SRCS :
La programmation réseau en Java.
Version 15.01

Julien Sopena[1]

[1]julien.sopena@lip6.fr
Équipe REGAL - INRIA Rocquencourt
LIP6 - Université Pierre et Marie Curie

Master SAR 1ère année - SRCS - 2016/2017

Grandes lignes du cours

Une abstraction des protocoles réseau

Gestion des addresses réseau en Java

Le sockets TCP

Le sockets UDP

Outline

Quel niveau d'abstraction pour le réseau ?

Les 7 couches du modèle *OSI*

Couche 7 - **Applicative** : Les logiciels *(ex : NFS)*

Couche 6 - **Présentation** : Représentation des données *(ex : XDR)*

Couche 5 - **Session** : Établissement et maintient des sessions *(ex : RPC)*

Couche 4 - **Transport** : Liaison de bout en bout *(ex : UDP, TCP, ...)*

Couche 3 - **Réseau** : Adressage et routage entre machines *(ex : IP)*

Couche 2 - **Liaison** : Gestion des envois point à point *(ex : Ethernet)*

Couche 1 - **Physique** : Support de transmission

Quel niveau d'abstraction pour le réseau ?

Les 7 couches du modèle *OSI*

Couche 7 - **Applicative** : Les logiciels *(ex : NFS)*

Couche 6 - **Présentation** : Représentation des données *(ex : XDR)*

Couche 5 - **Session** : Établissement et maintient des sessions *(ex : RPC)*

Couche 4 - **Transport** : Liaison de bout en bout *(ex : UDP, TCP, ...)*

Couche 3 - **Réseau** : Adressage et routage entre machines *(ex : IP)*

Couche 2 - **Liaison** : Gestion des envois point à point *(ex : Ethernet)*

Couche 1 - **Physique** : Support de transmission

Les **Sockets** Java offrent une abstraction du réseau au niveau des couches Transport (4) et Réseau (3). Java introduit donc différentes classes correspondant aux protocoles abstraits : TCP ou UDP.

La notion de mode de connecté

Définition

*En **mode connecté**, la communication entre un client et un serveur est précédée d'une connexion et suivie d'une fermeture :*

- *Facilite la gestion d'état*
- *Meilleurs contrôle des arrivées/départs de clients*
- *Uniquement communication unicast*
- *Plus lent au démarrage*

Définition

*En **mode non connecté**, les messages sont envoyés librement :*

- *Plus facile à mettre en œuvre*
- *Plus rapide au démarrage*

Les protocoles TCP et UDP

TCP : une communication de type téléphone

- **Mode connecté :** protocole de prise de connexion (lent)
- **Sans perte :** un message arrive au moins un fois
- **Sans duplication :** un message arrive au plus une fois
- **Avec fragmentation :** les messages peuvent être coupés
- **Ordre respecté :** messages délivrés dans l'ordre d'émission

UDP : une communication de type courrier

- **Mode non connecté :** pas de protocole de connexion (plus rapide)
- **Avec perte :** l'émetteur n'est pas assuré de la délivrance
- **Avec duplication :** un message peut arriver plus d'une fois
- **Sans fragmentation :** les messages envoyés ne sont jamais coupés
  $\Longrightarrow$ soit un message arrive entièrement, soit il n'arrive pas
- **Ordre non respecté :** messages délivrés dans l'ordre d'arrivé

La notion de mode de connecté

Attention

Il ne faut pas confondre connexion au niveau transport et connexion au niveau applicatif :

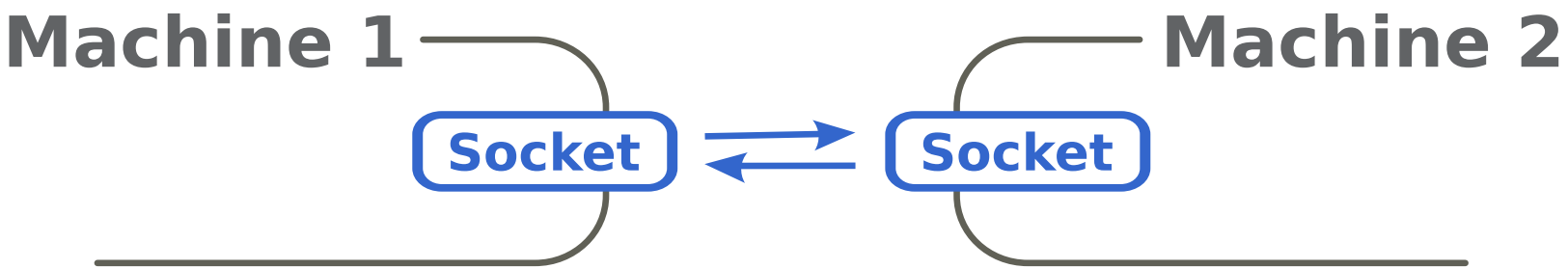
- **FTP** fonctionne en connecté en utilisant **TCP**
- **DNS** fonctionne en non-connecté en utilisant **UDP**
- **HTTP** fonctionne en non-connecté en utilisant **TCP**

Les Sockets

Définition

*Une **Socket** est une API (interface logicielle) avec les services du système d'exploitation, qui permet d'exploiter facilement et de manière uniforme les services d'un protocole réseau.*

Une socket est un demi-point de connexion d'une application.



Java défini plusieurs classes de sockets suivant le protocole :

- TCP : **ServerSocket** et **Socket**
- UDP : **DatagramSocket**

Outline

Identification des sockets

Une socket est identifiée par :

- en absence de connexion : $\langle$@IP,port$\rangle$
- en cas de connexion : ($\langle$@IP$_{\text{src}}$,port$_{\text{src}}\rangle$,$\langle$@IP$_{\text{dest}}$,port$_{\text{dest}}\rangle$)

Attention

Deux sockets peuvent occuper le même port local si elles sont connectées sur des IPs et/ou ports différents.

Il existe deux classes pour gérer les addresses :

InetAddress : représente une adresse internet (ip/fqdn)

SocketAddress : représente l'adresse d'une socket

InetAddress : les adresses internet

La classe **InetAddress** sert à représenter les adresses internet

- pas de constructeur mais des méthodes statiques

```
InetAddress me = InetAddress.getByName("sopena.fr");
System.out.println(me.getHostAddress()+" = "+me.getHostName());
```

```
90.46.19.126 = sopena.fr
```

```
InetAddress g = InetAddress.getByAddress(new byte[] {8,8,8,8});
System.out.println(g.getHostAddress()+" = "+g.getCanonicalHostName());
```

```
8.8.8.8 = google-public-dns-a.google.com
```

```
InetAddress ici = InetAddress.getLocalHost();
```

SocketAddress : les adresses internet

La classe **InetSocketAddress** sert à représenter ⟨@IP,port⟩ :

- elle hérite de la classe abstraite **SocketAddress**
- ces instances sont des objets immuables
- elle n'associe pas le protocole utilisé

On peut les construire ou les récupèrer de :

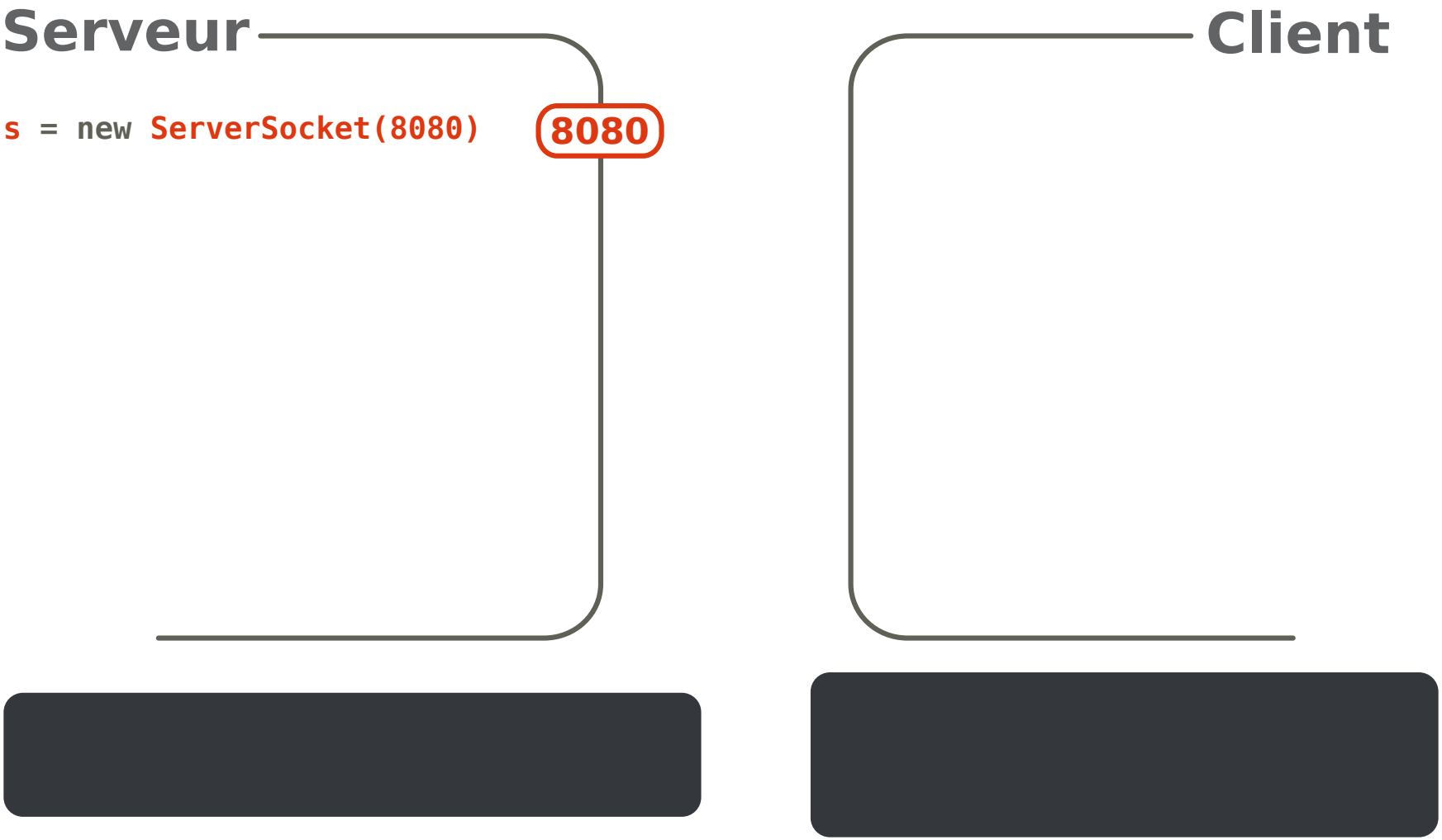
- sockets existantes
- de packets reçu ou émis

```
InetSocketAddress localhost =
                  new InetSocketAddress("localhost", 8080);
socket.connect(localhost);

SocketAddress ici = socket.getLocalSocketAddress();
SocketAddress labas = socket.getRemoteSocketAddress();
```

