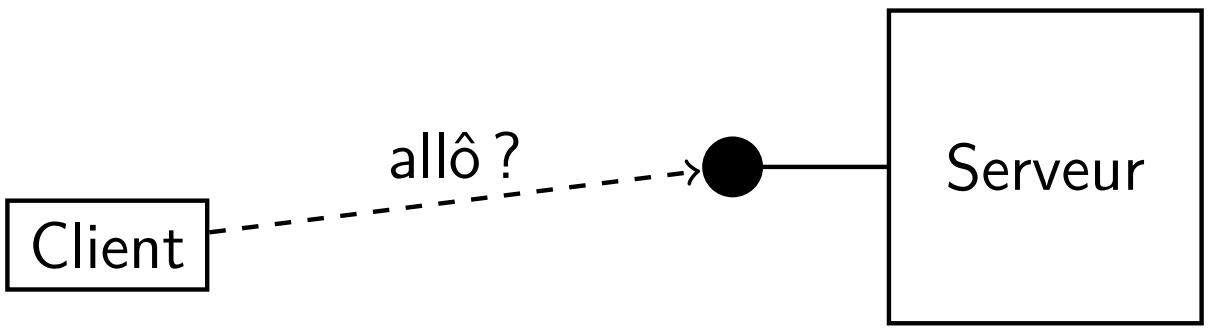
- Requêtes indépendantes (pas de préservation de l'état entre deux requêtes).
- Gestion des paquets à la main (y compris les erreurs).
- Choisir la taille des datagrammes en fonction de la capacité du réseau (max 8Ko).