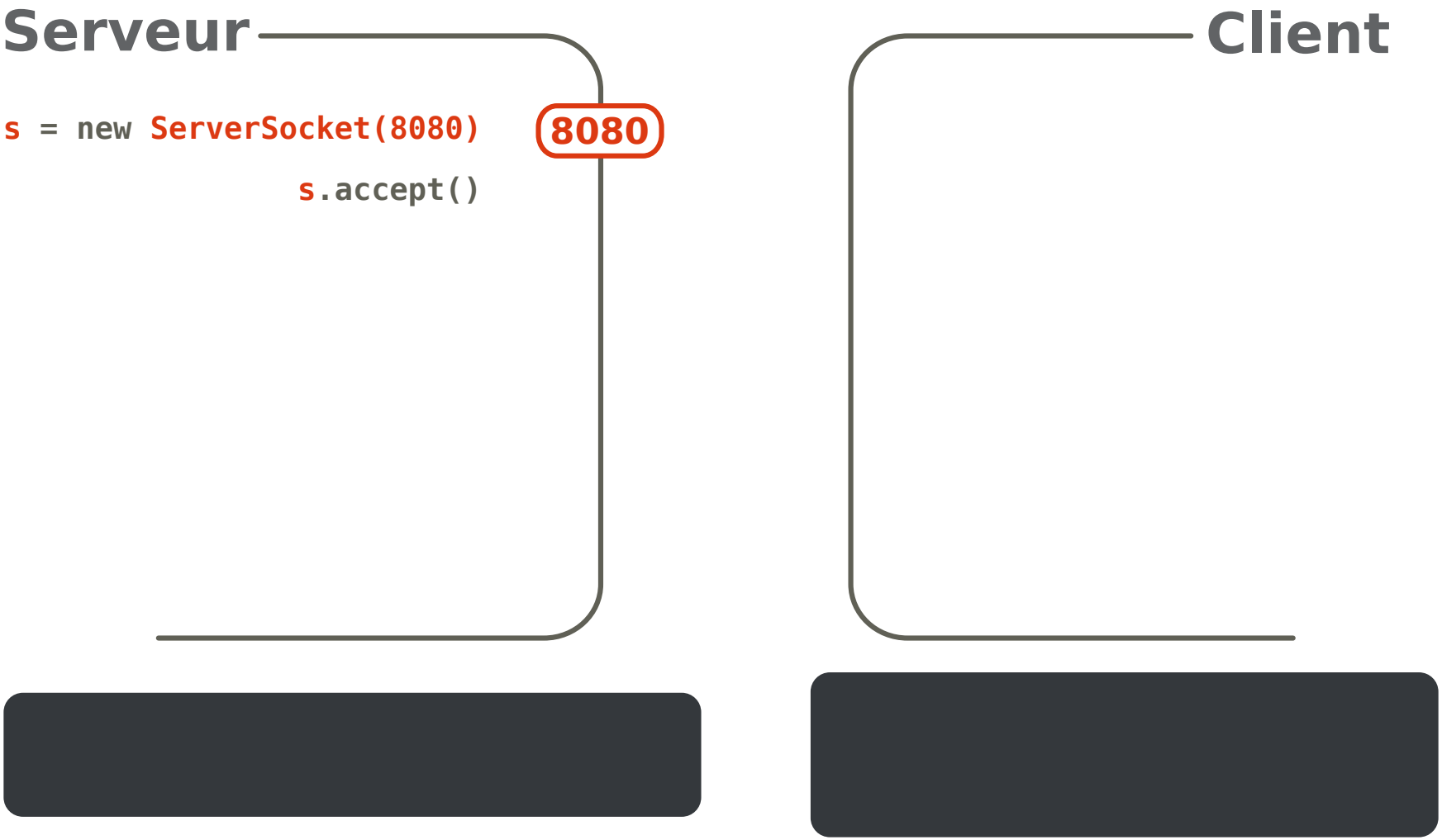
Outline

Utilisation d'une socket TCP

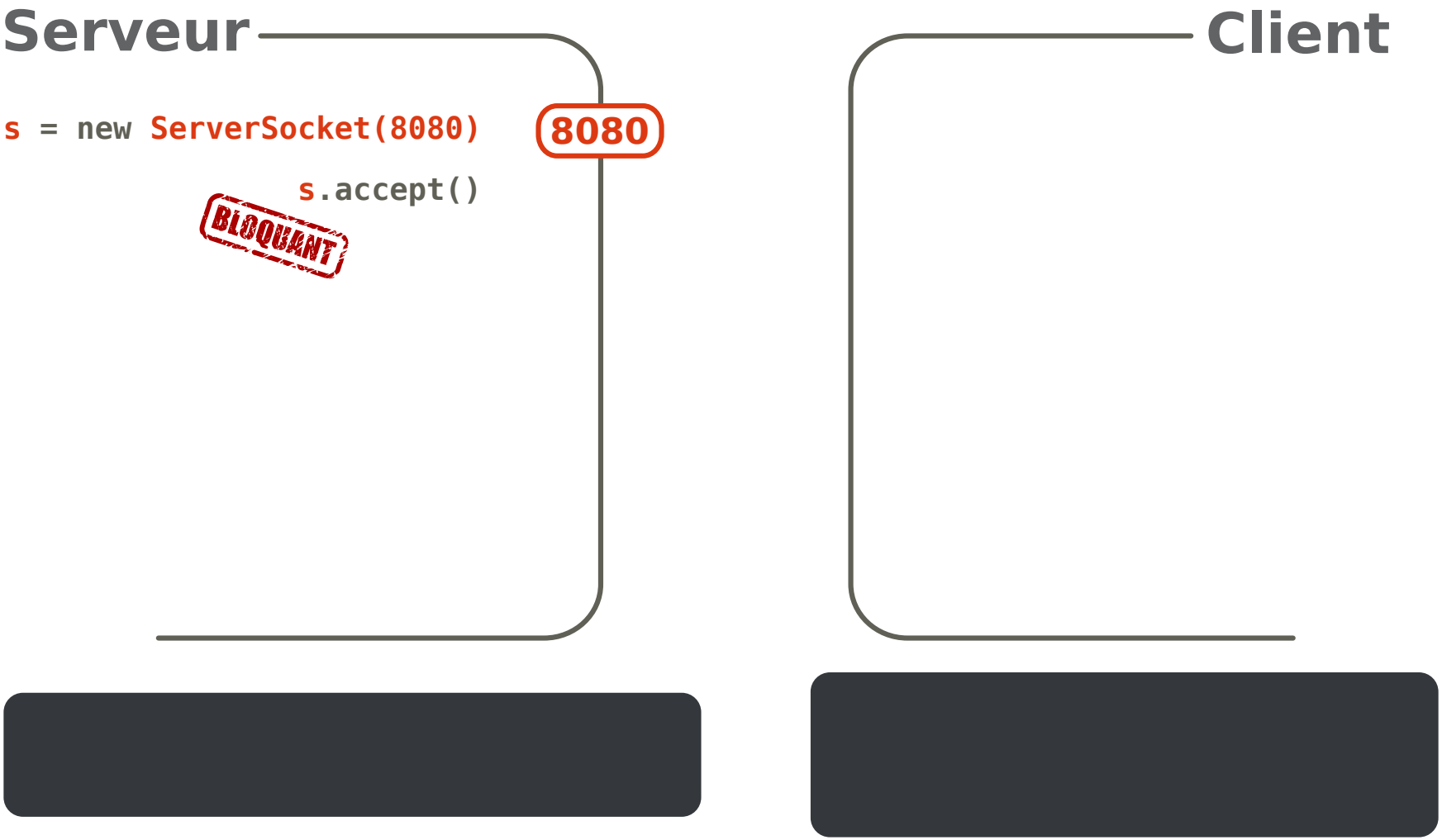
Utilisation d'une socket TCP



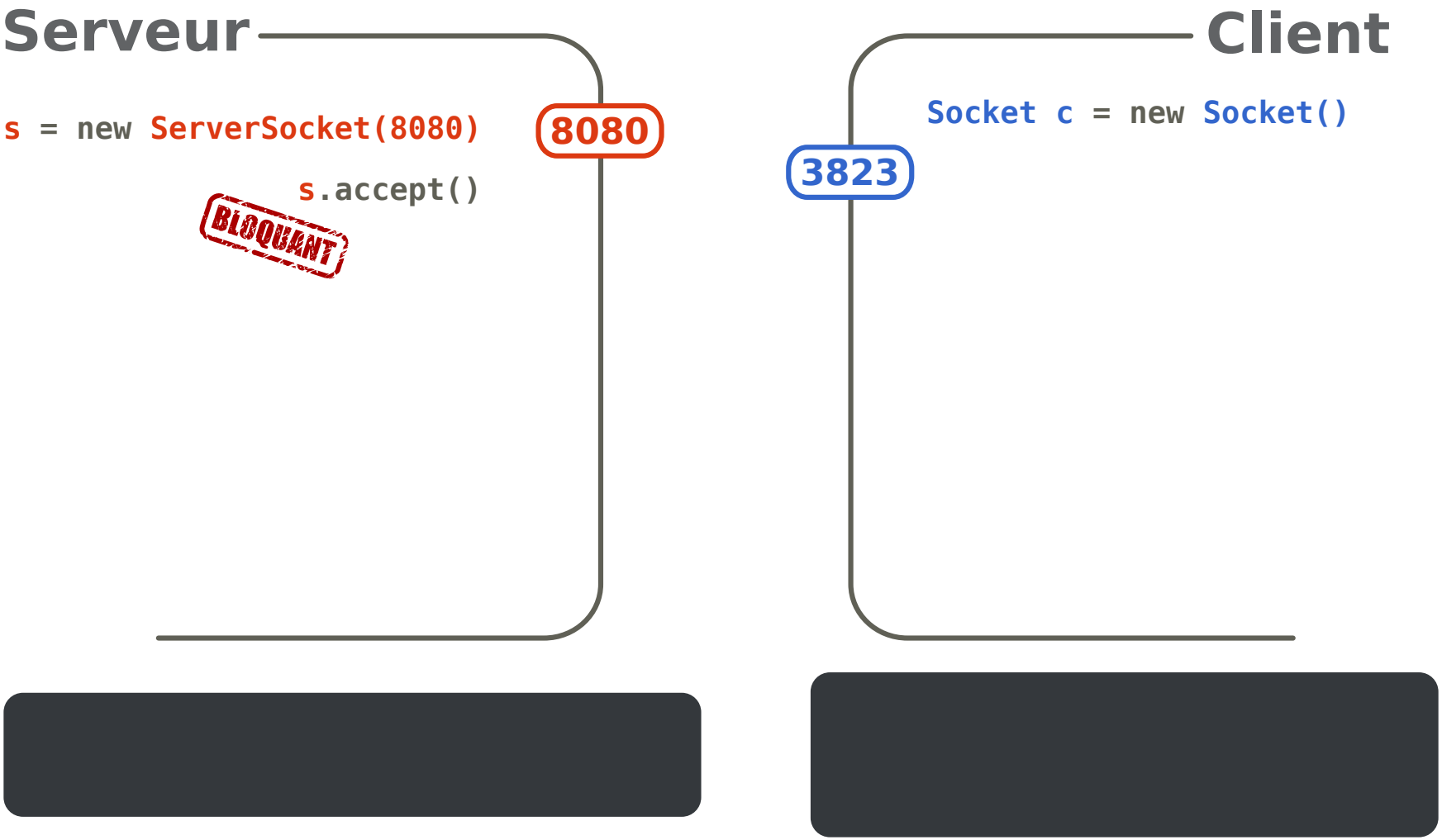
Utilisation d'une socket TCP



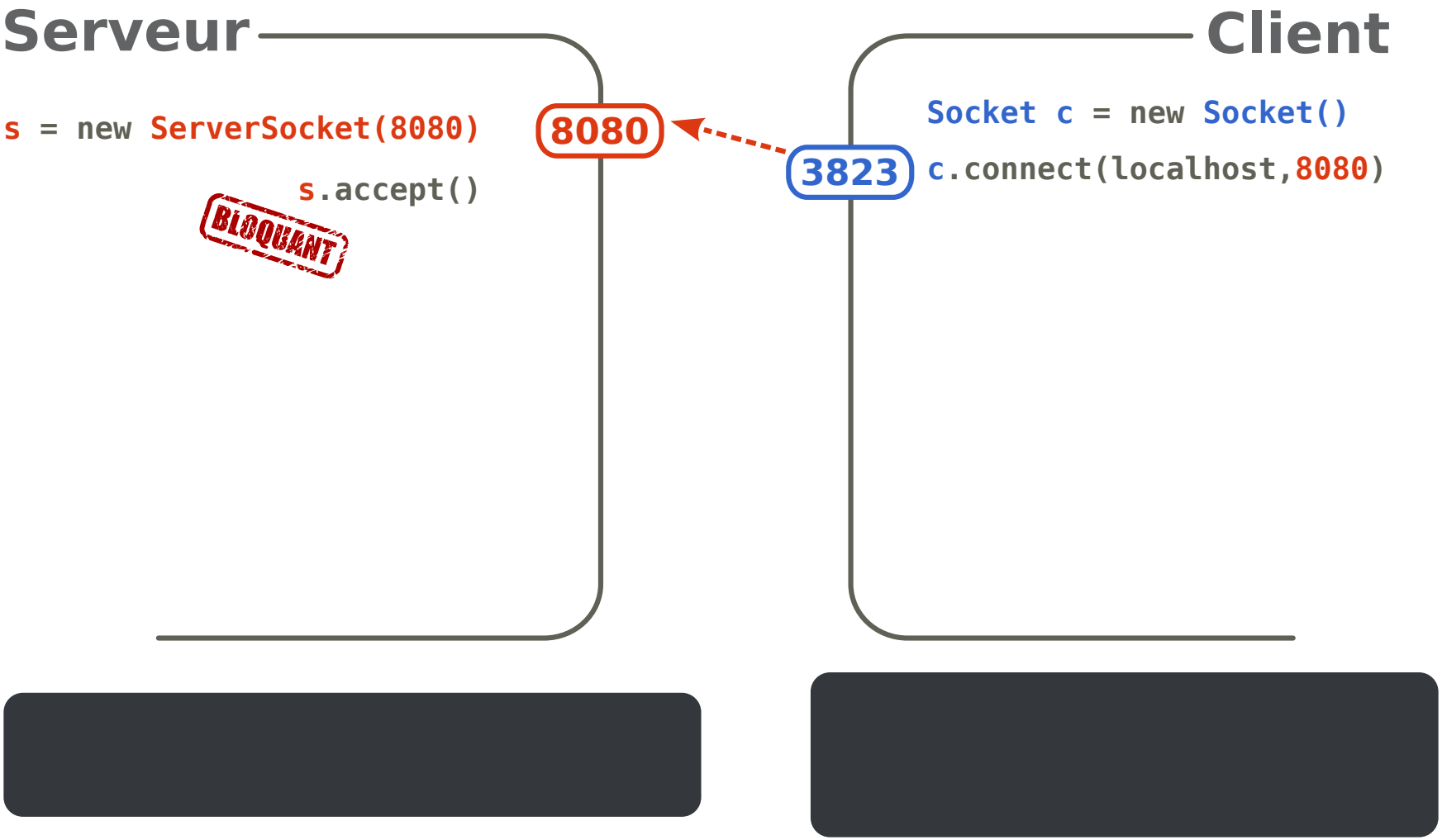
Utilisation d'une socket TCP



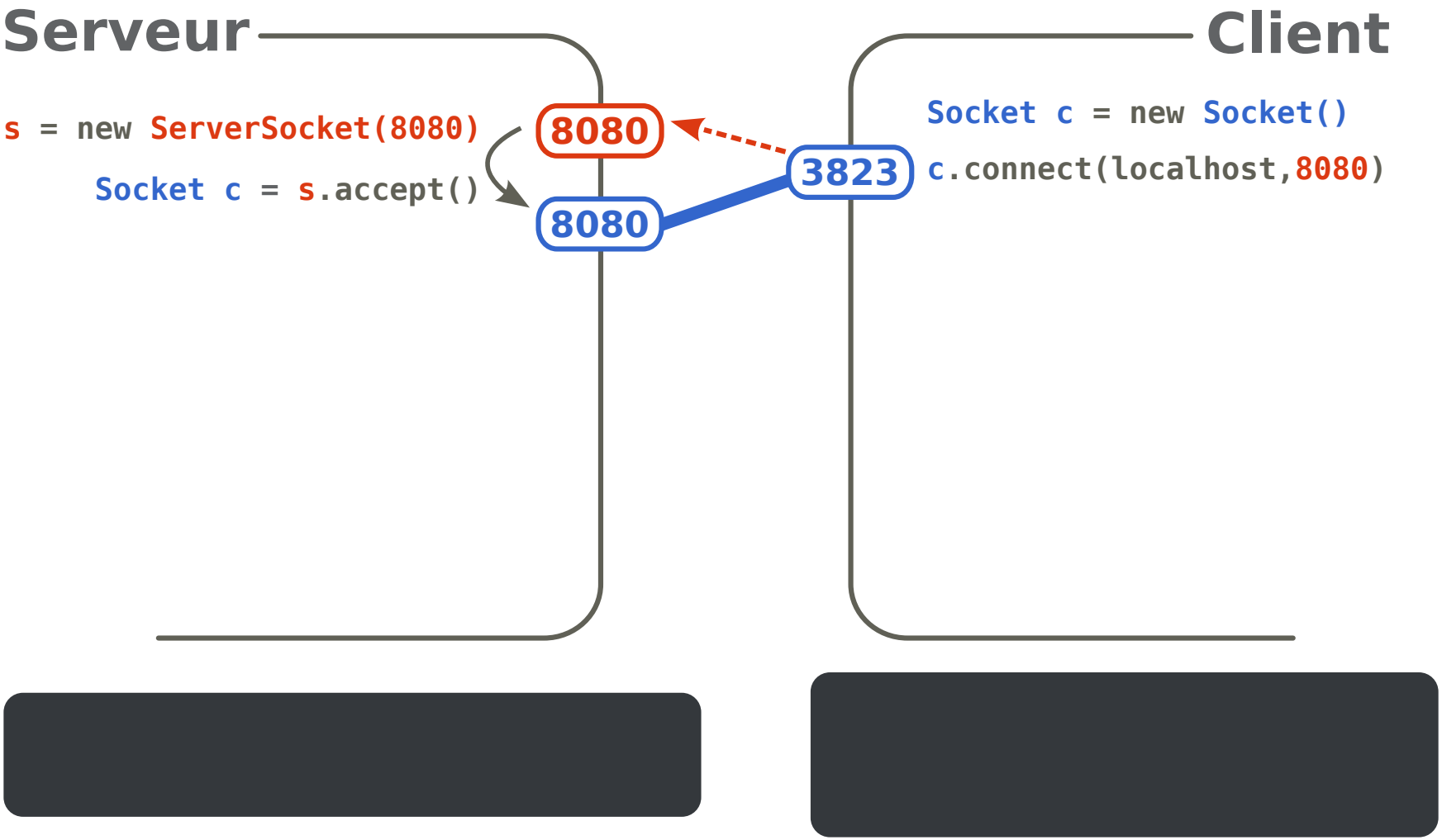
Utilisation d'une socket TCP



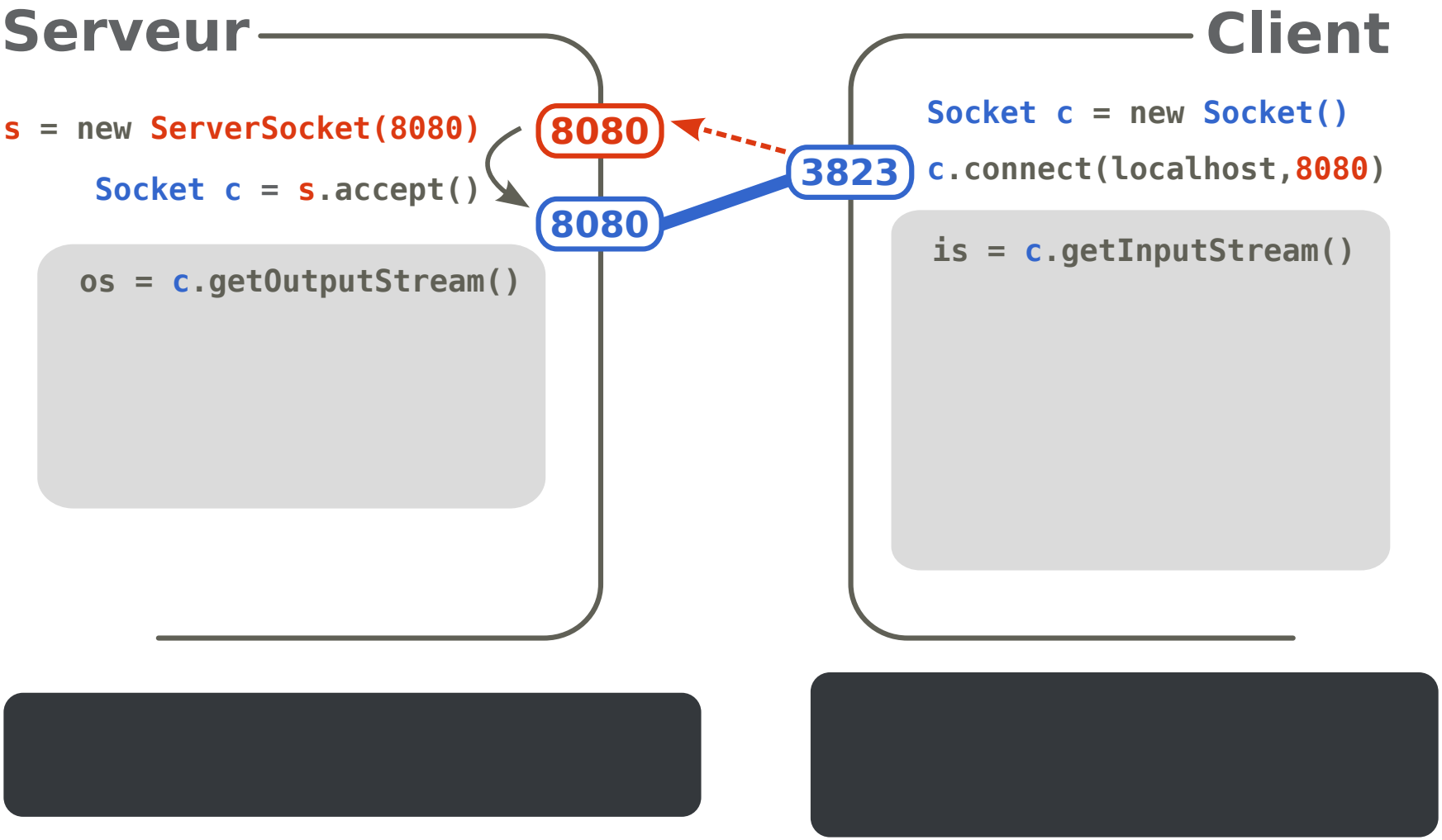
Utilisation d'une socket TCP



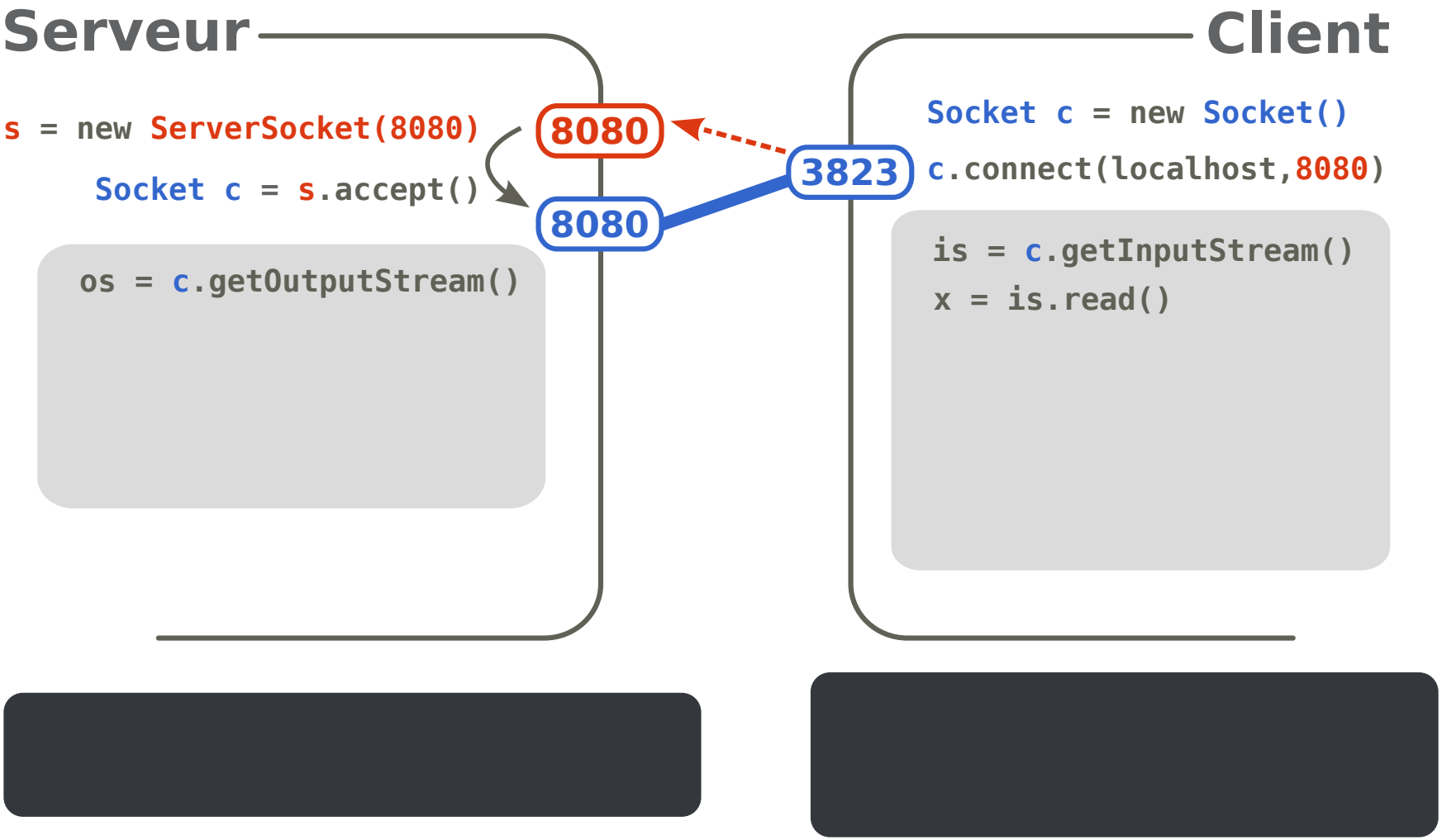
Utilisation d'une socket TCP



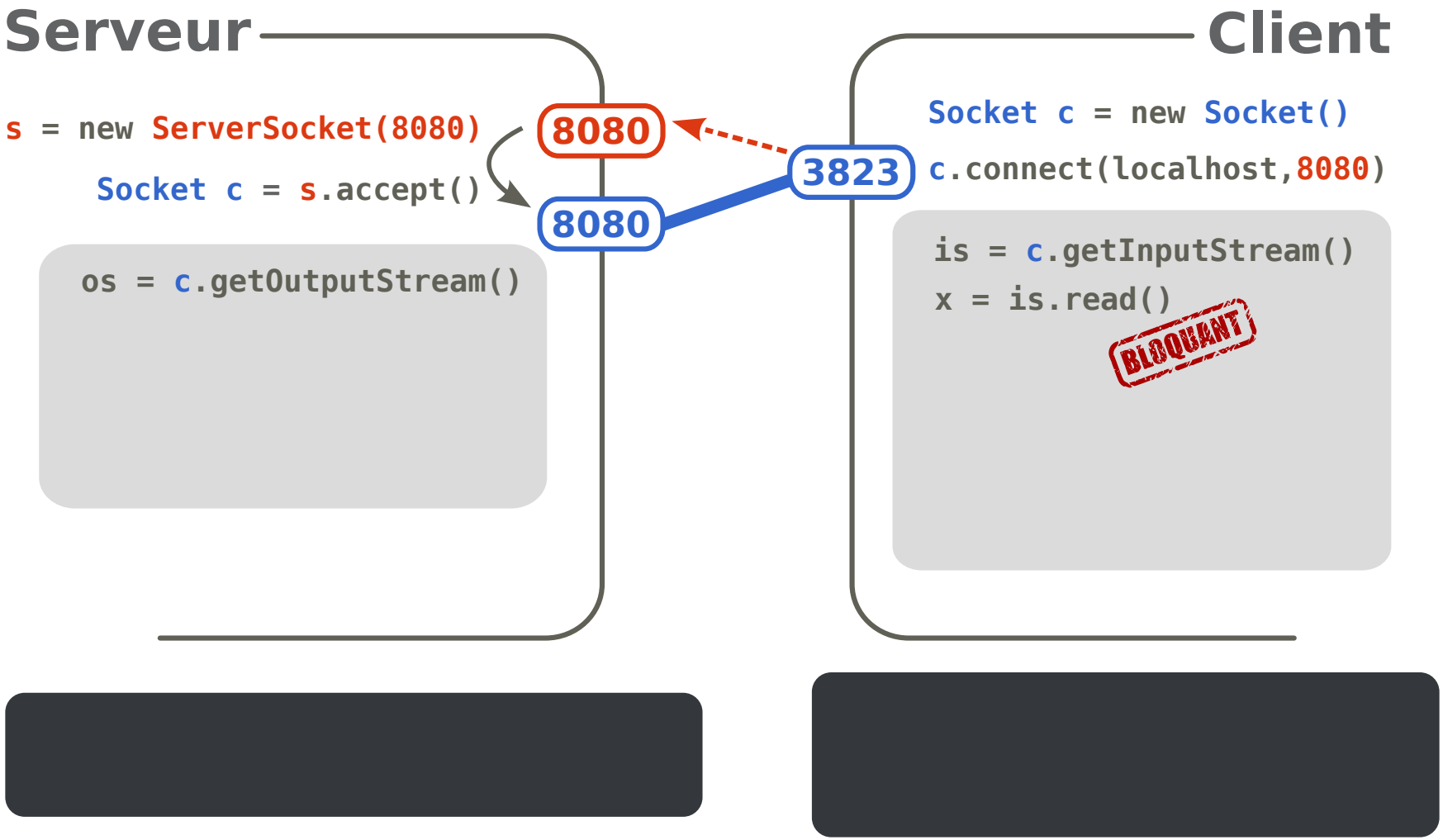
Utilisation d'une socket TCP



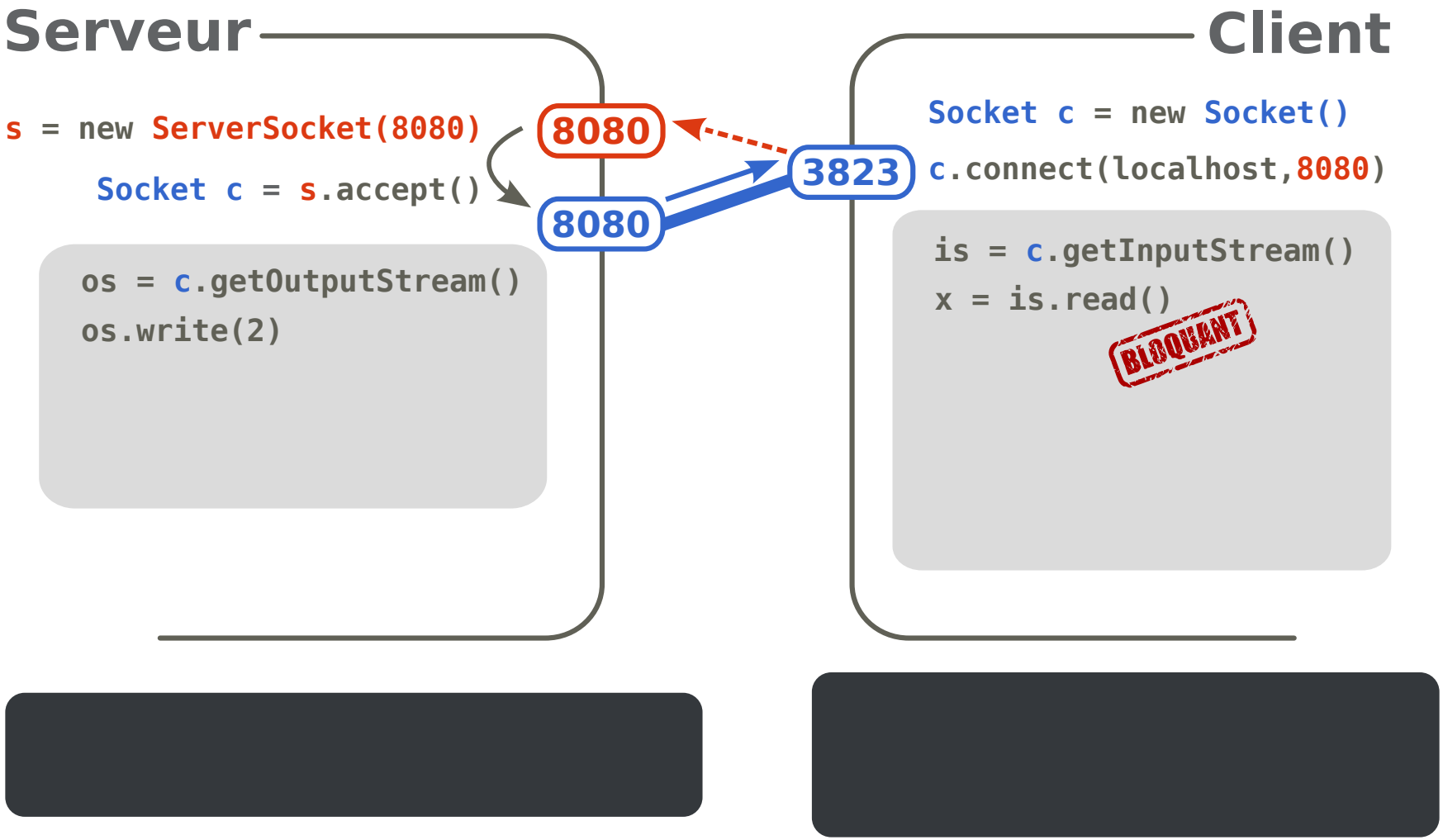
Utilisation d'une socket TCP



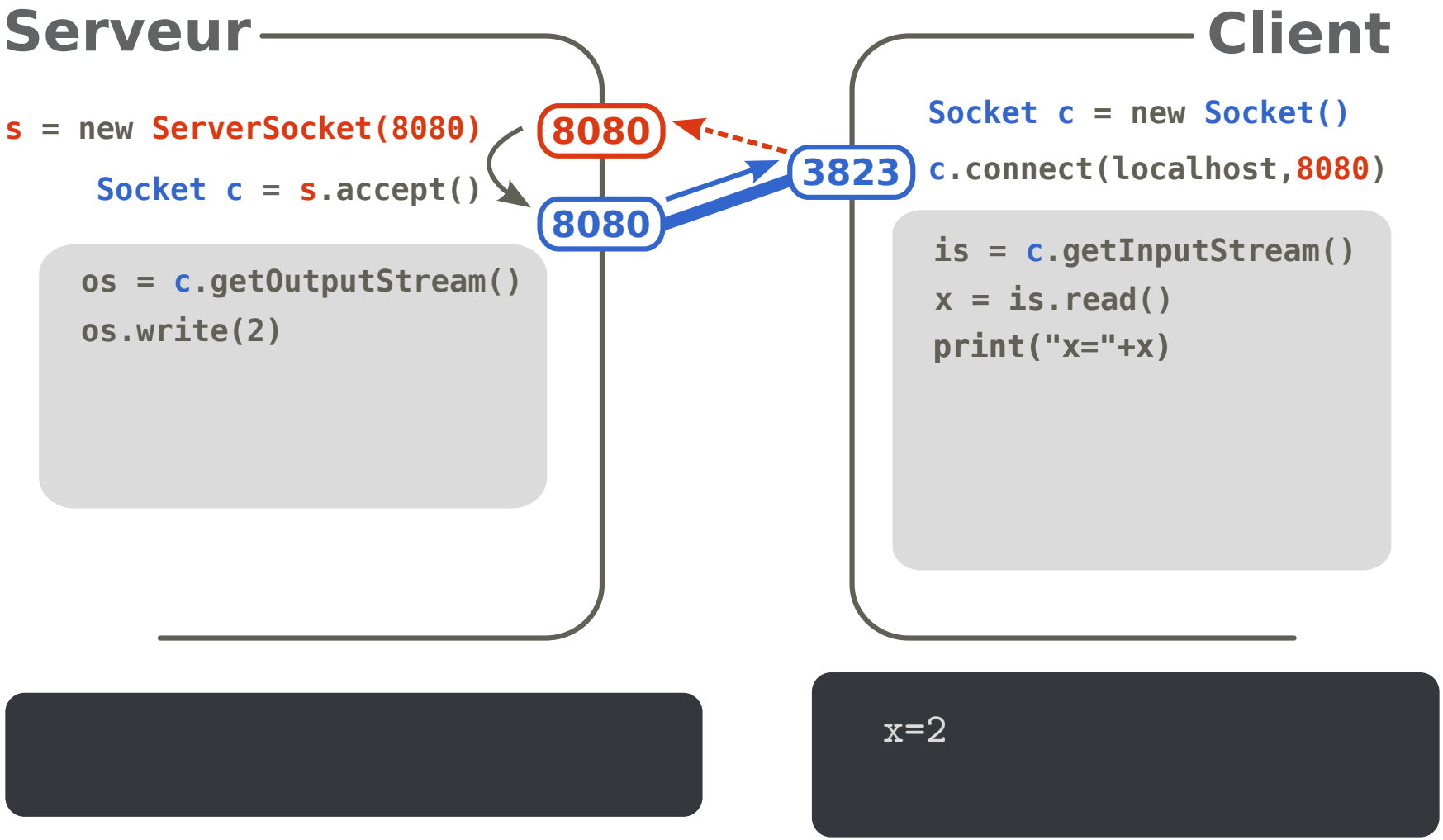
Utilisation d'une socket TCP



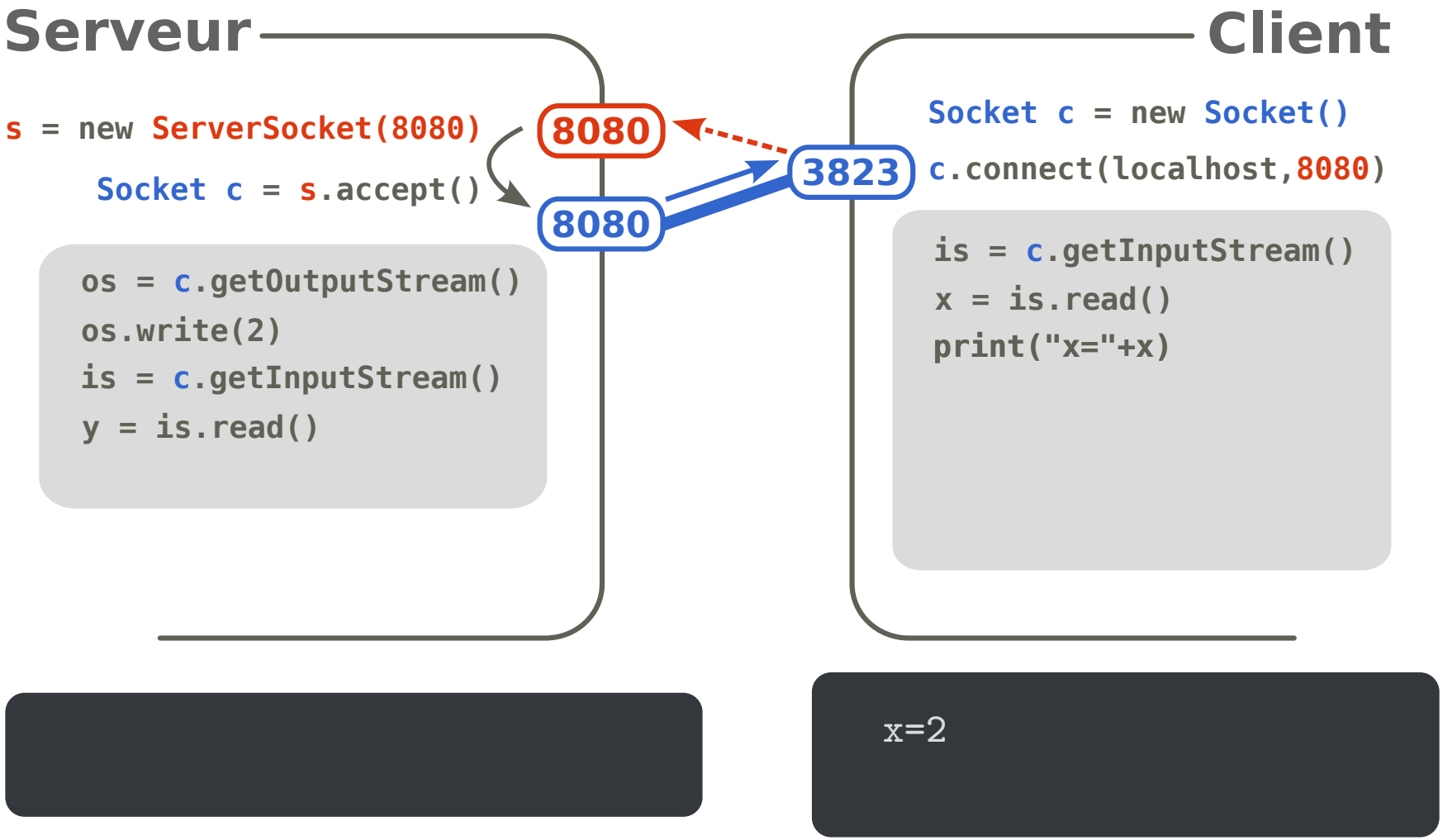
Utilisation d'une socket TCP



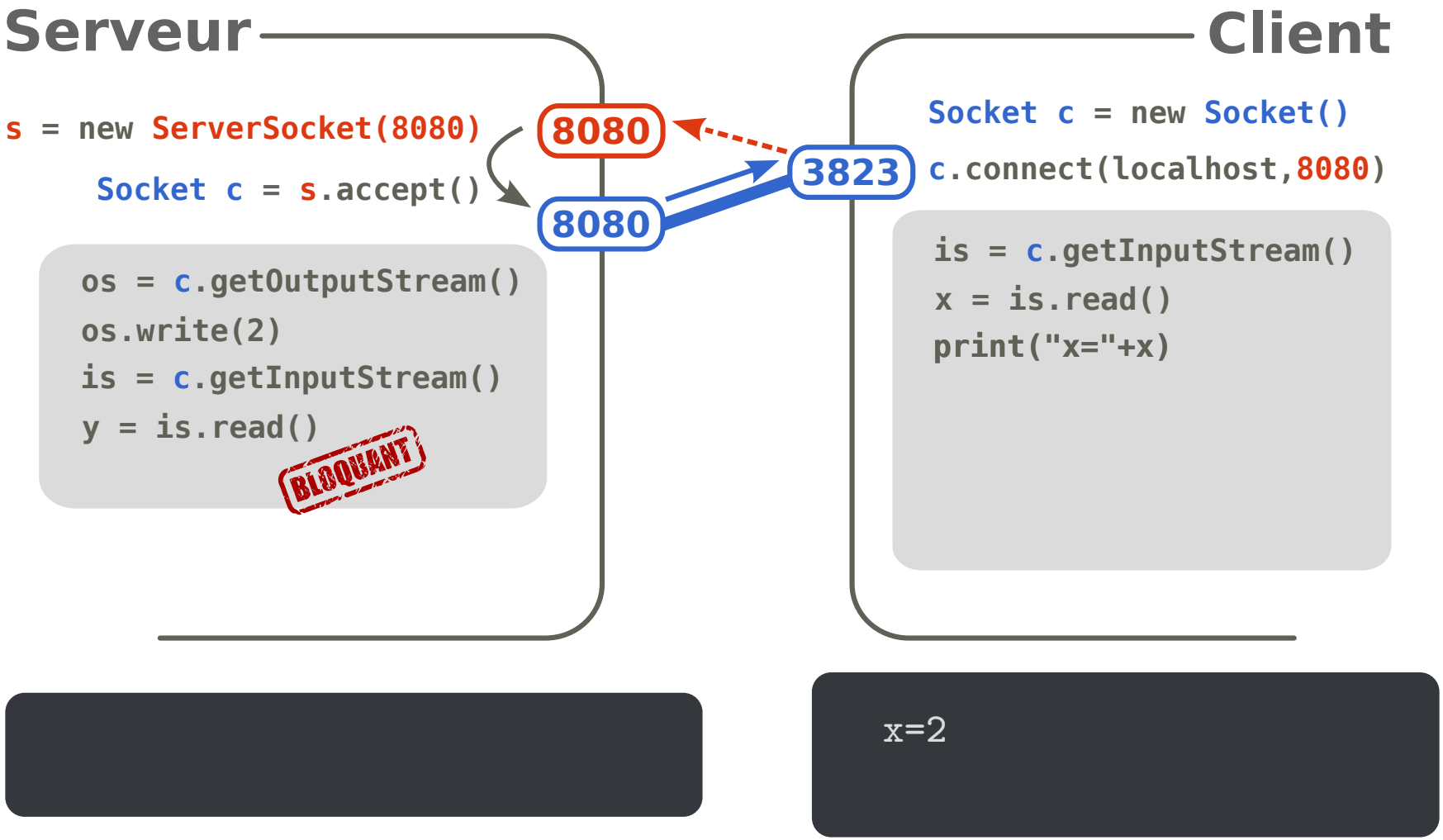
Utilisation d'une socket TCP



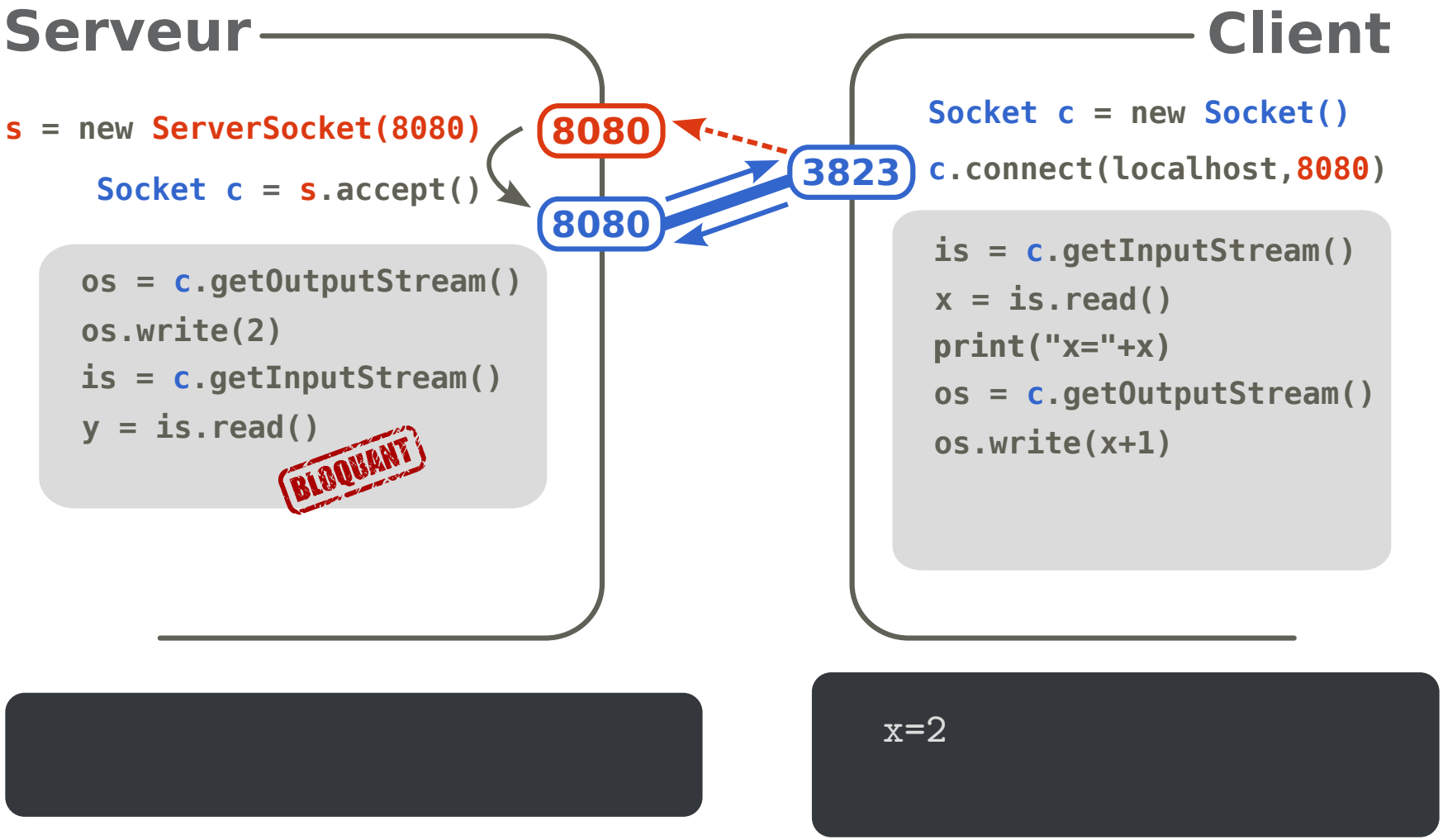
Utilisation d'une socket TCP



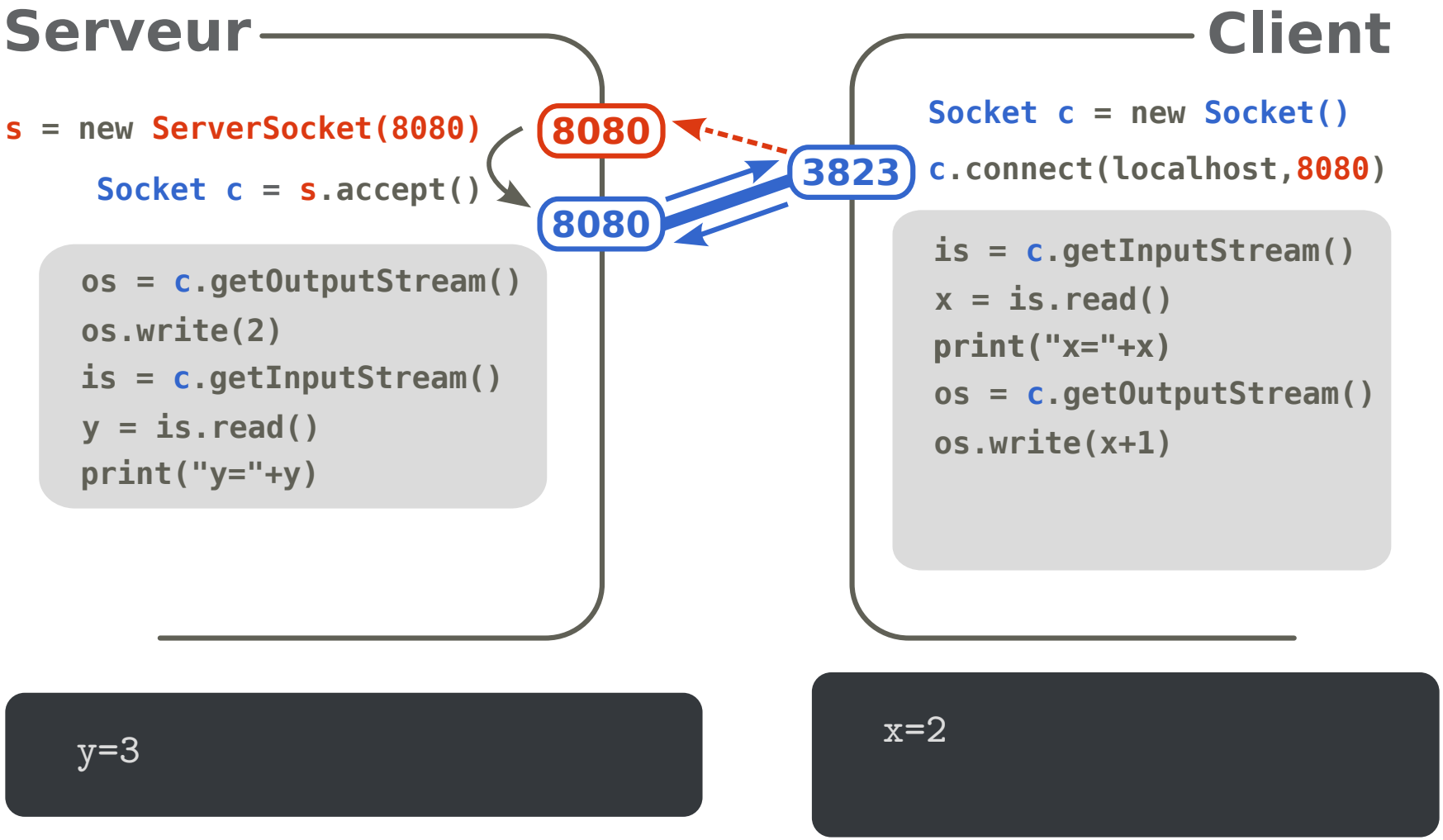
Utilisation d'une socket TCP



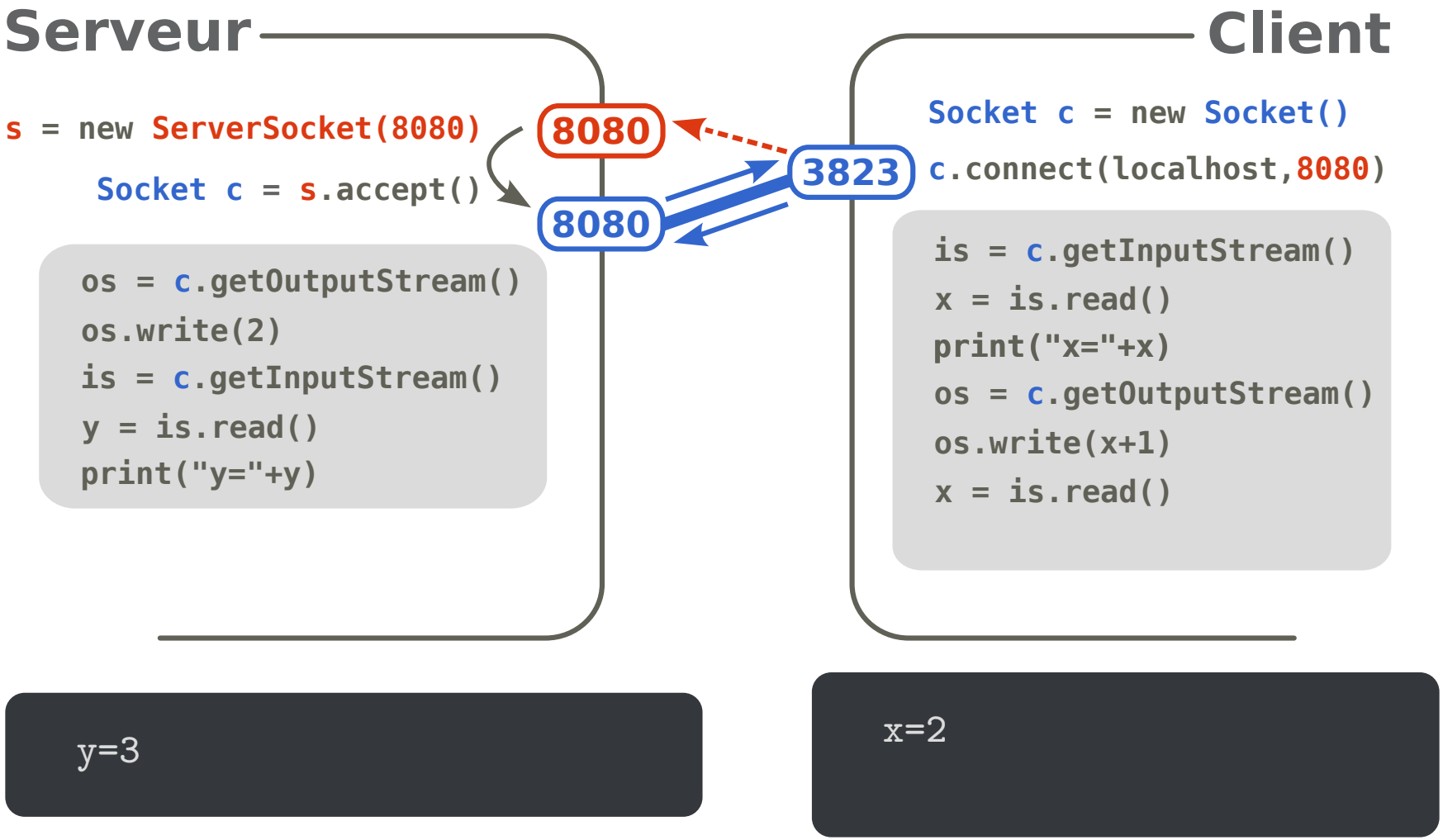
Utilisation d'une socket TCP



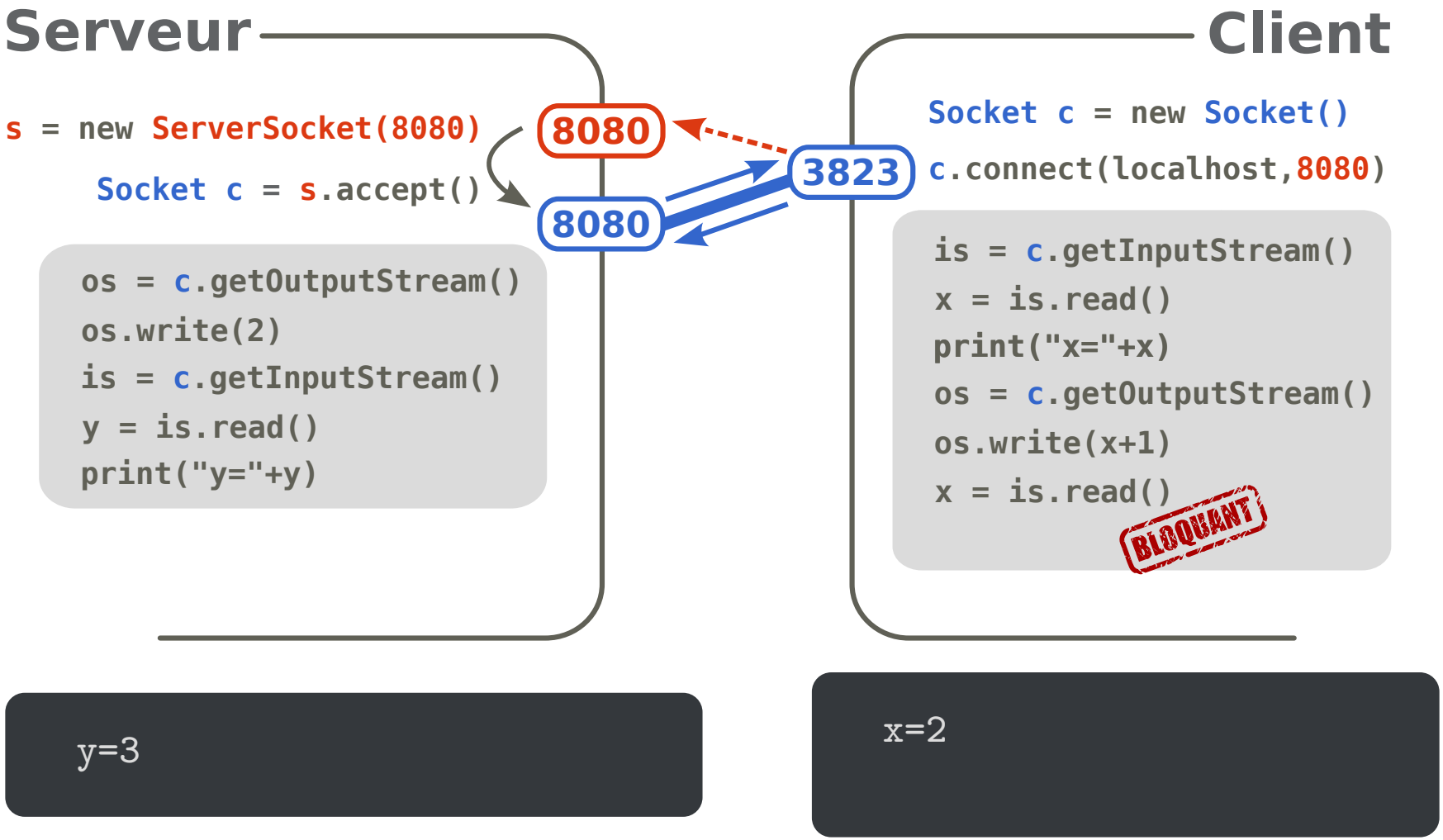
Utilisation d'une socket TCP



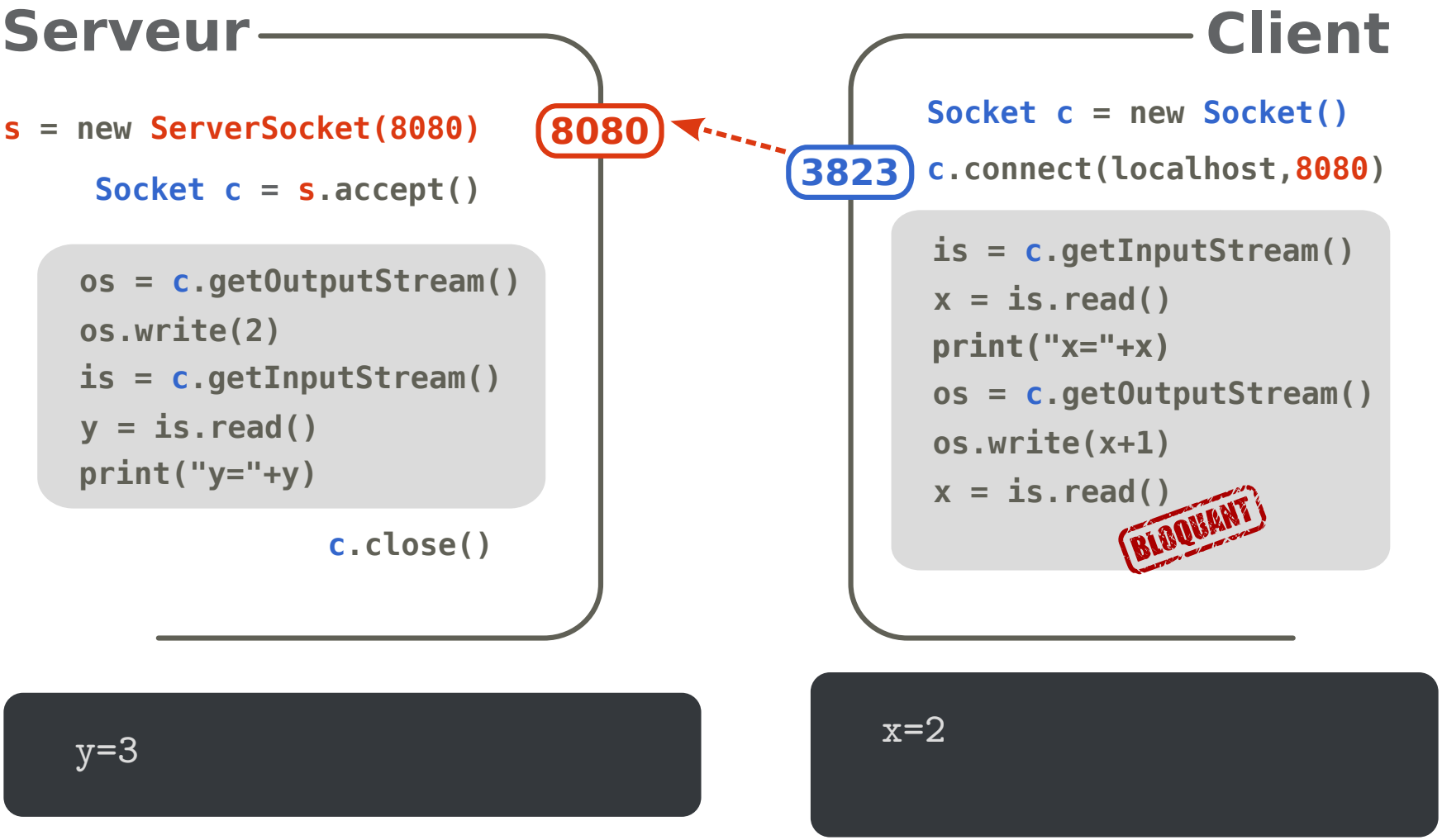
Utilisation d'une socket TCP



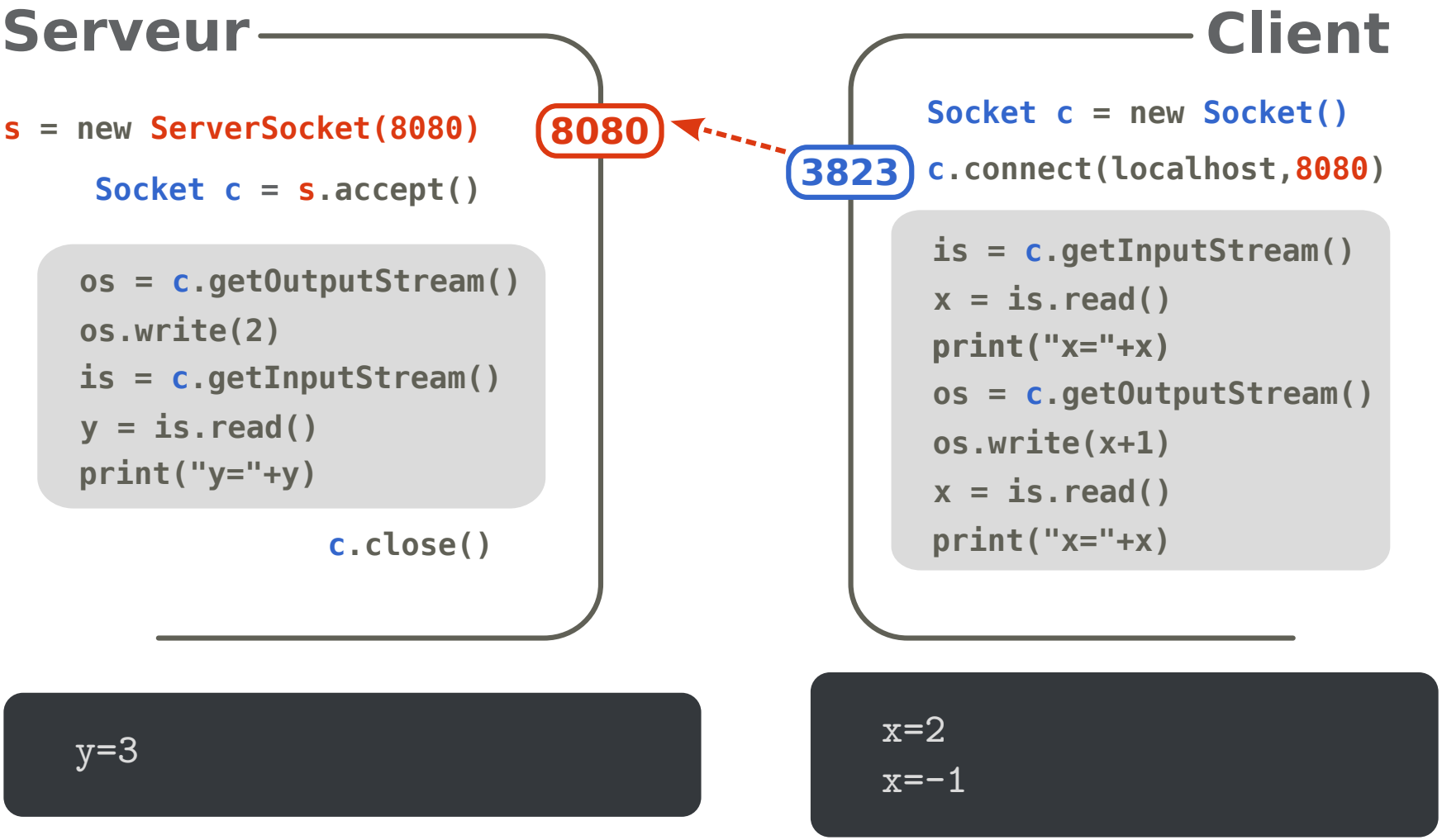
Utilisation d'une socket TCP



Utilisation d'une socket TCP



Utilisation d'une socket TCP



Utilisation d'une socket TCP

Serveur

```
s = new ServerSocket(8080)
```

8080

```
Socket c = s.accept()
```

```
os = c.getOutputStream()
os.write(2)
is = c.getInputStream()
y = is.read()
print("y="+y)
```

```
c.close()
```

```
y=3
```