Architecture parallèle TCP

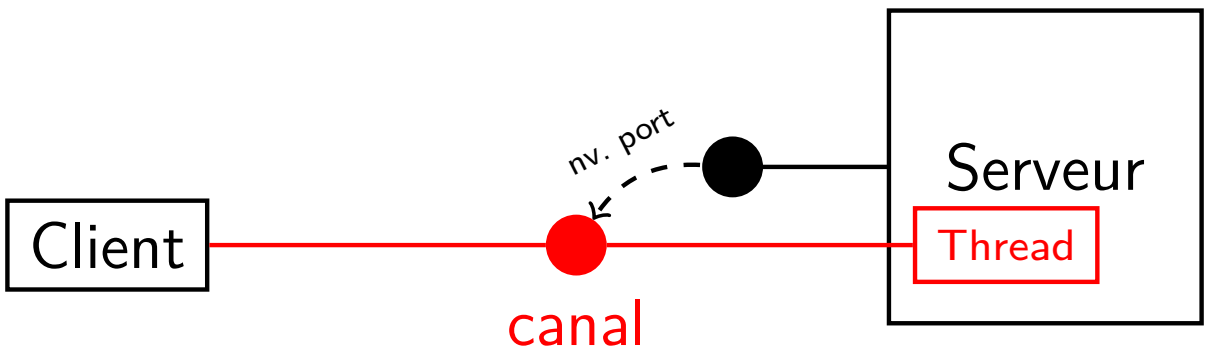
Fonctionnement générique



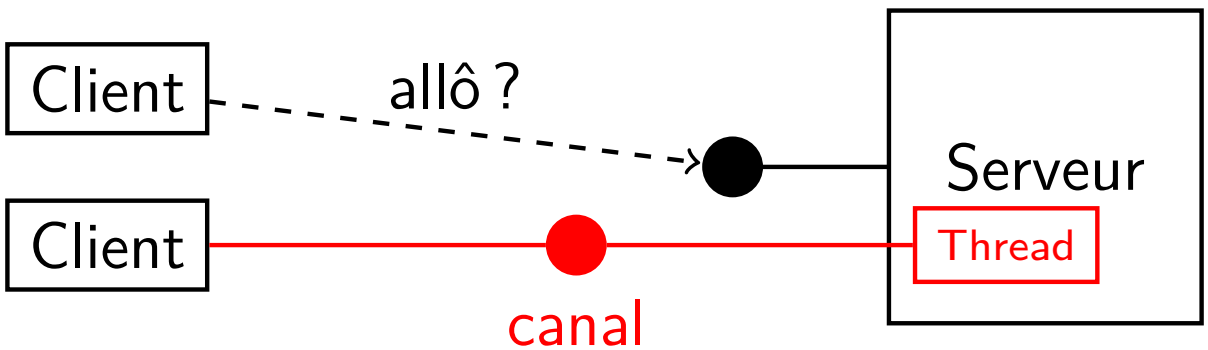
Fonctionnement générique



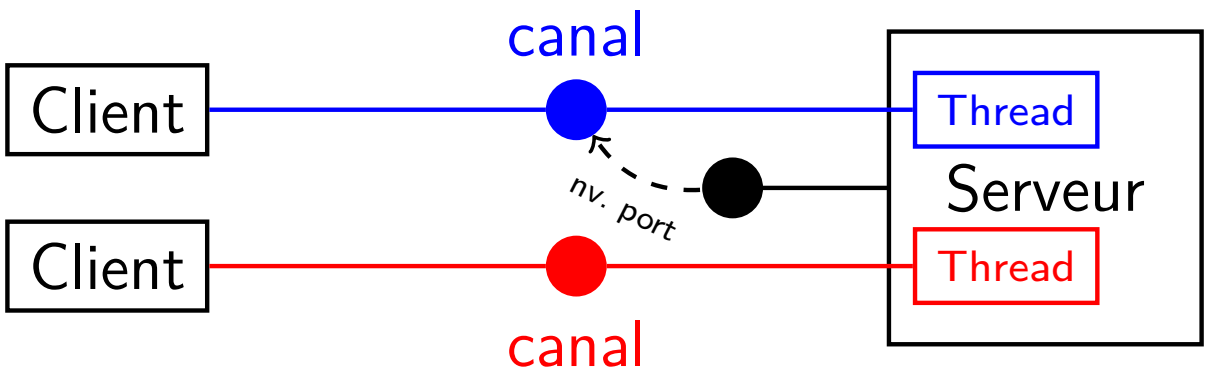
Fonctionnement générique



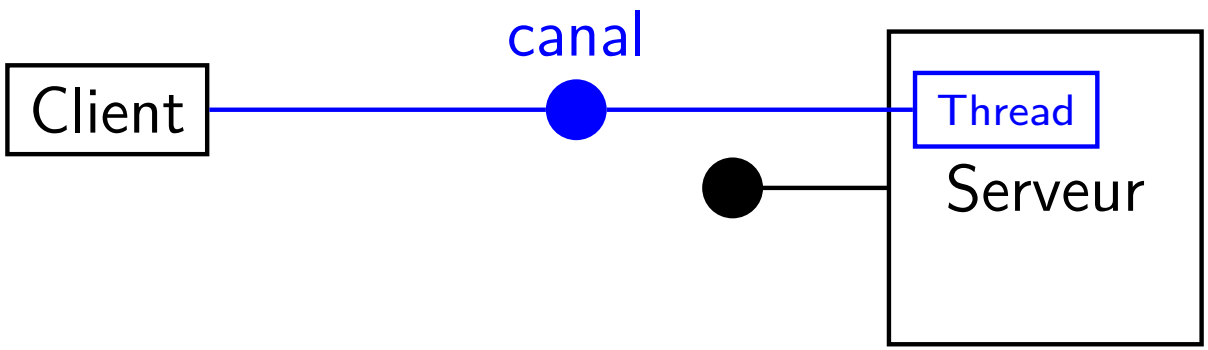
Fonctionnement générique



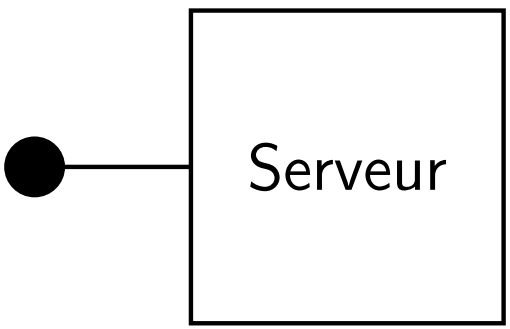
Fonctionnement générique



Fonctionnement générique



Fonctionnement générique



Exemple générique

Bibliothèques :

- Sockets : java.net.ServerSocket, java.net.Socket
- Lecture/Écriture : java.io.PrintWriter, java.util.Scanner
- Threading : java.util.concurrent.Executors

Code côté serveur (1)

```
1  public static void main() {
2
3      var listener = new ServerSocket(12345);
4      var pool = Executors.newFixedThreadPool(20);
5
6
7
8
9
10 }
```

Code côté serveur (1)

```
1  public static void main() {
2
3      var listener = new ServerSocket(12345);
4      var pool = Executors.newFixedThreadPool(20);
5
6      while (true) {
7          Socket new_task = listener.accept();
8          pool.execute(new Process(new_task));
9      }
10 }
```

Code côté serveur (2)

```
1  public static class Process implements Runnable {
2      private Socket socket;
3      Process(Socket socket) {
4          this.socket = socket;
5      }
6
7
8
9
10
11
12
13
14
15
16 }
```

Code côté serveur (2)

```
1  public static class Process implements Runnable {
2      private Socket socket;
3      Process(Socket socket) {
4          this.socket = socket;
5      }
6
7      public void run() {
8          var in = new Scanner(socket.getInputStream());
9          var out = new PrintWriter(socket.getOutputStream());
10
11
12
13
14
15     }
16 }
```

Code côté serveur (2)

```
1  public static class Process implements Runnable {
2    private Socket socket;
3    Process(Socket socket) {
4      this.socket = socket;
5    }
6
7    public void run() {
8      var in = new Scanner(socket.getInputStream());
9      var out = new PrintWriter(socket.getOutputStream());
10
11     Lecture dans in
12     Traitement
13     Ecriture dans out
14     socket.close();
15   }
16 }
```

Code côté client

```
1  public static void main() {
2      var socket = new Socket("serveur.fr", 12345);
3      var in = new Scanner(socket.getInputStream());
4      var out = new PrintWriter(socket.getOutputStream());
5
6
7
8  }
```

Code côté client

```
1  public static void main() {
2      var socket = new Socket("serveur.fr", 12345);
3      var in = new Scanner(socket.getInputStream());
4      var out = new PrintWriter(socket.getOutputStream());
5
6      Ecriture dans out
7      Lecture dans in
8  }
```

Remarques

- Protocole fiable.
- Attention à flush les buffers (println).
- Découpage des données de l'émetteur inconnu.
- Gestion des problèmes réseau avec des exceptions.