Client

```
Socket c = new Socket()
c.connect(localhost,8080)
```

```
is = c.getInputStream()
x = is.read()
print("x="+x)
os = c.getOutputStream()
os.write(x+1)
x = is.read()
print("x="+x)
```

```
c.close()
```

```
x=2
x=-1
```

Utilisation d'une socket TCP

Serveur

```
s = new ServerSocket(8080)
     Socket c = s.accept()
  os = c.getOutputStream()
  os.write(2)
  is = c.getInputStream()
  y = is.read()
  print("y="+y)
                  c.close()
                  s.close()
```

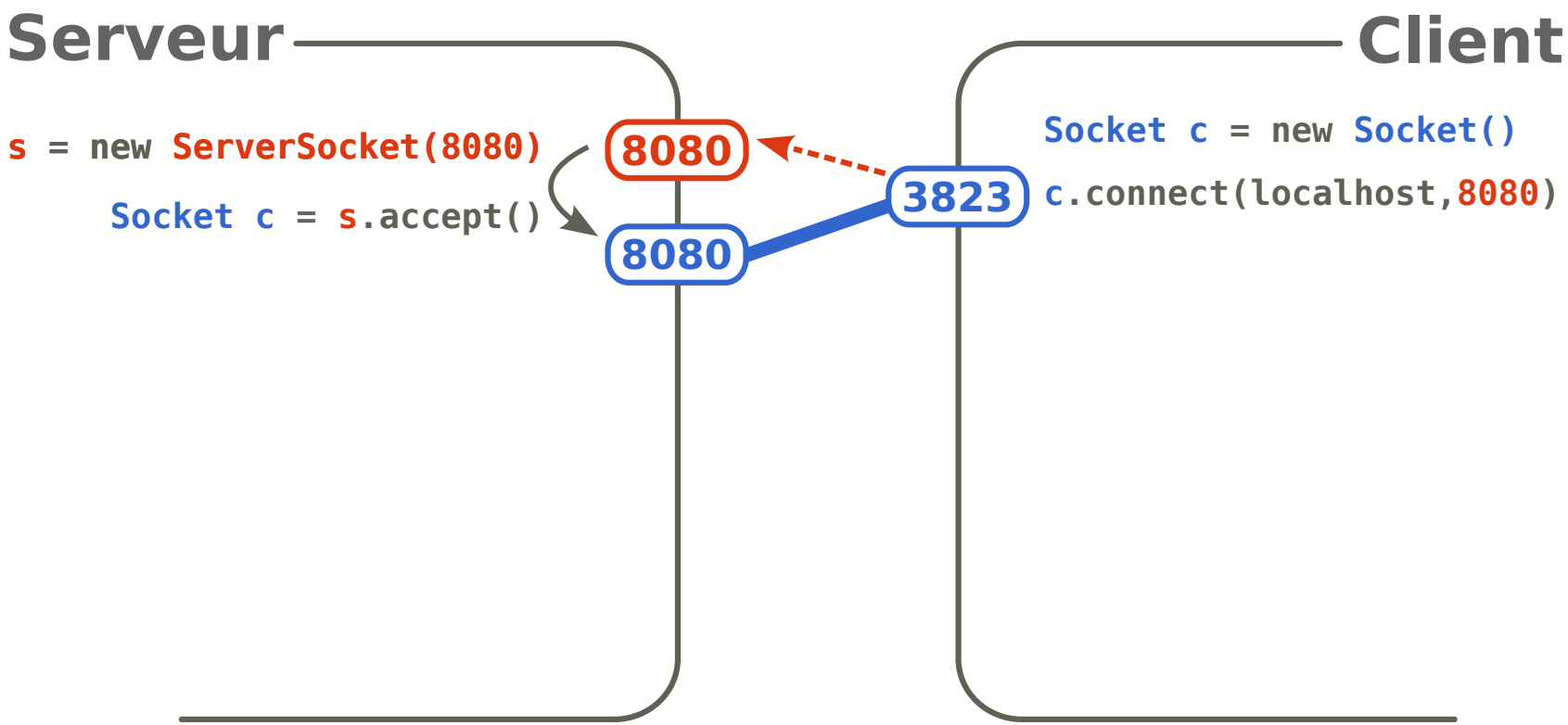
```
y=3
```

Client

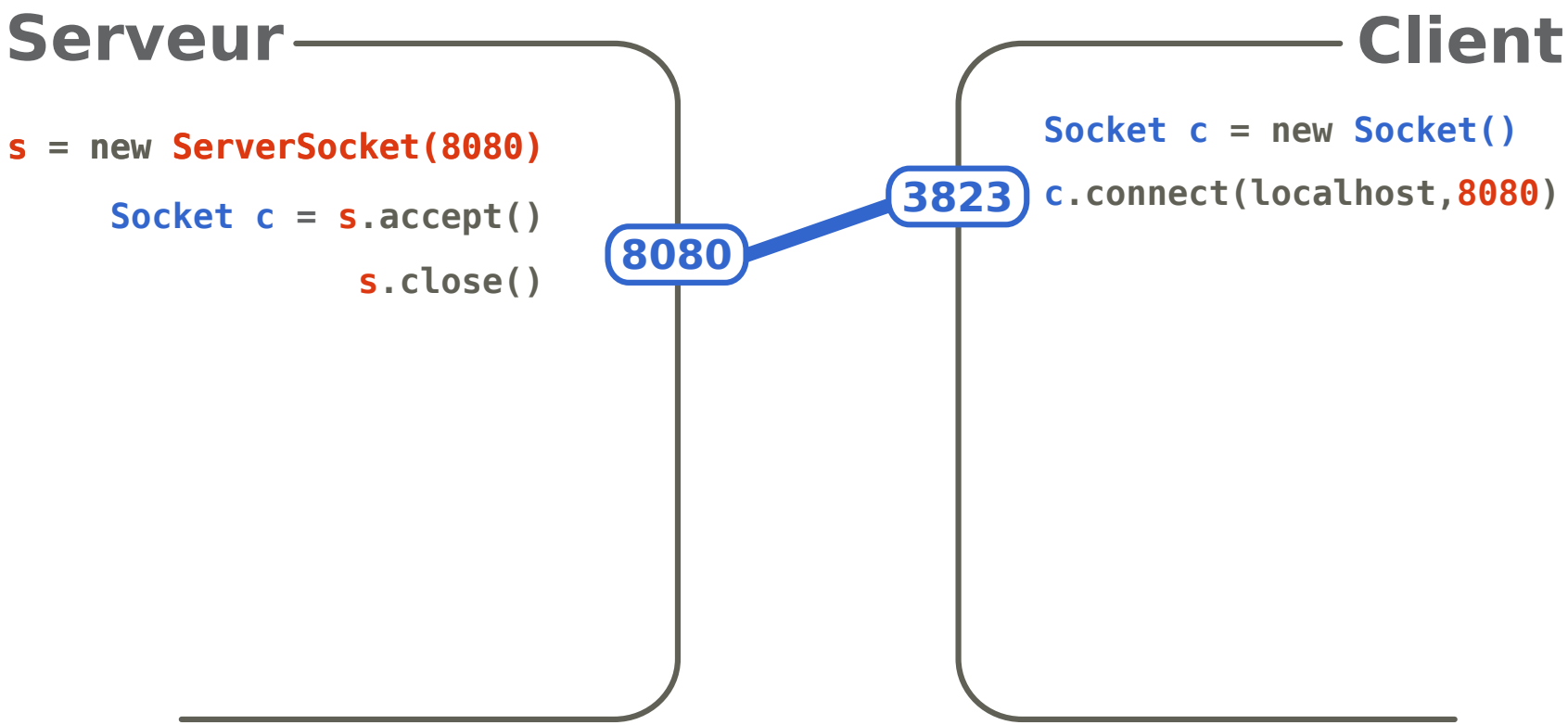
```
Socket c = new Socket()
c.connect(localhost,8080)
  is = c.getInputStream()
  x = is.read()
  print("x="+x)
  os = c.getOutputStream()
  os.write(x+1)
  x = is.read()
  print("x="+x)
c.close()
```

```
x=2
x=-1
```

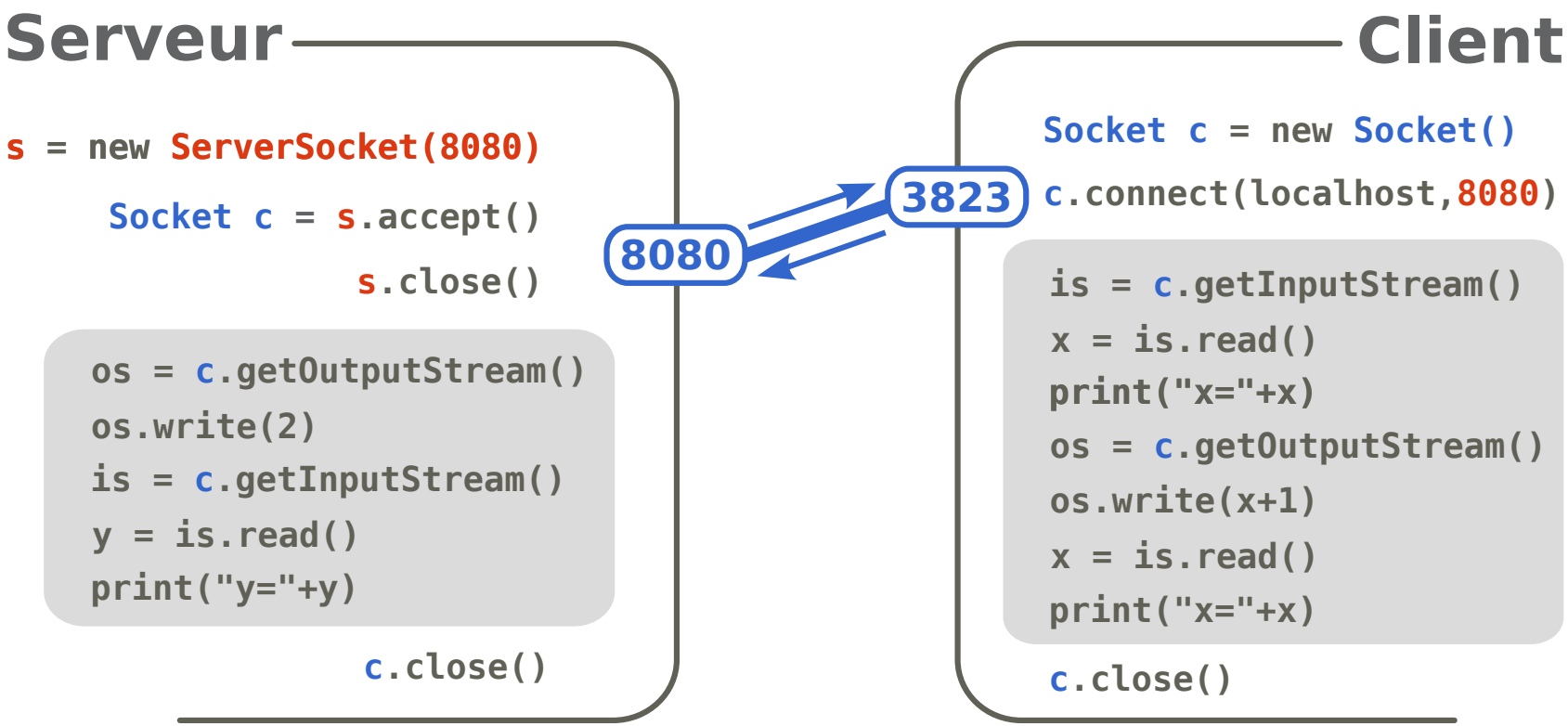
Fermeture de la socket de connexion



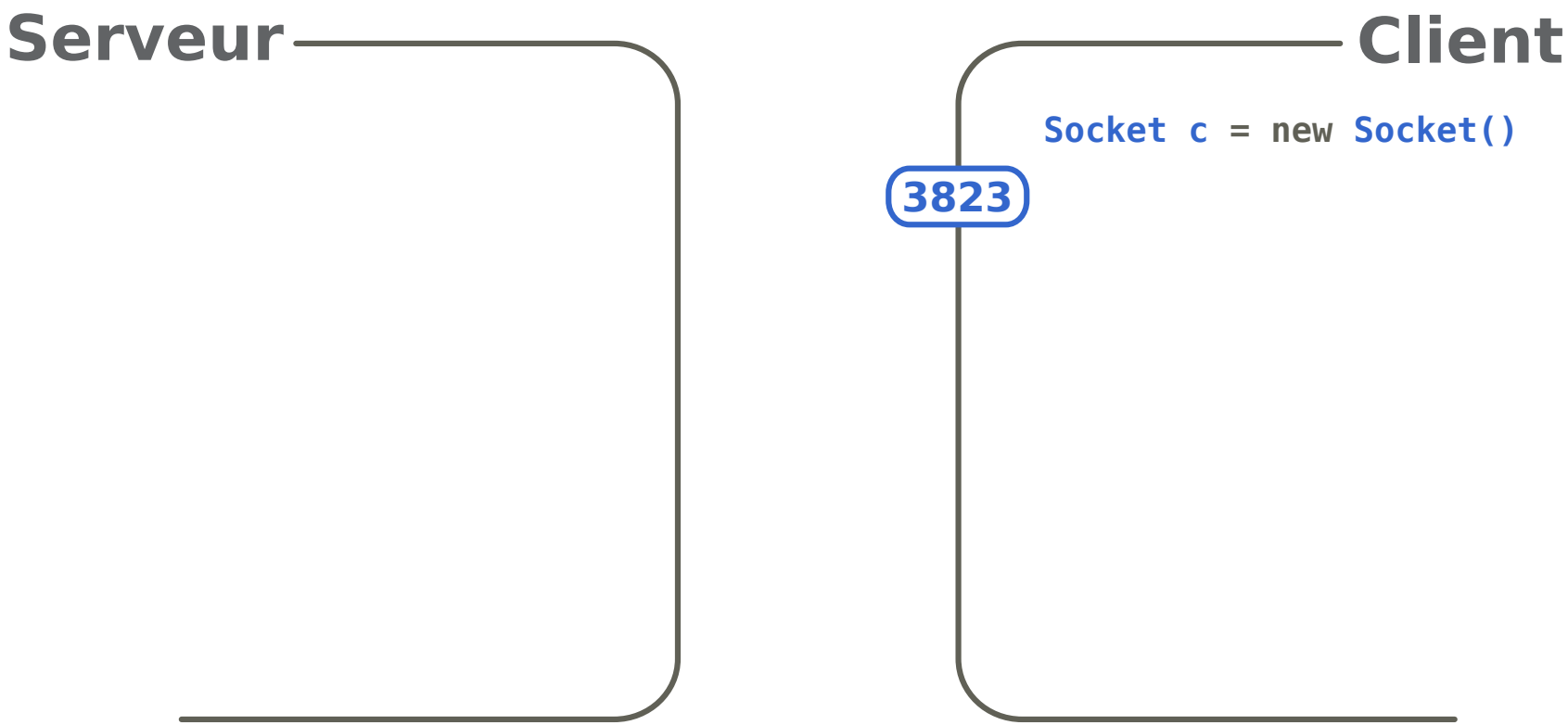
Fermeture de la socket de connexion



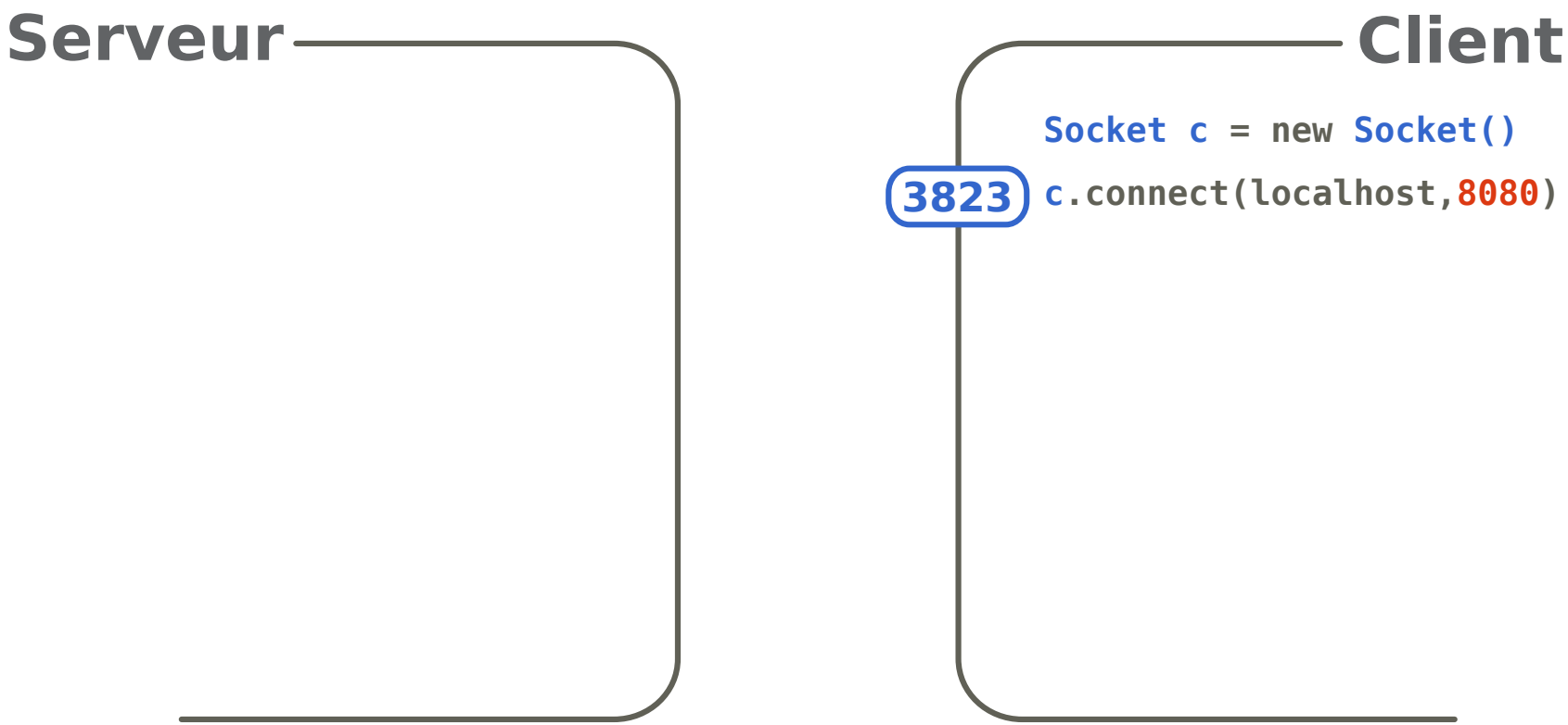
Fermeture de la socket de connexion



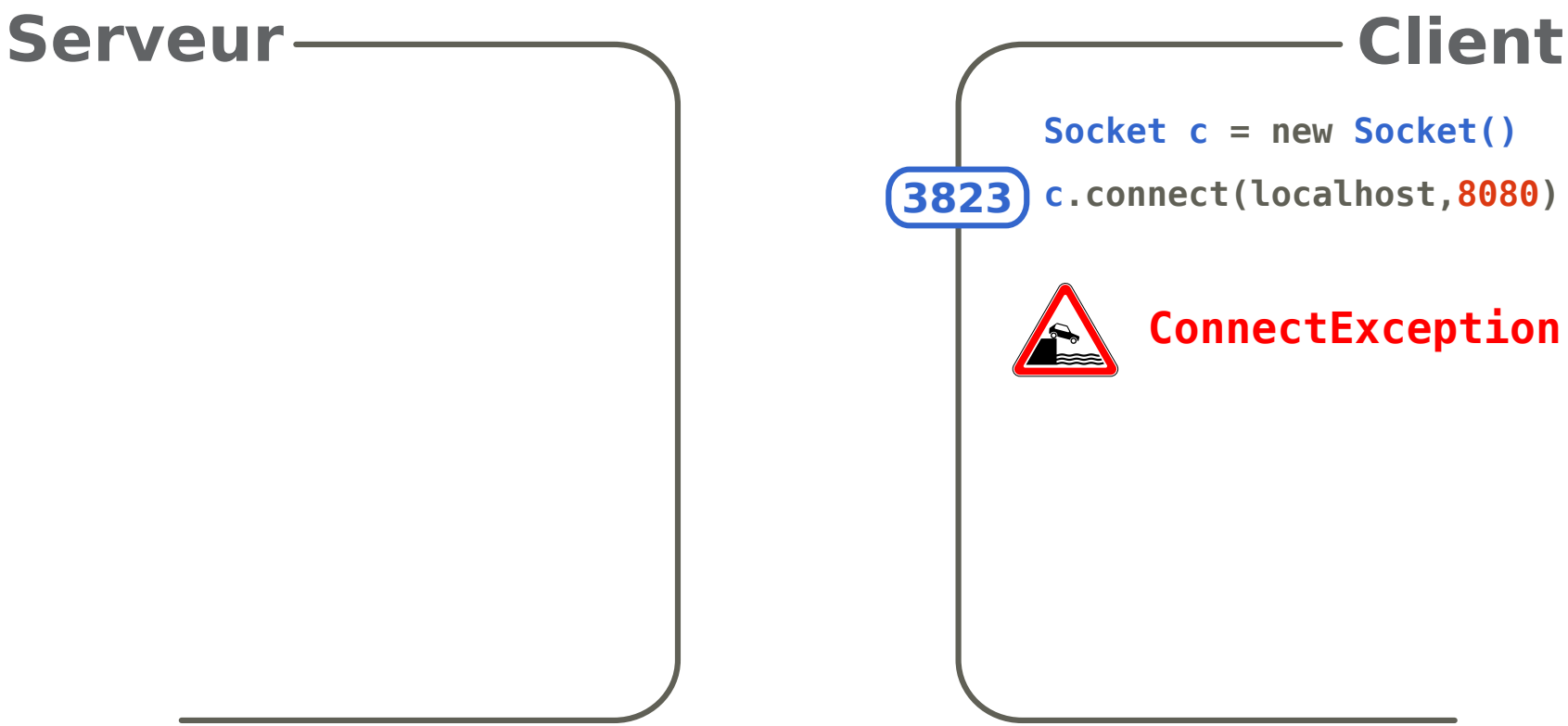
Connexion précoce : pas de socket de connexion



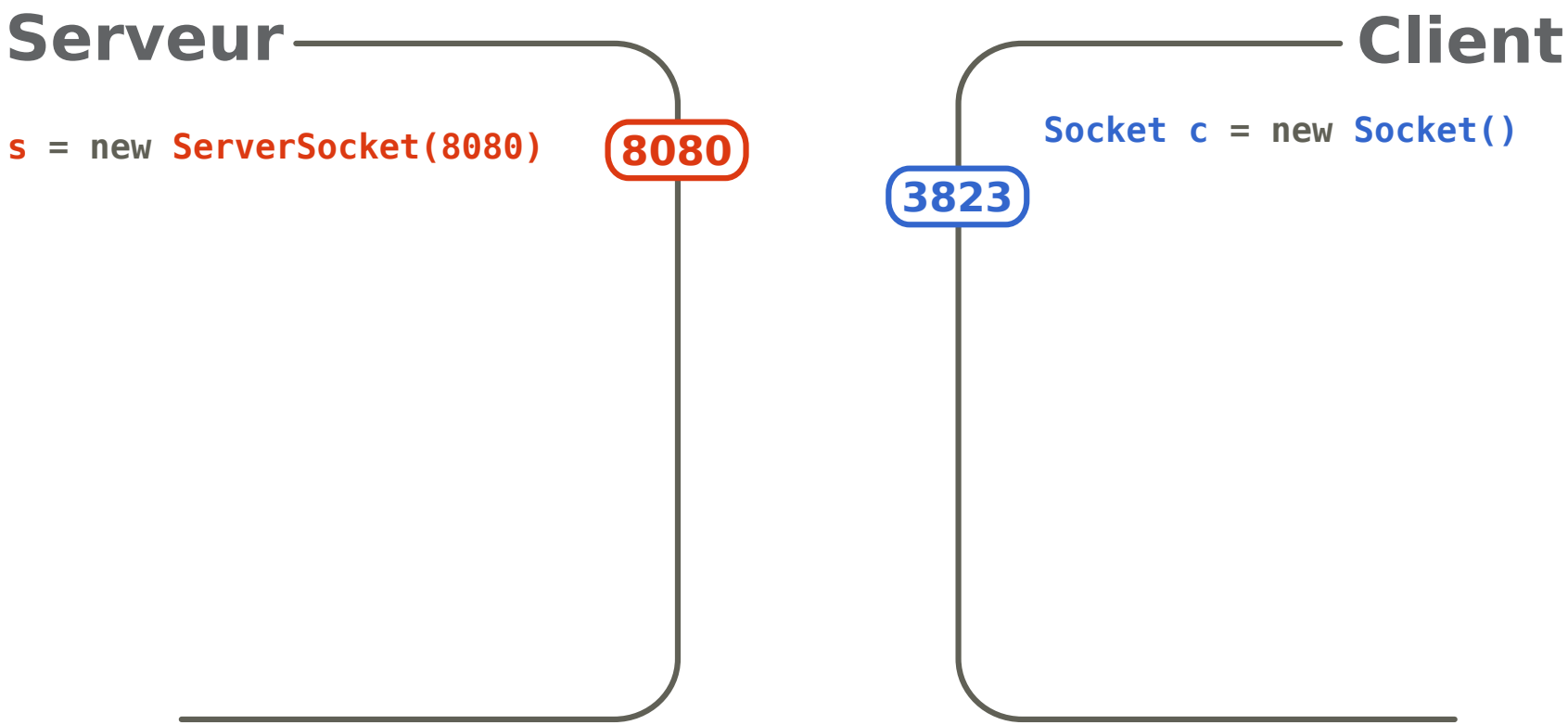
Connexion précoce : pas de socket de connexion



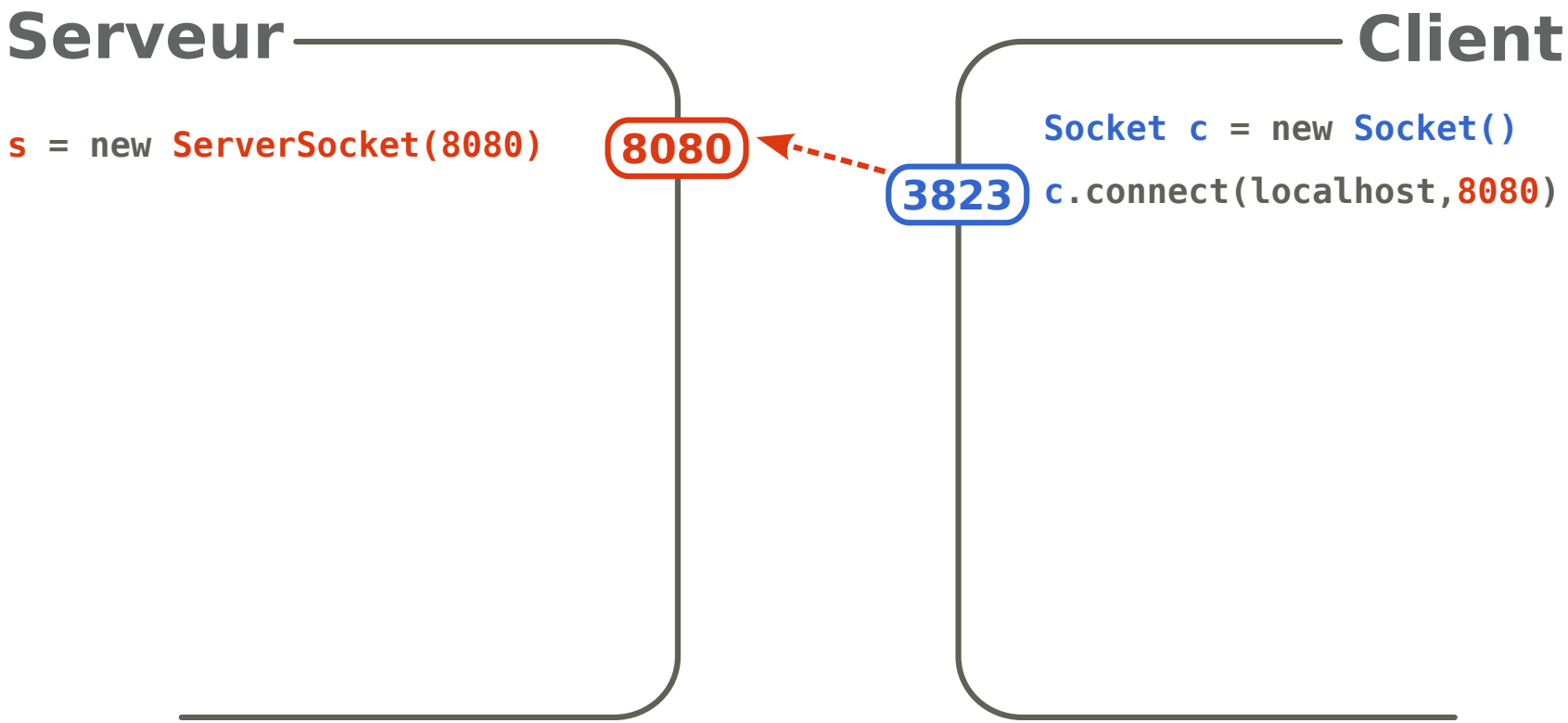
Connexion précoce : pas de socket de connexion



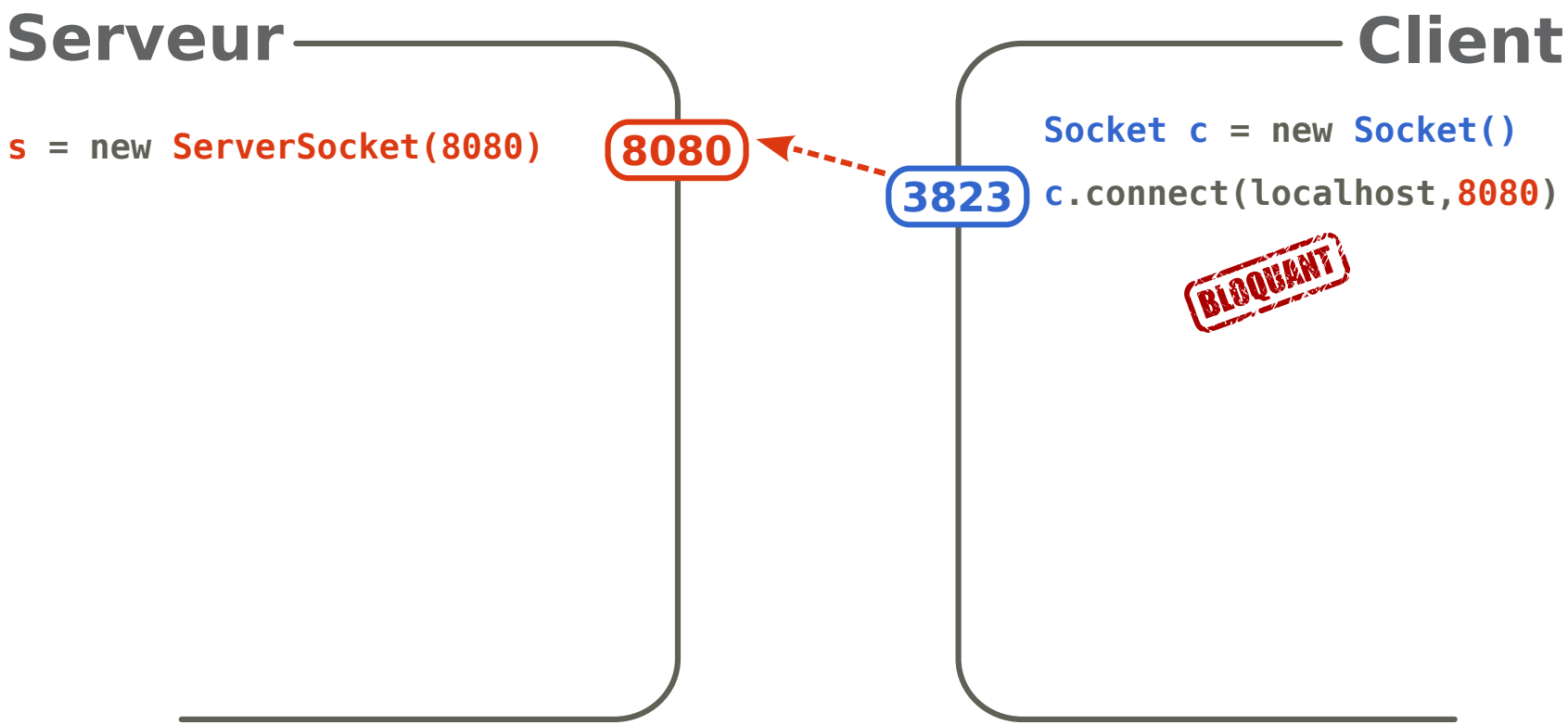
Connexion précoce : avant le `accept` du serveur



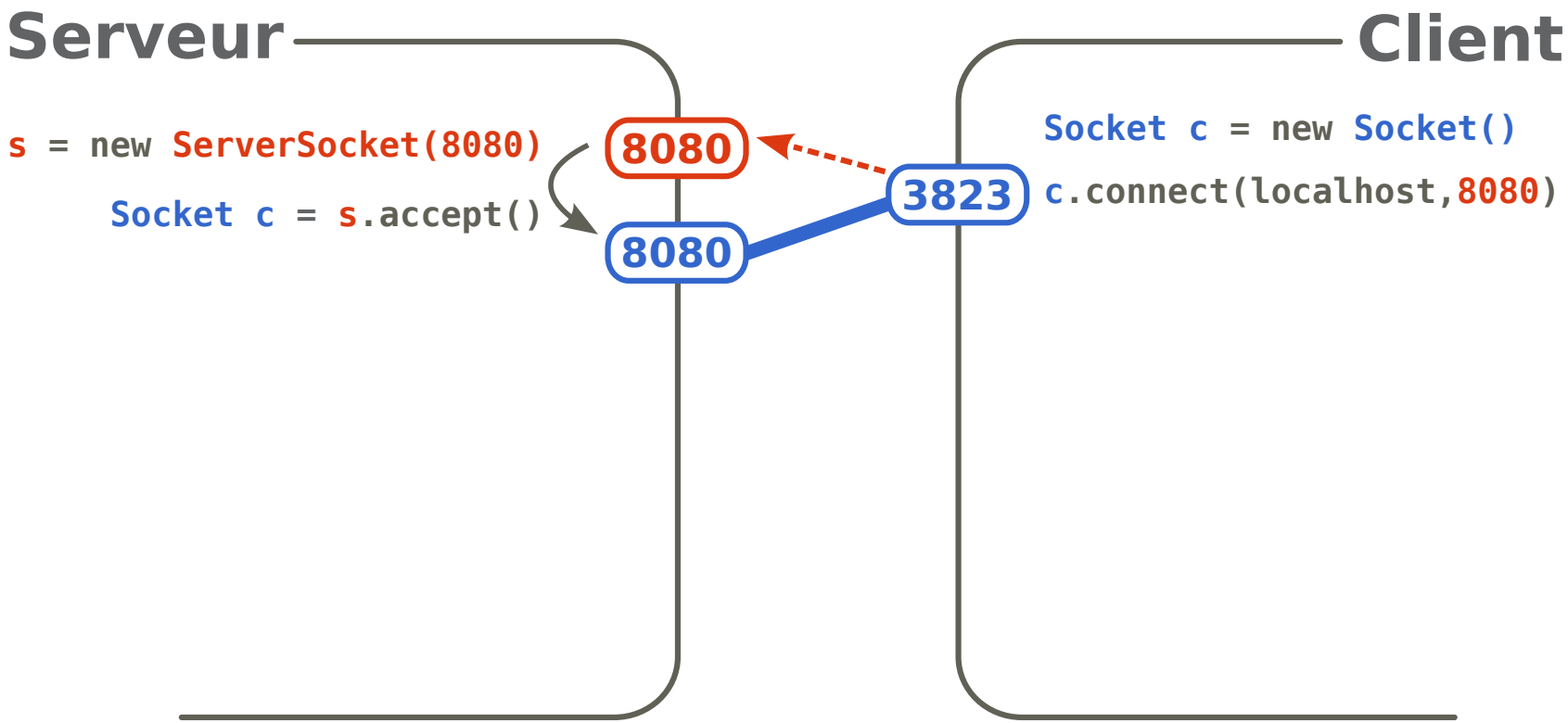
Connexion précoce : avant le `accept` du serveur



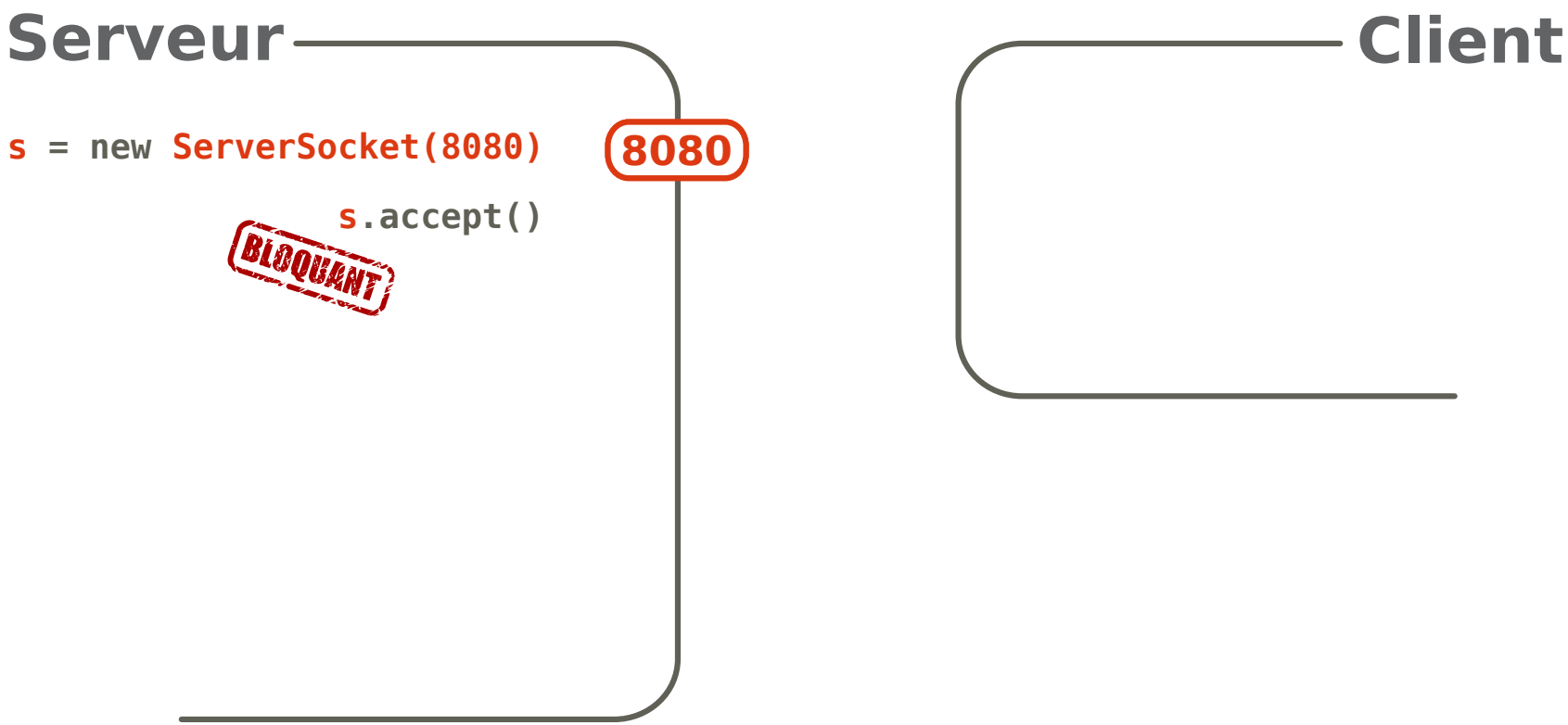
Connexion précoce : avant le `accept` du serveur



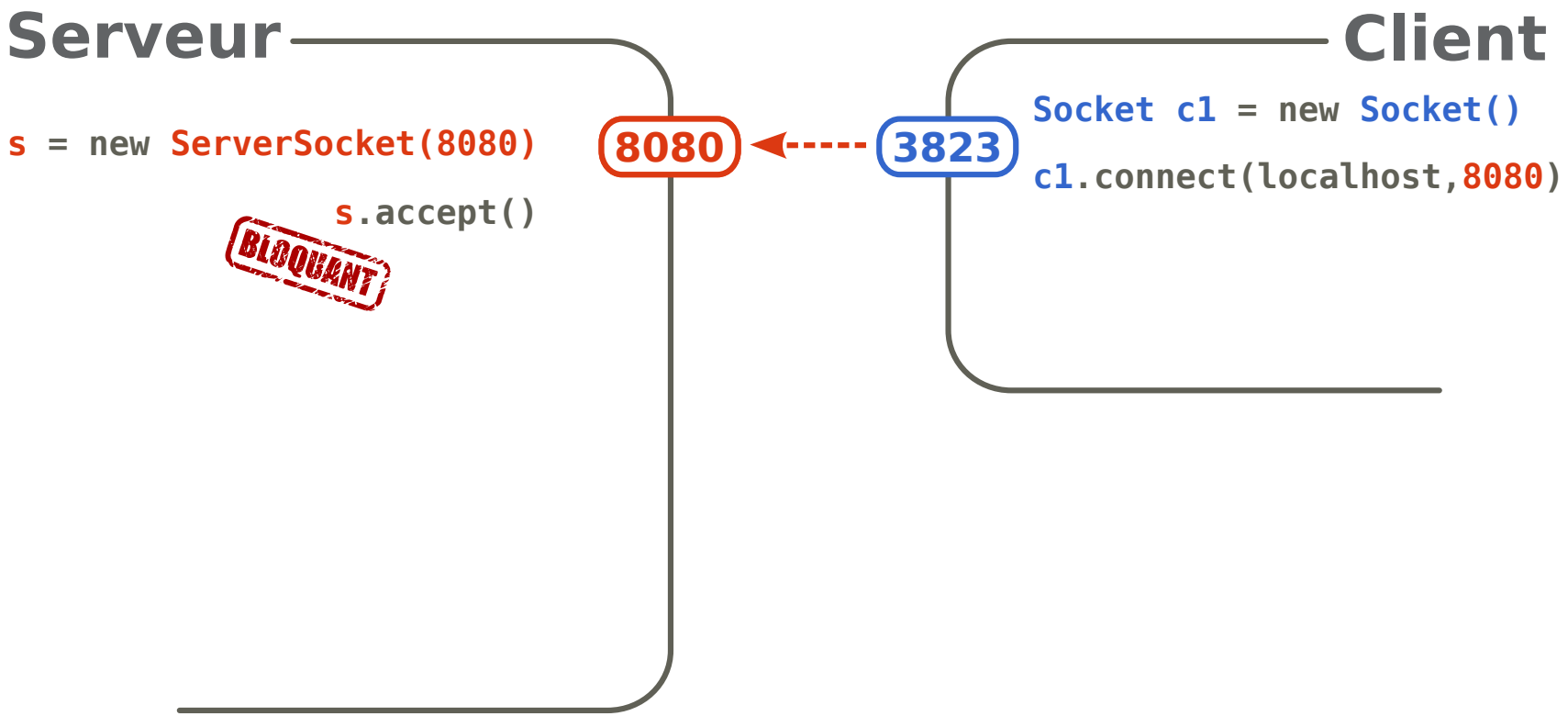
Connexion précoce : avant le `accept` du serveur



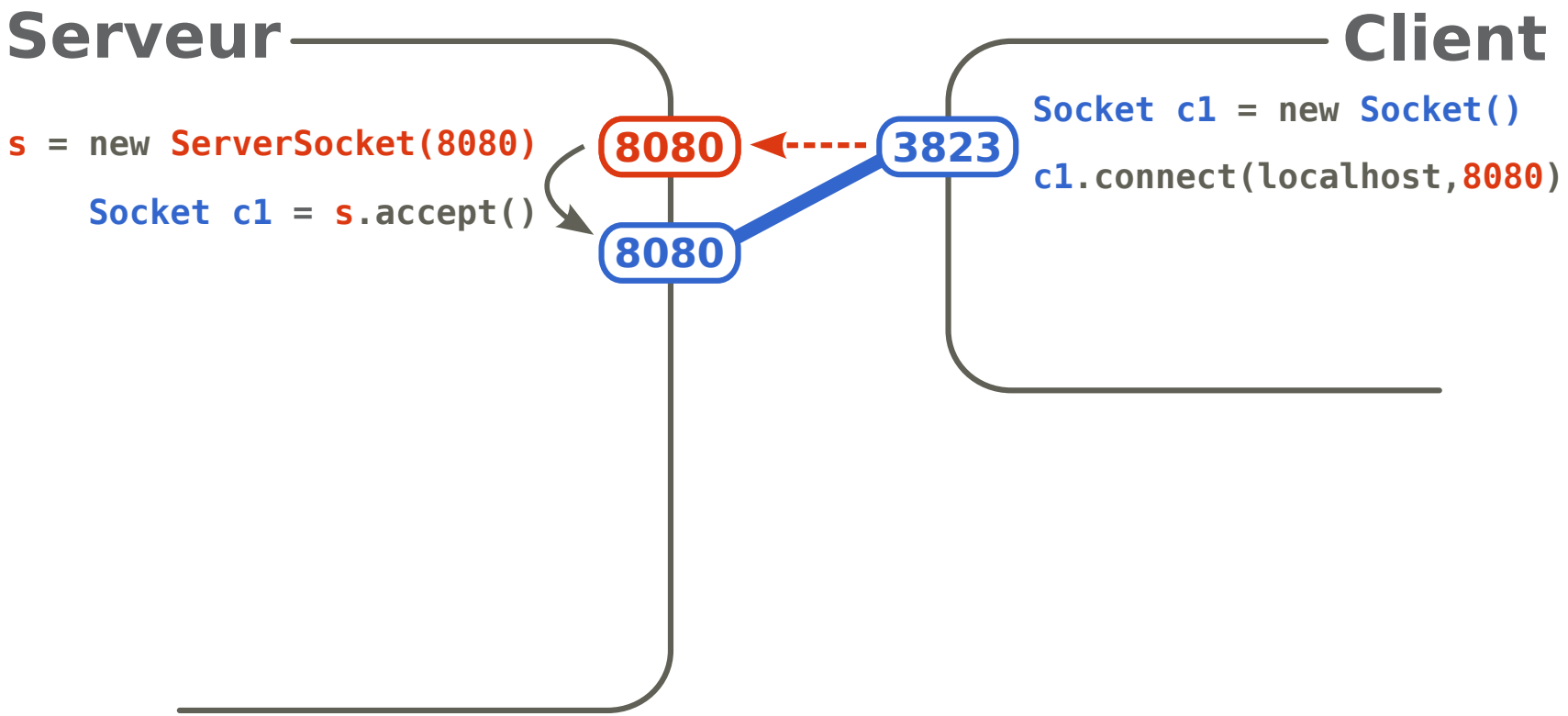
Connexions multiples sur une socket TCP



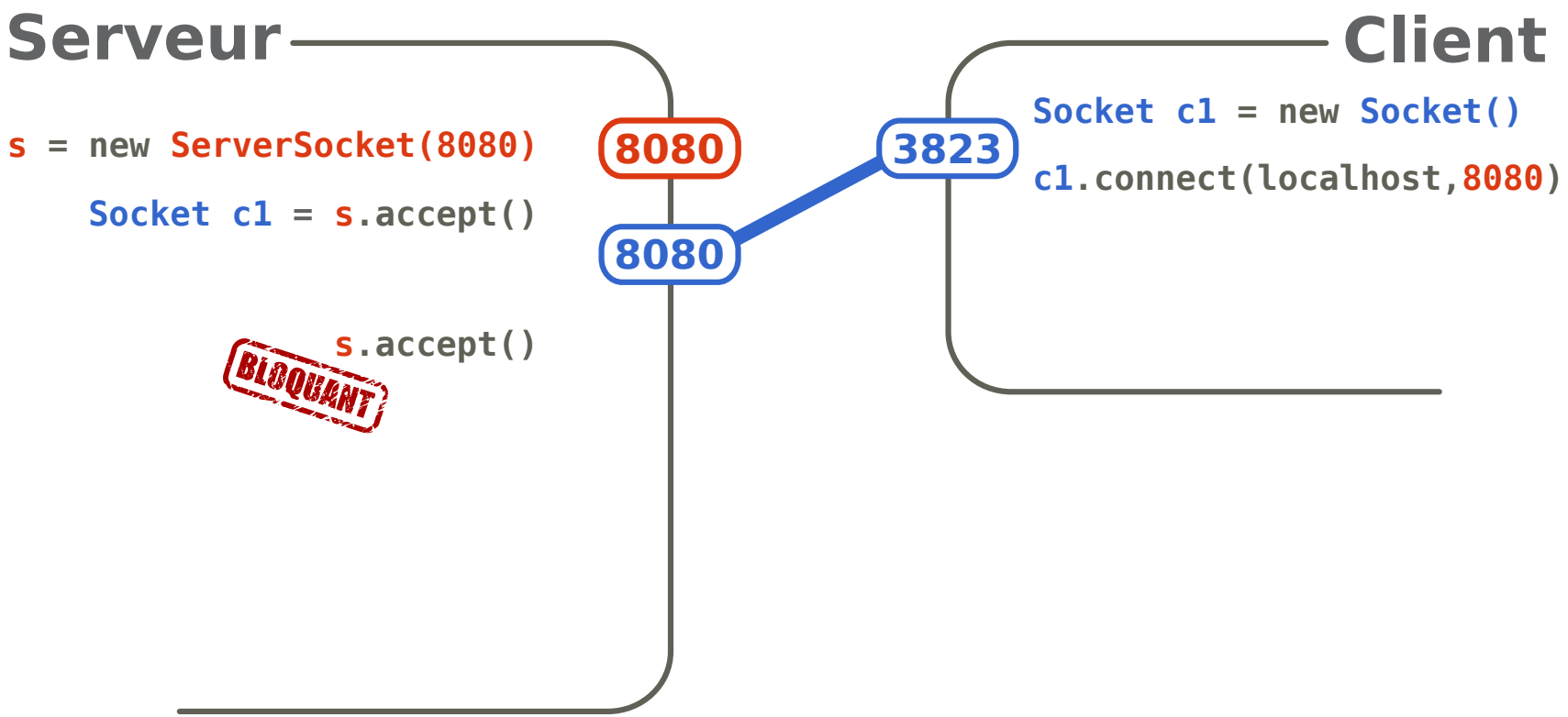
Connexions multiples sur une socket TCP



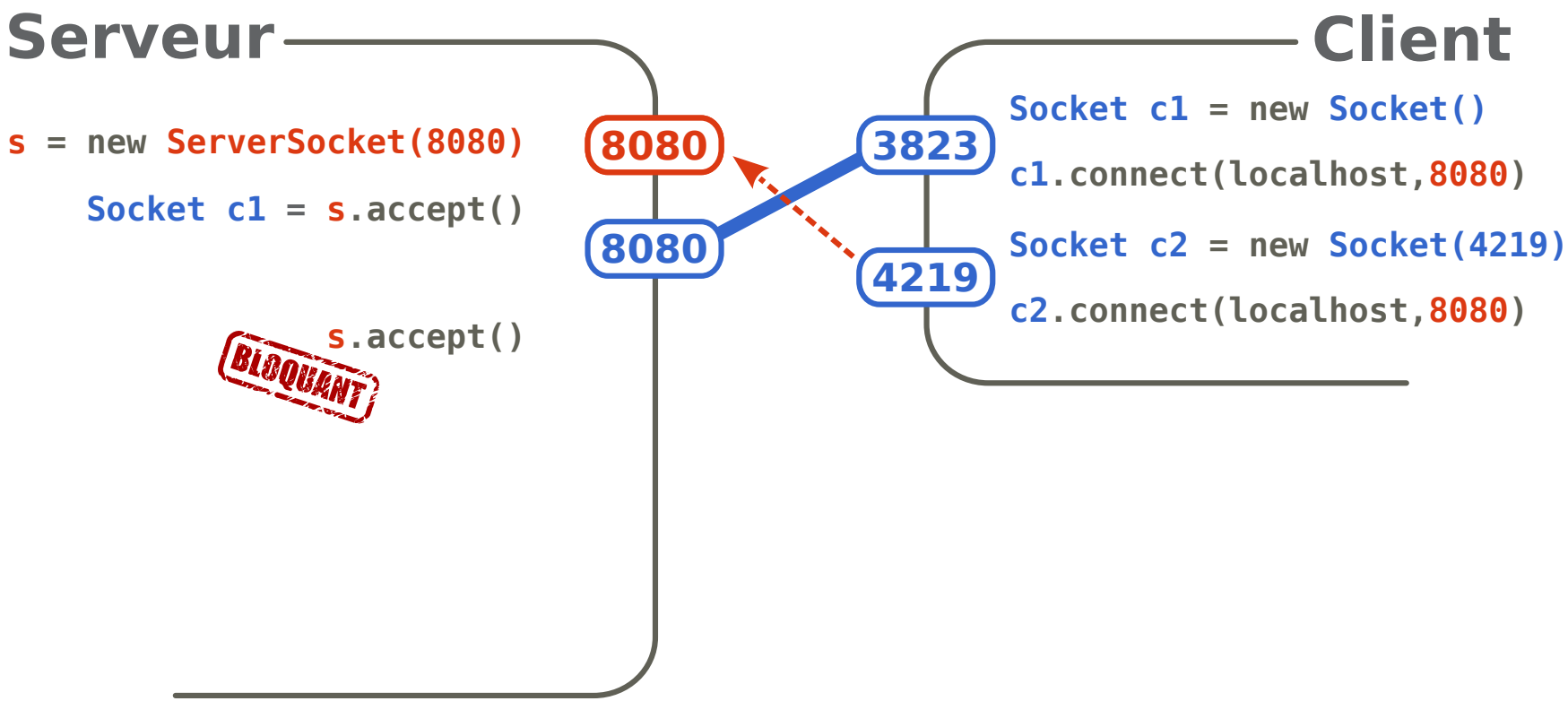
Connexions multiples sur une socket TCP



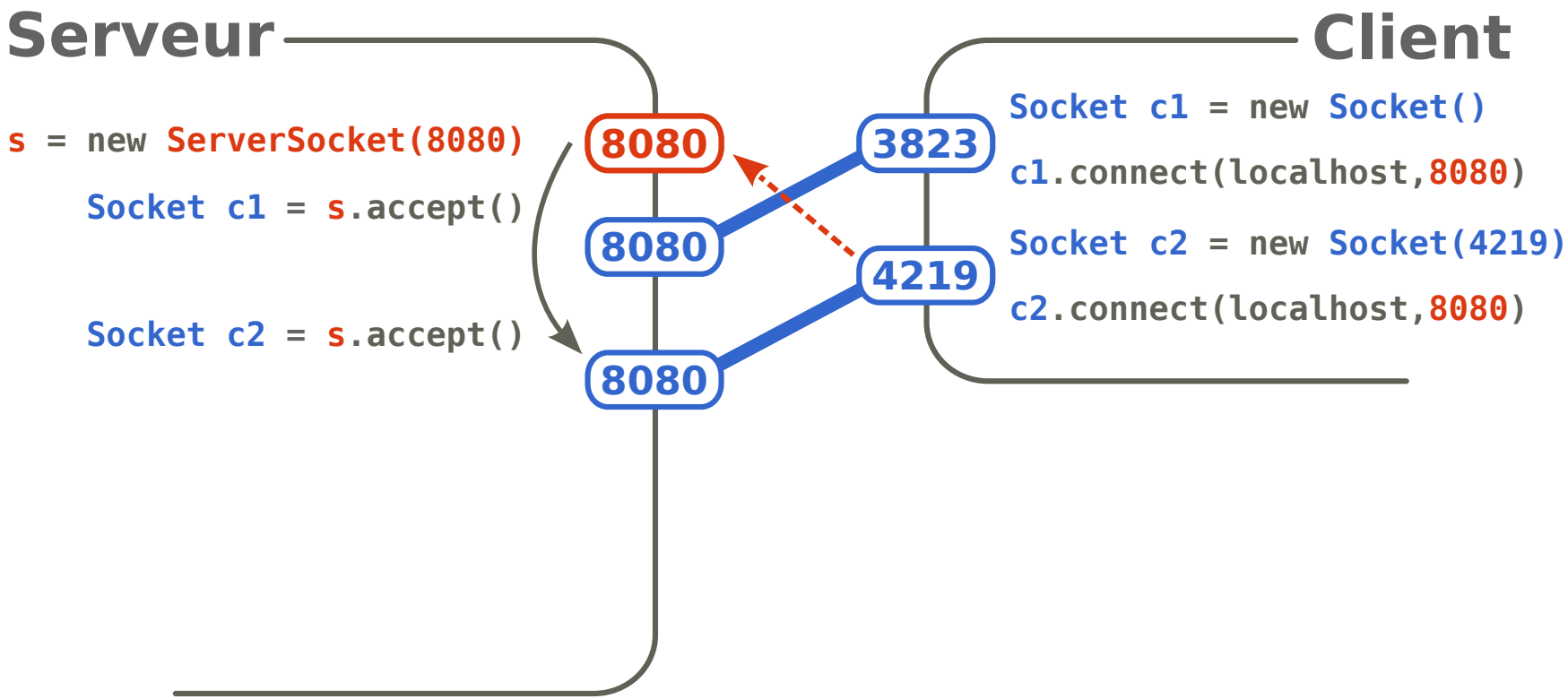
Connexions multiples sur une socket TCP



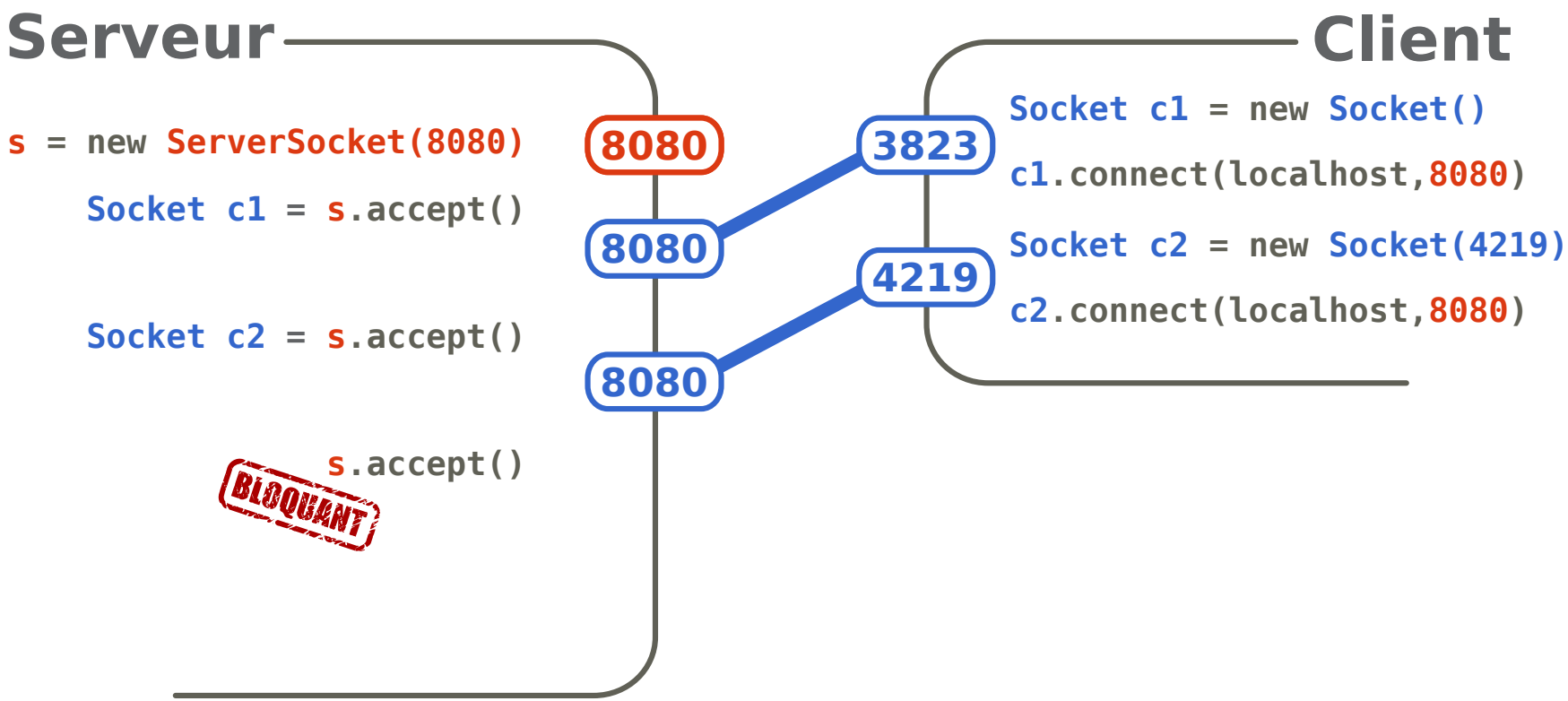
Connexions multiples sur une socket TCP



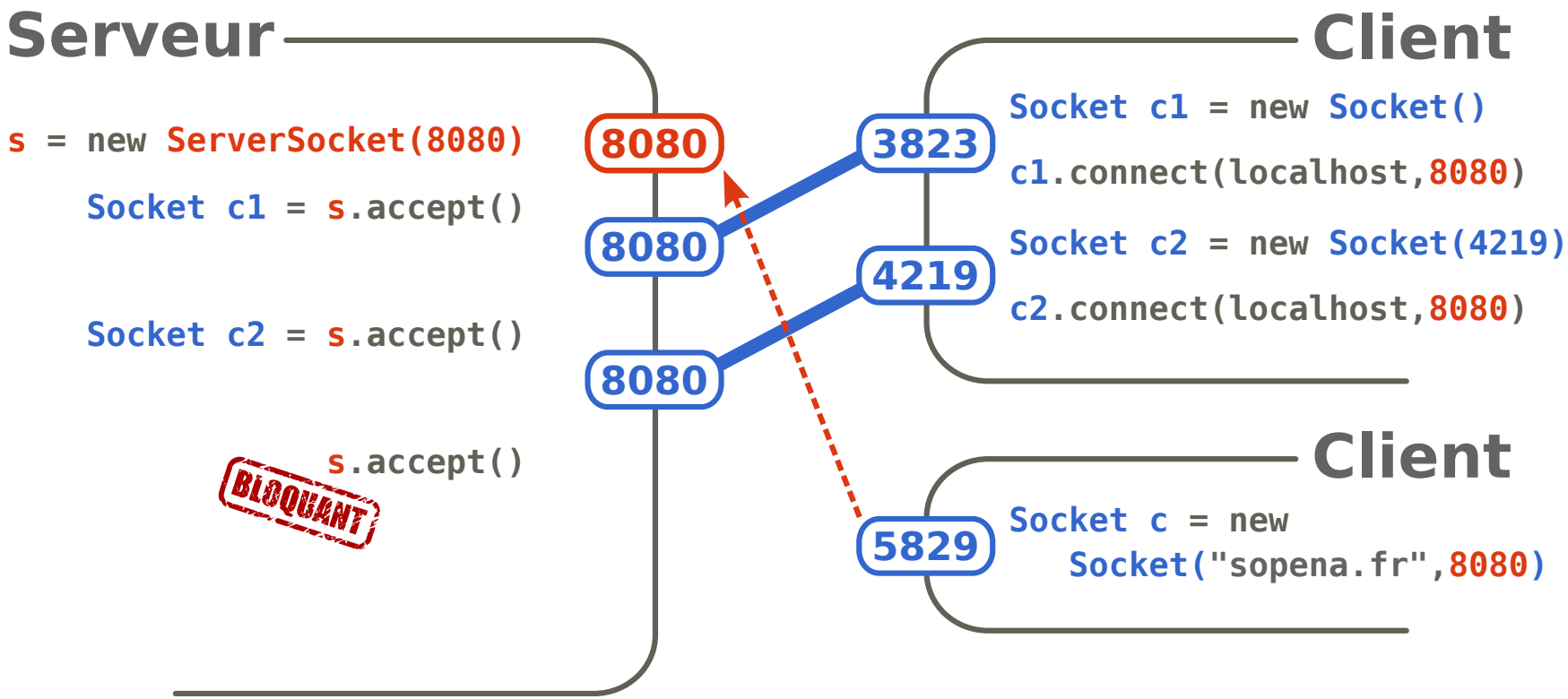
Connexions multiples sur une socket TCP



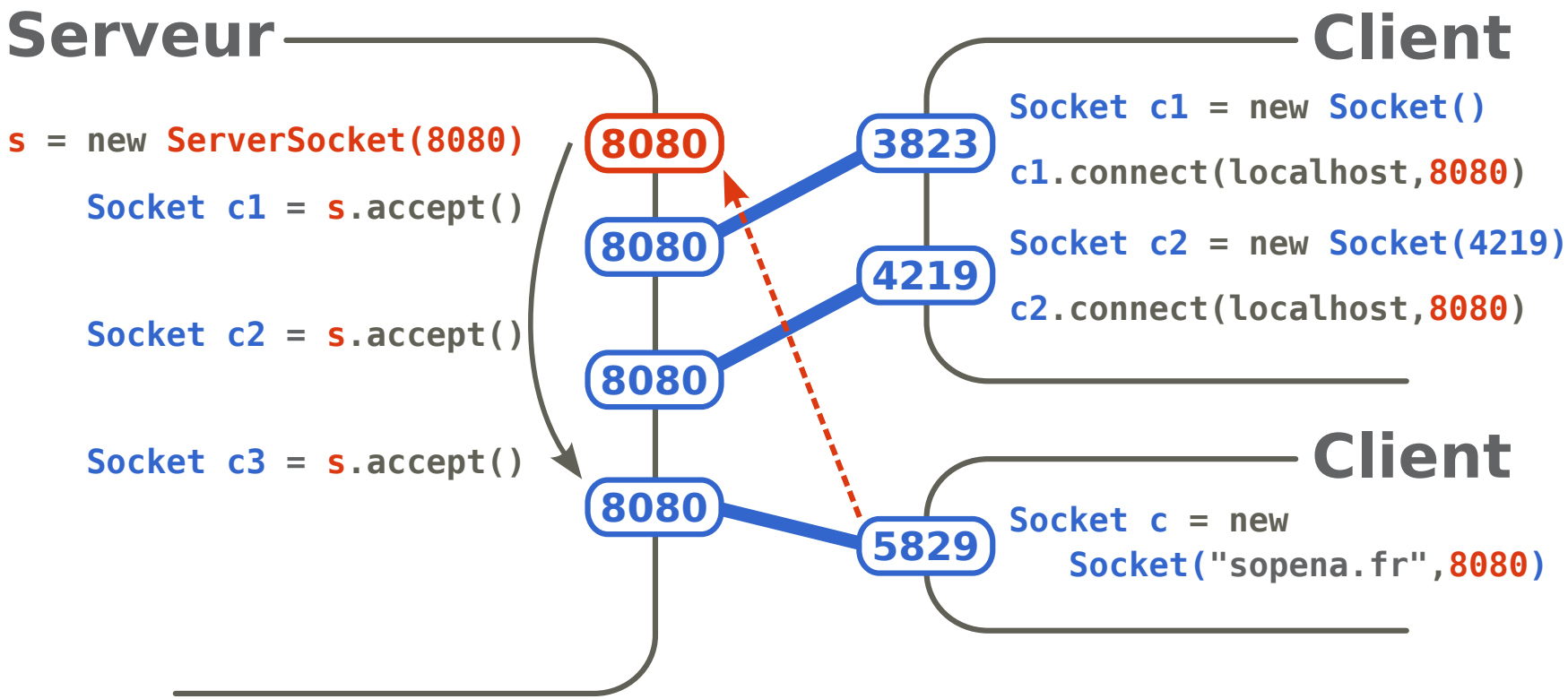
Connexions multiples sur une socket TCP



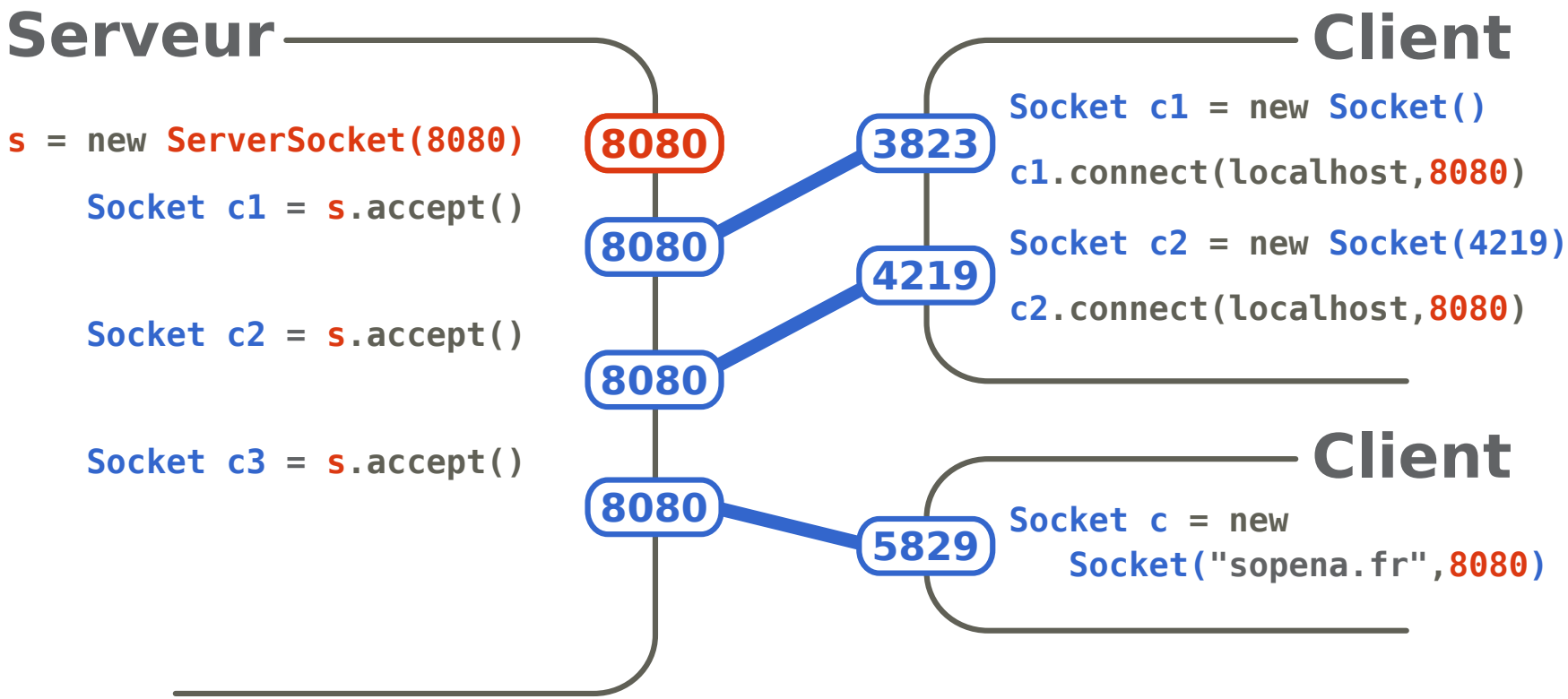
Connexions multiples sur une socket TCP



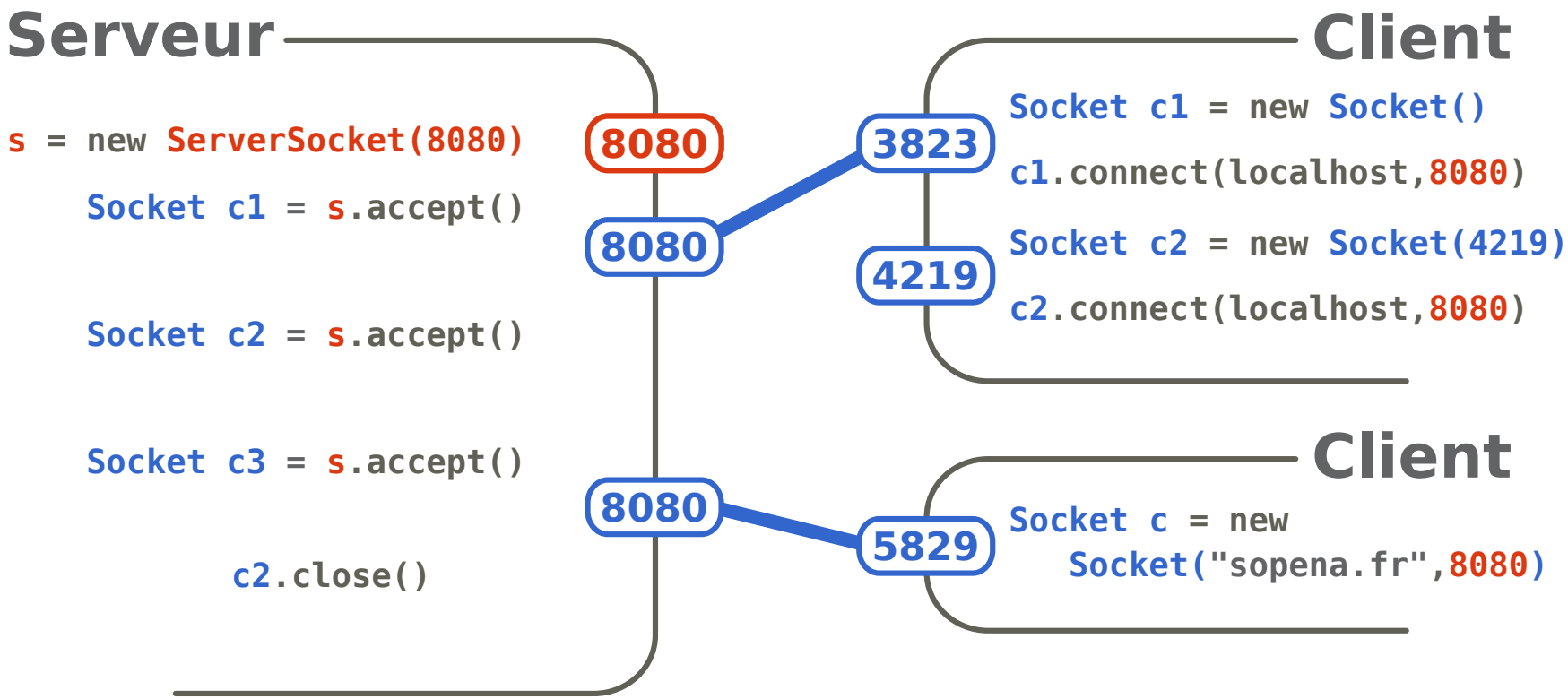
Connexions multiples sur une socket TCP



Connexions multiples sur une socket TCP

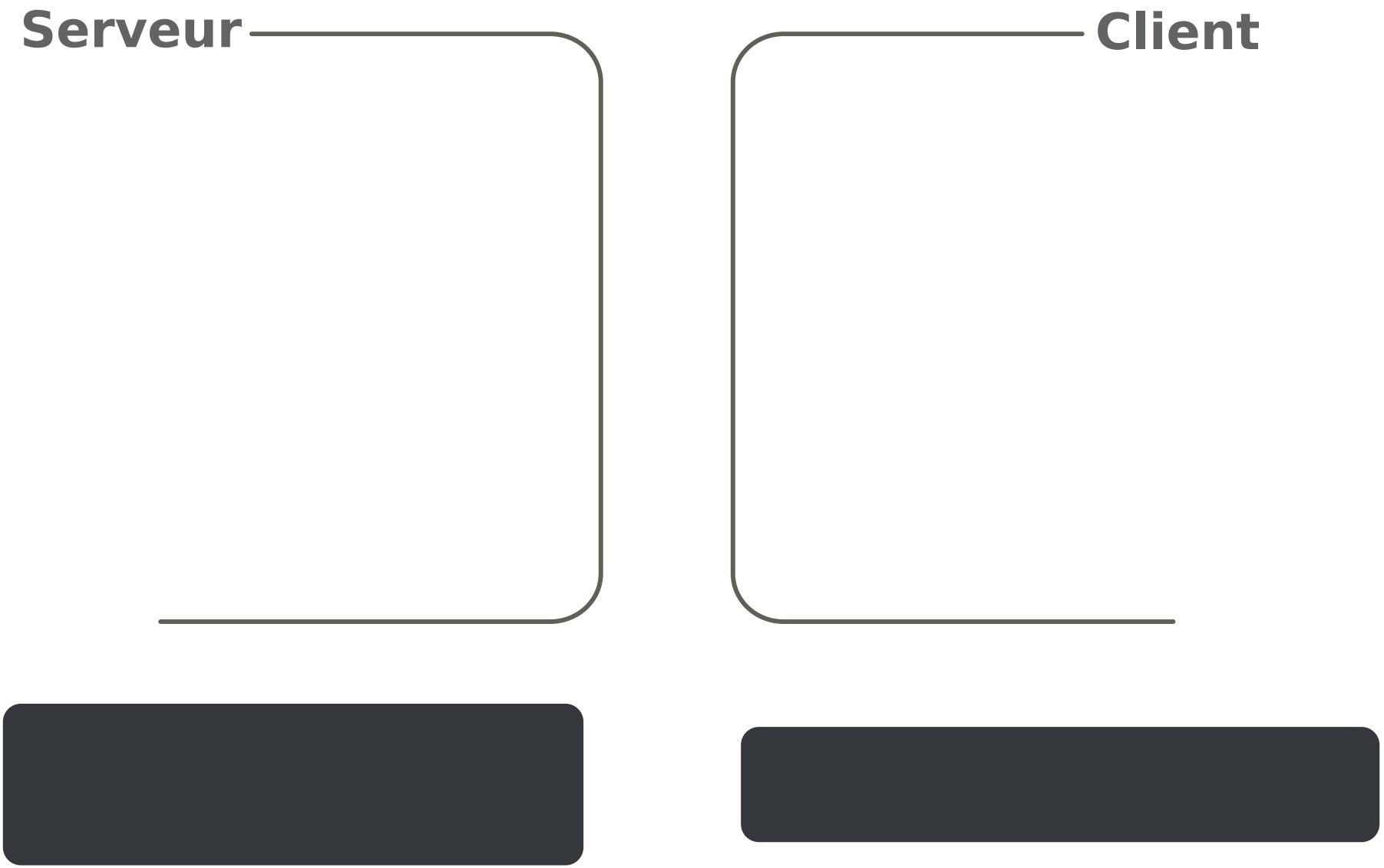


Connexions multiples sur une socket TCP

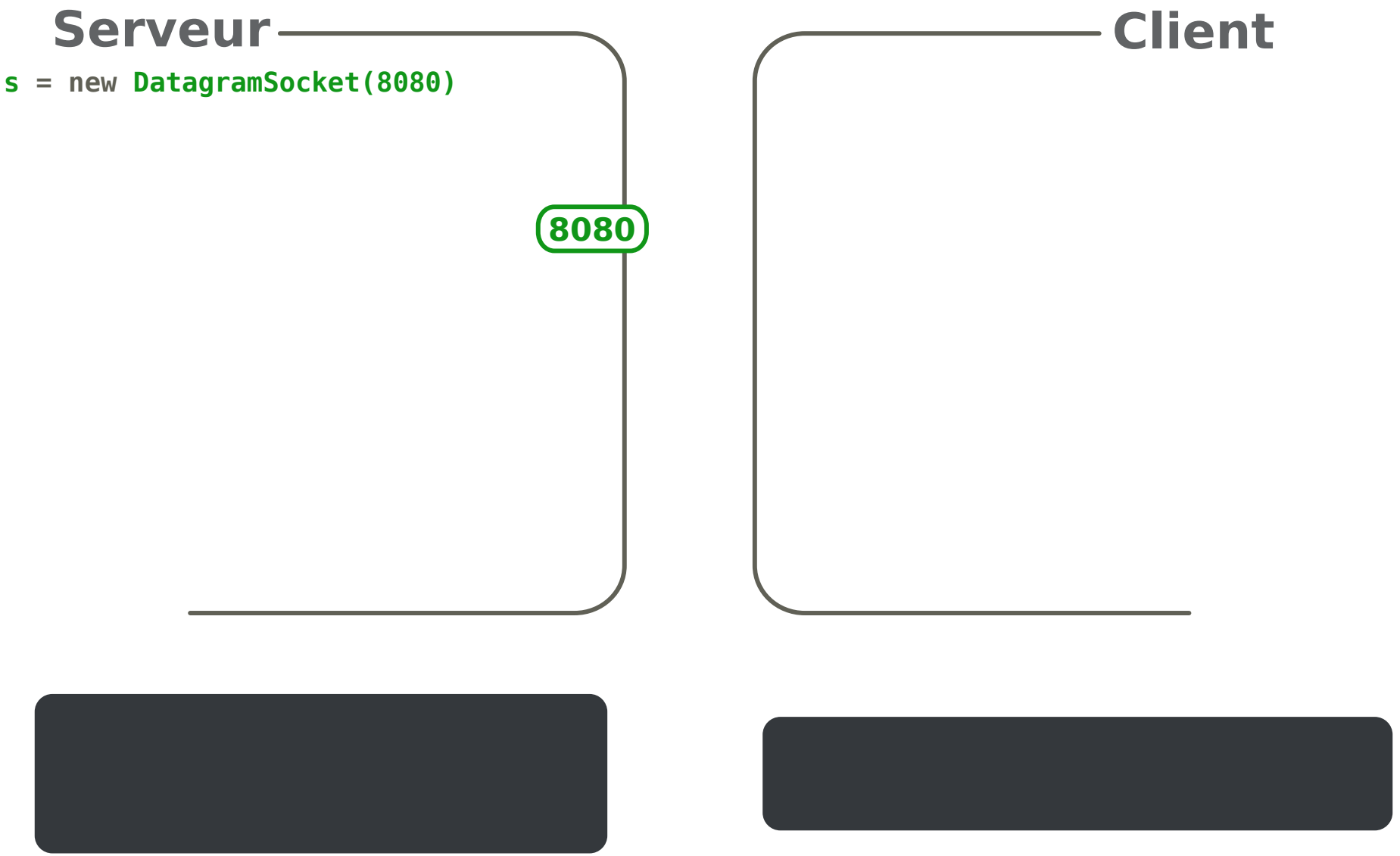


Outline

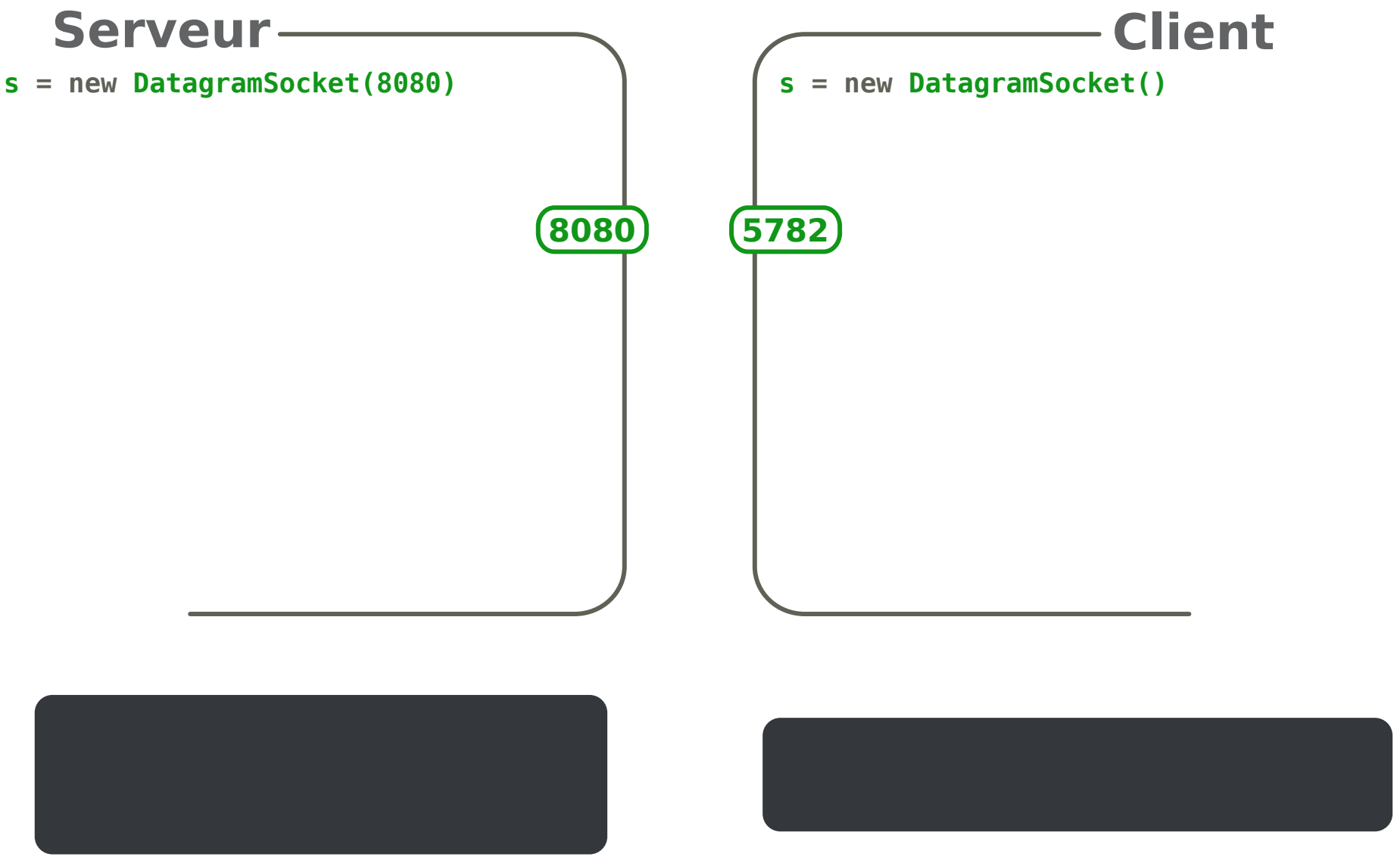
Utilisation d'une socket UDP



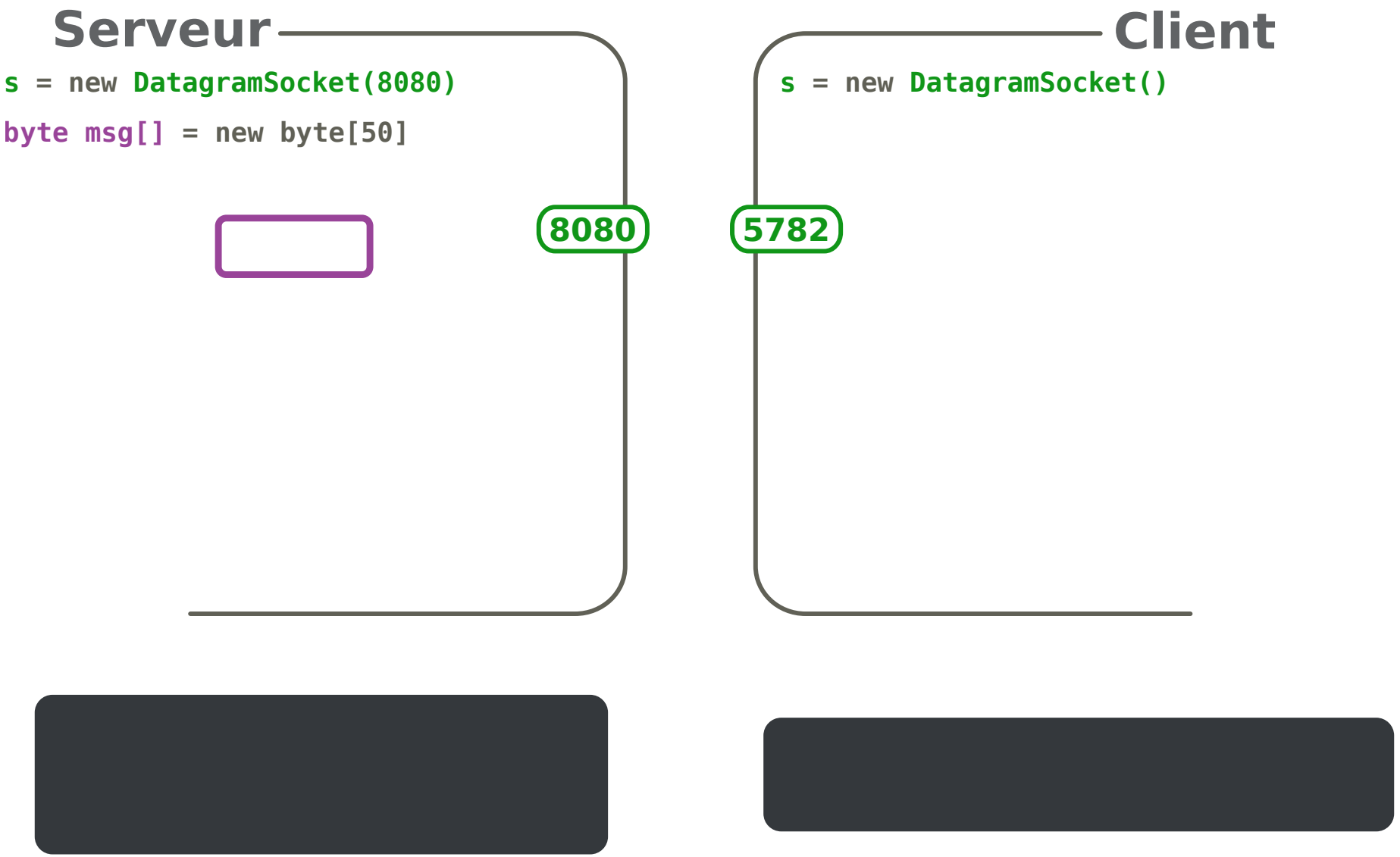
Utilisation d'une socket UDP



Utilisation d'une socket UDP



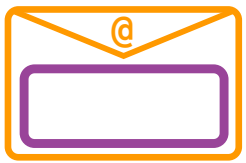
Utilisation d'une socket UDP



Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```




```
DatagramPacket in =
          new DatagramPacket(msg, 50)
```

8080

Client

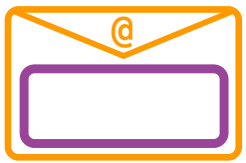
```
s = new DatagramSocket()
```

5782

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```




```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

8080

Client

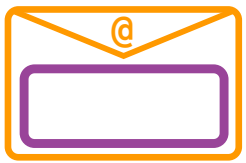
```
s = new DatagramSocket()
```

5782

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```




```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080

5782

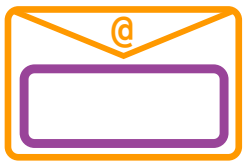
Client

```
s = new DatagramSocket()
```

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



8080

```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```

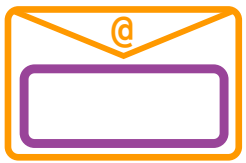
5782

Hello!

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080

5782

Client

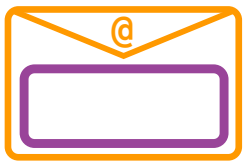
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
```

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```




```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080 ← 5782 ←

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

8080 ← 5782

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
System.out.println(new String(msg))
```

8080

5782

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

```
Hello !
```

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
System.out.println(new String(msg))
System.out.println("From port : " +
                        in.getPort())
```

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

```
Hello !
To port : 5782
```

Utilisation d'une socket UDP

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
System.out.println(new String(msg))
System.out.println("From port : " +
                        in.getPort())
```

Client

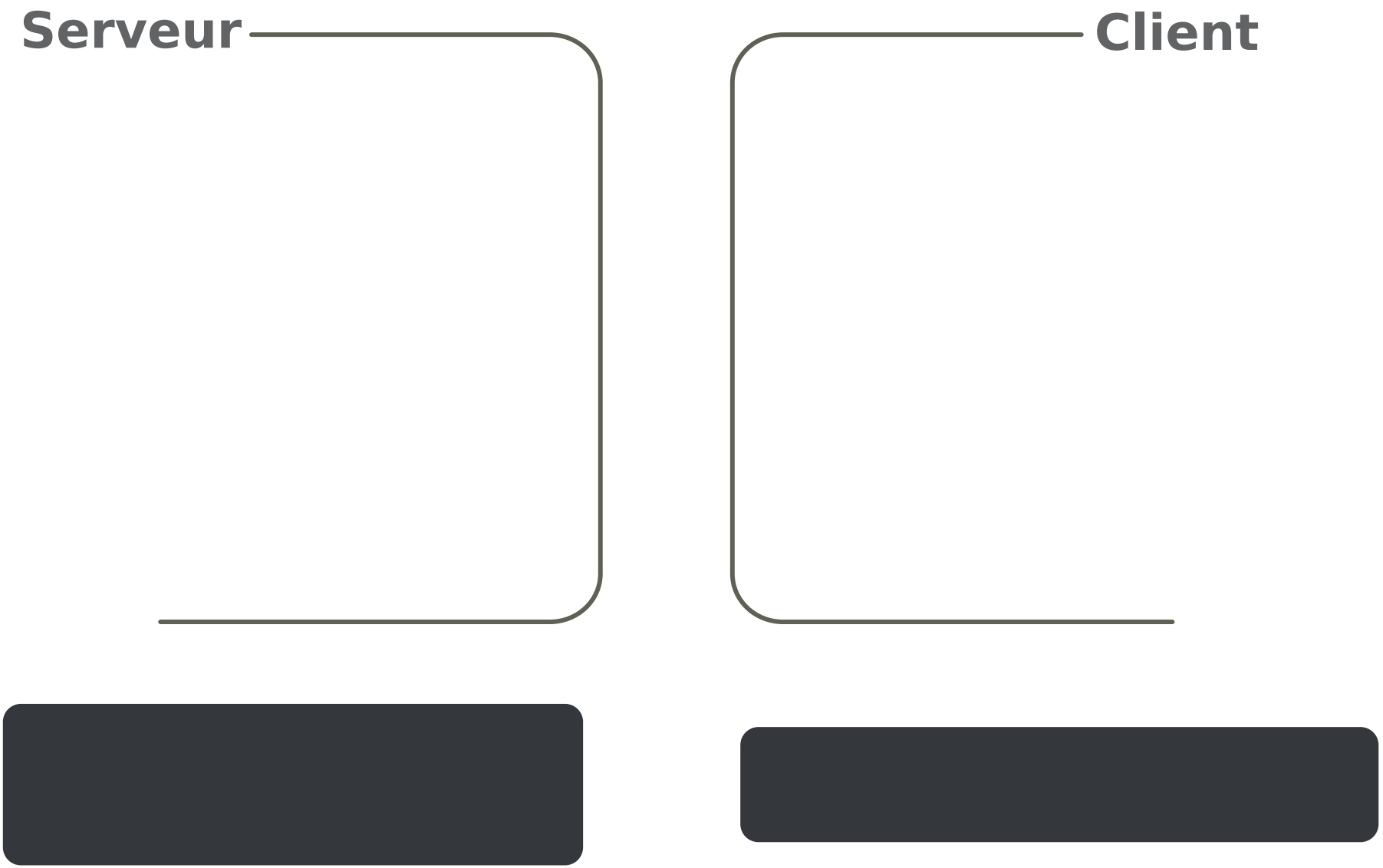
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                        out.getPort())
```

```
Hello !
To port : 5782
```

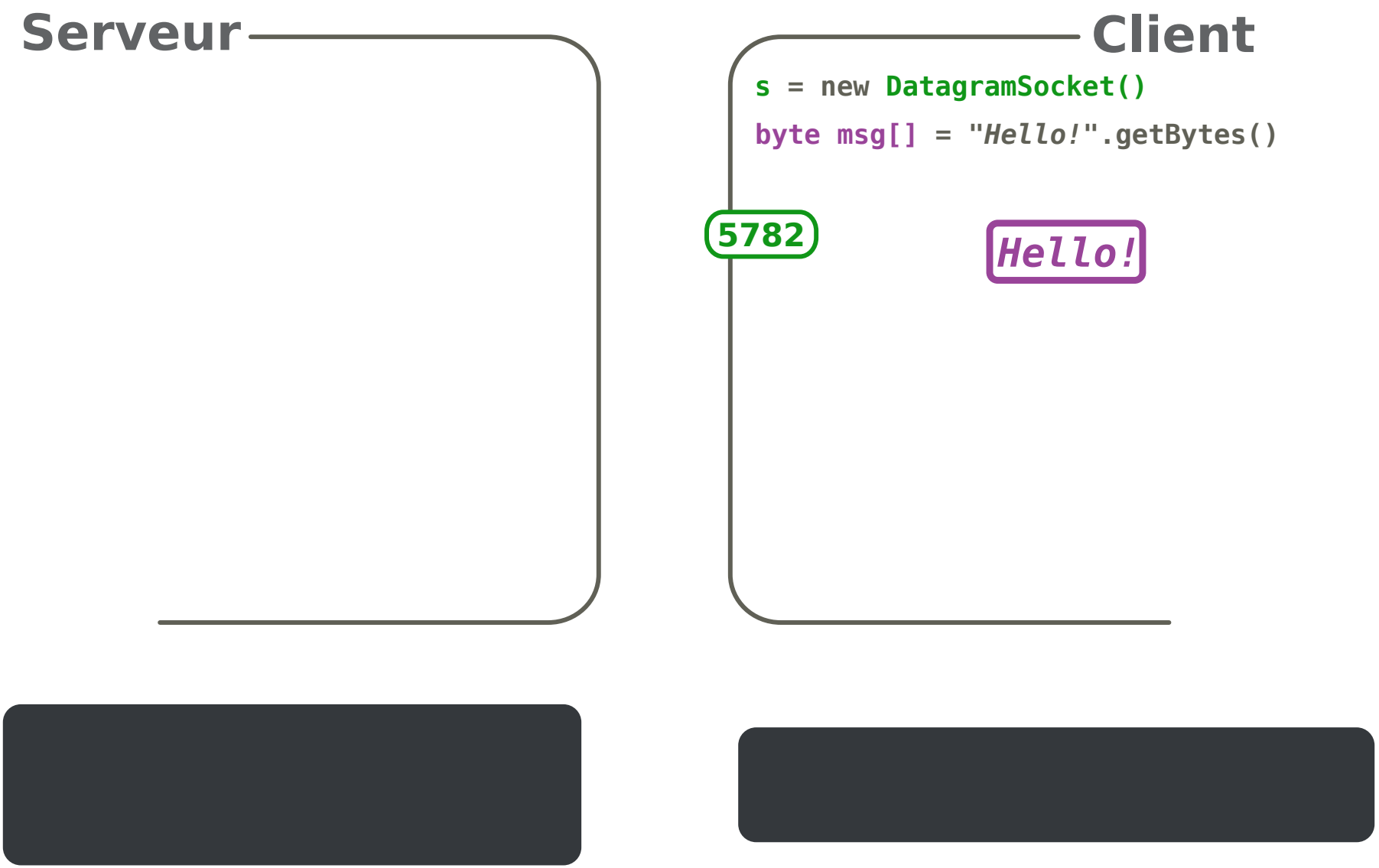
```
To port : 8080
```

Émission précoce : avant ouverture de l'autre socket



Émission précoce : avant ouverture de l'autre socket

Émission précoce : avant ouverture de l'autre socket



Émission précoce : avant ouverture de l'autre socket

Serveur

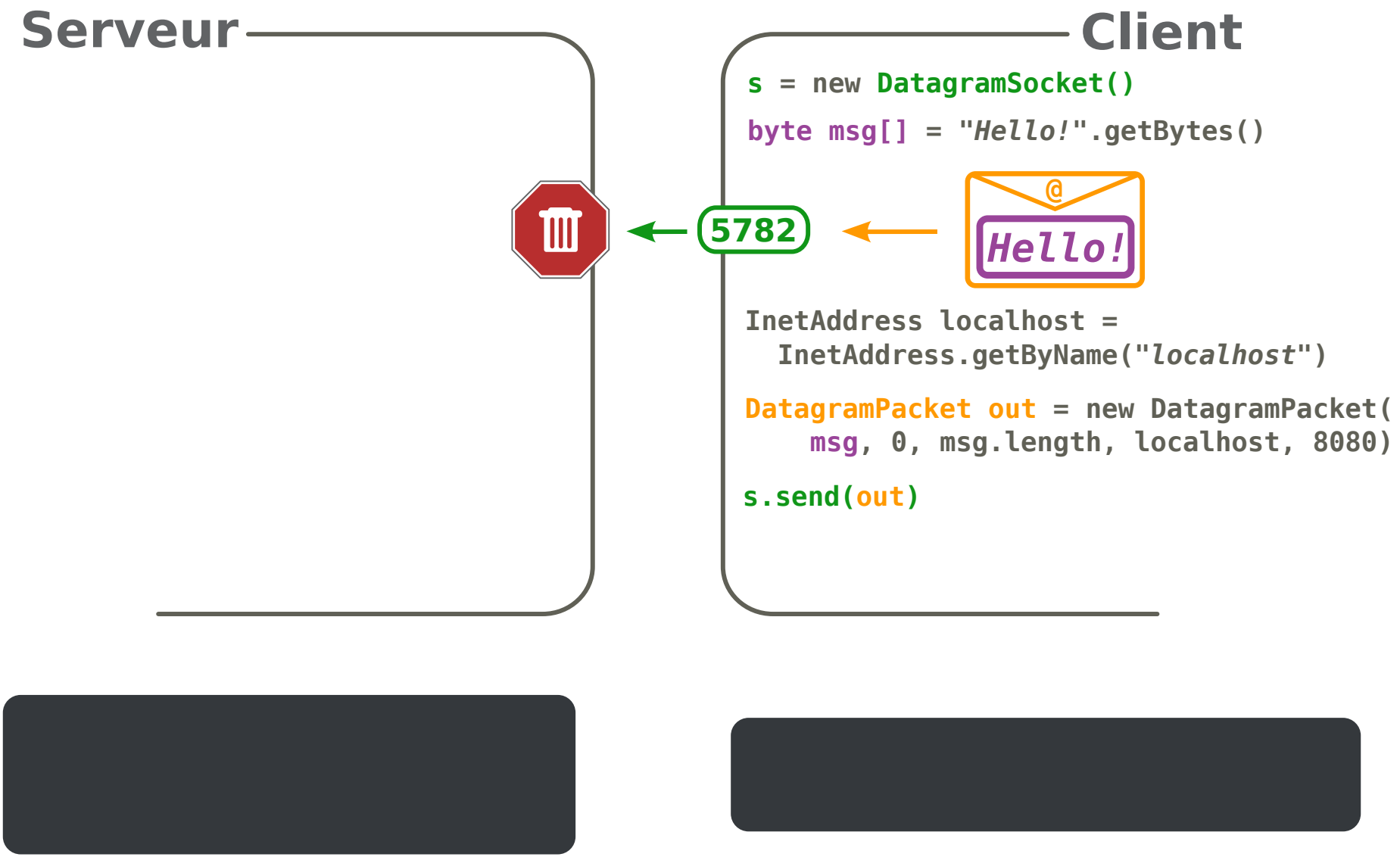
Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```

5782


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
```

Émission précoce : avant ouverture de l'autre socket



Émission précoce : avant ouverture de l'autre socket

Serveur

5782

Client

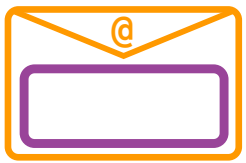
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()

InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                   out.getPort())
```

```
To port : 8080
```

Émission précoce : avant ouverture de l'autre socket

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080

5782

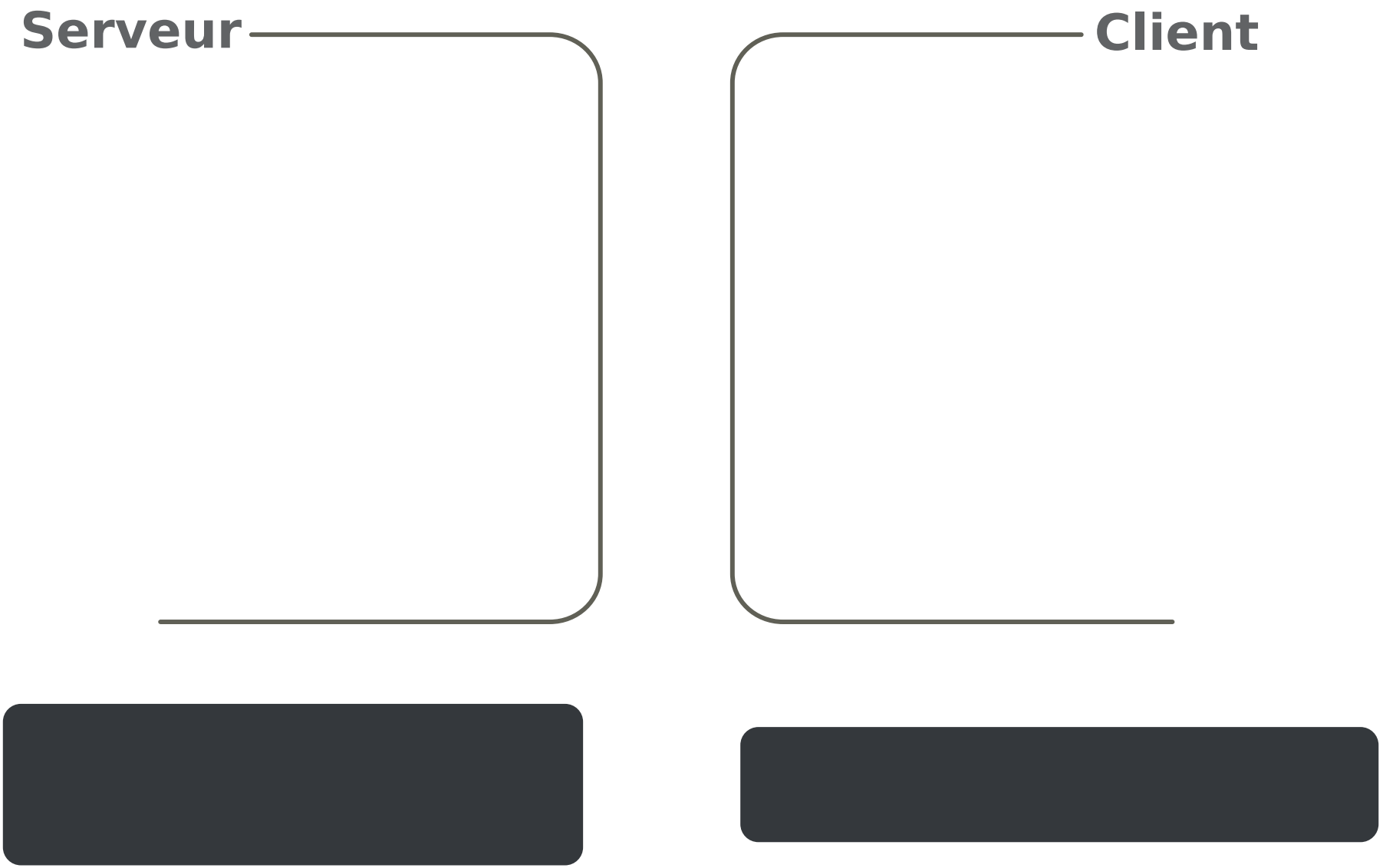
Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                   out.getPort())
```

```
To port : 8080
```

Émission précoce : avant le `receive` du serveur



Émission précoce : avant le `receive` du serveur

Serveur

Client

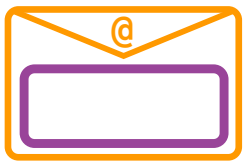
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
```

Émission précoce : avant le `receive` du serveur

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
```

8080

5782

Client

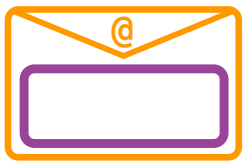
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
```

Émission précoce : avant le `receive` du serveur

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```




```
DatagramPacket in =
          new DatagramPacket(msg, 50)
```

8080 ← 5782 ←

Client

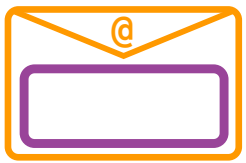
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

Émission précoce : avant le `receive` du serveur

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
```

8080 ← 5782 ←

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                        out.getPort())
```

```
To port : 8080
```

Émission précoce : avant le `receive` du serveur

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```

8080 ← 5782


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                        out.getPort())
```

```
To port : 8080
```

Émission précoce : avant le `receive` du serveur

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```

8080 ← 5782

```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
System.out.println(new String(msg))
System.out.println("From port : " +
                        in.getPort())
```

Client

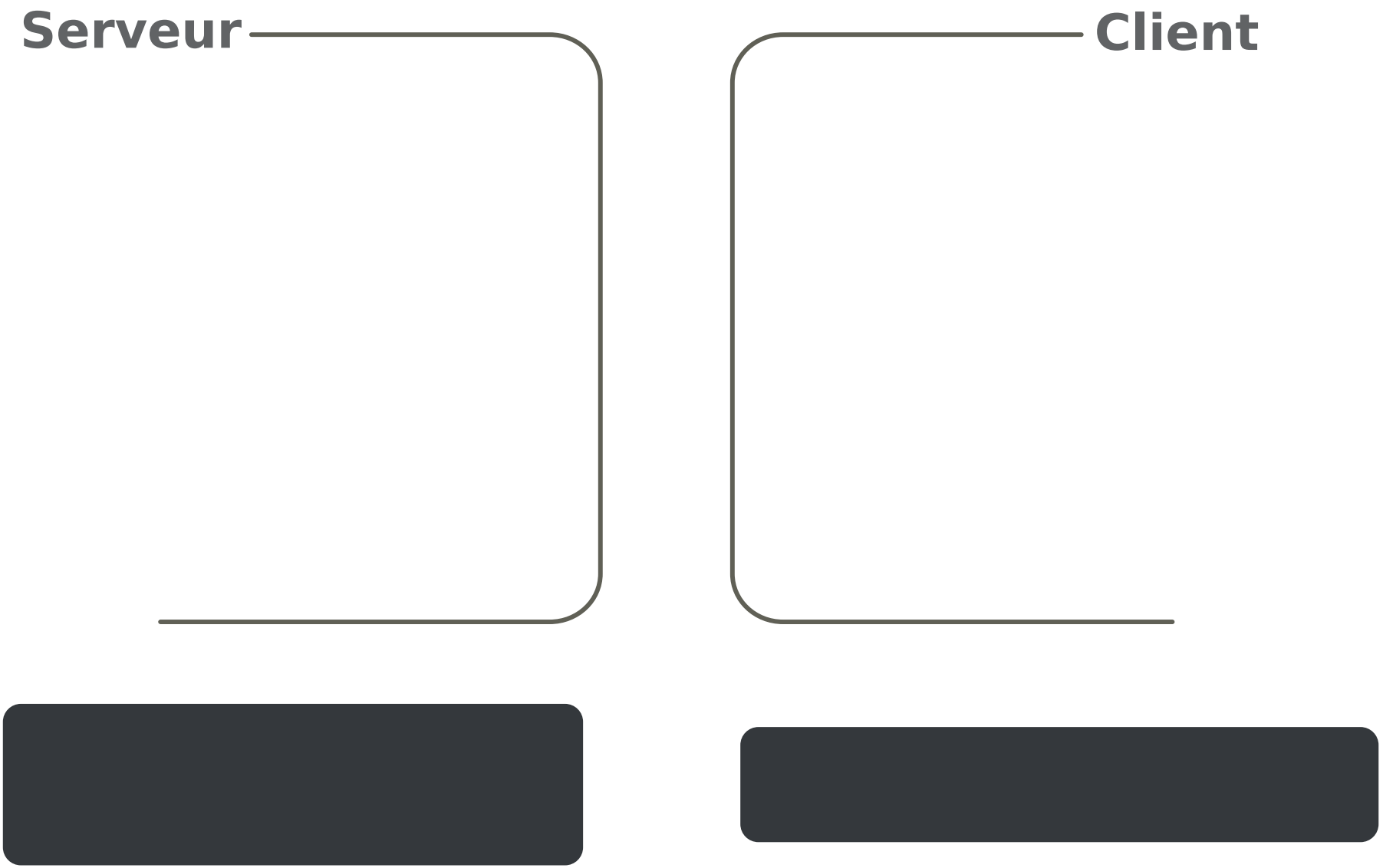
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                        out.getPort())
```

```
Hello !
To port : 5782
```

```
To port : 8080
```

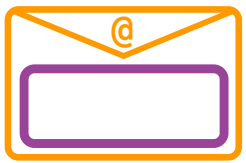
Buffer de réception trop petit



Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[50]
```



8080

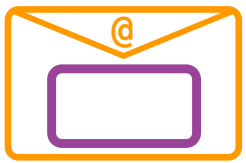
```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

Client

Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[4]
```



8080

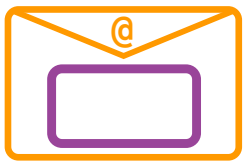
```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

Client

Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[4]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080

5782

Client

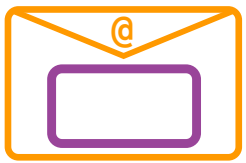
```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
```

Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[4]
```



```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

BLOQUANT

8080 ← 5782 ←

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[4]
```

8080 ← 5782 ←

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
```

```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
```

Buffer de réception trop petit

Serveur

```
s = new DatagramSocket(8080)
byte msg[] = new byte[4]
```


```
DatagramPacket in =
          new DatagramPacket(msg, 50)
s.receive(in)
System.out.println(new String(msg))
System.out.println("From port : " +
                        in.getPort())
```

8080 ← 5782

Client

```
s = new DatagramSocket()
byte msg[] = "Hello!".getBytes()
```


```
InetAddress localhost =
  InetAddress.getByName("localhost")
DatagramPacket out = new DatagramPacket(
    msg, 0, msg.length, localhost, 8080)
s.send(out)
System.out.println("To port : " +
                        out.getPort())
```

```
Hell
To port : 5782
```

```
To port : 8080
```