

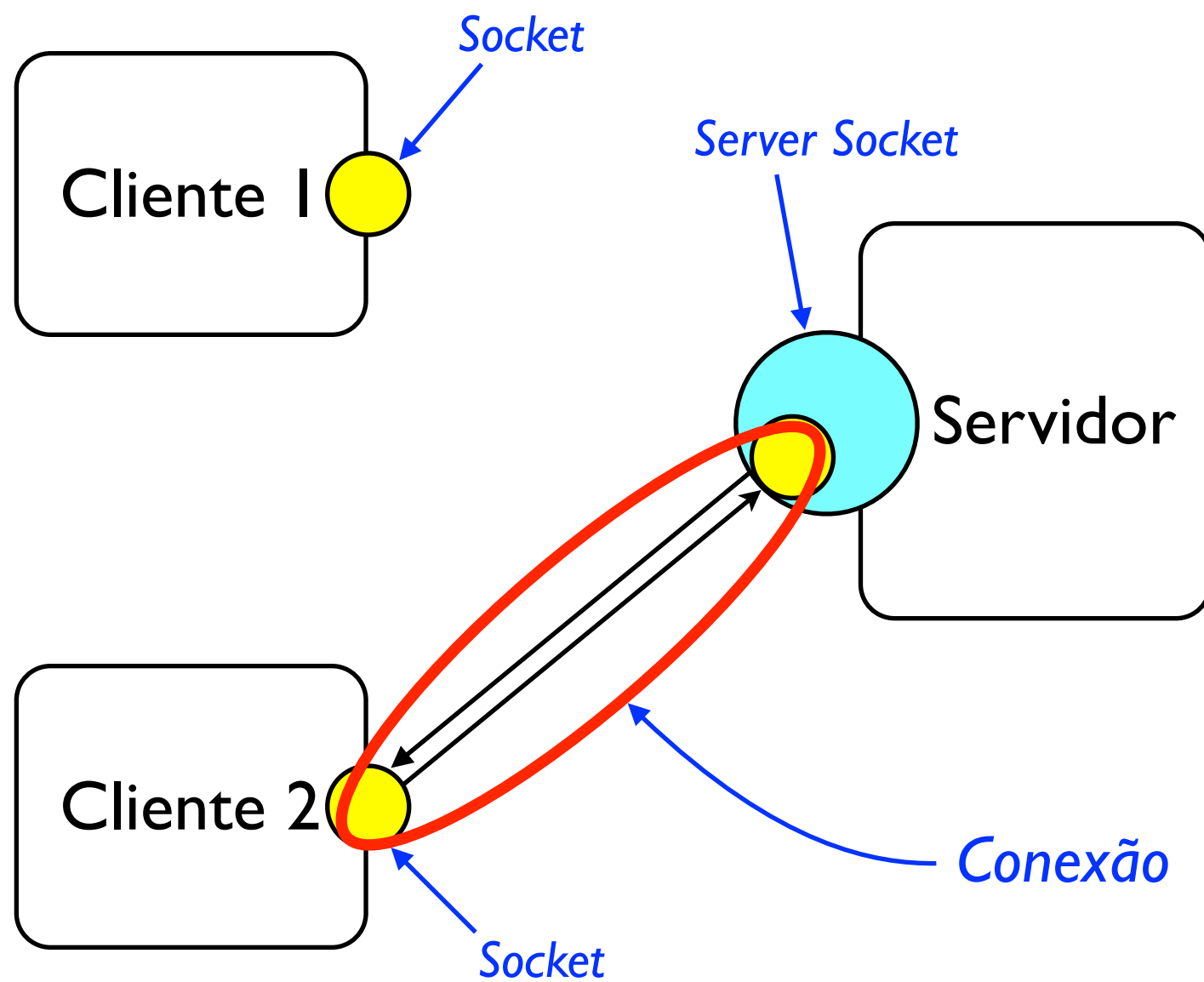
Cliente-Servidor com Sockets TCP

Sistemas Distribuídos

Paradigma cliente - servidor

- **Servidor** oferece um serviço ao cliente (e.g. servidor web envia html da página pedida, servidor de mail entrega as novos emails, ...)
- **Cliente** efectua pedidos ao servidor e trata as respostas (e.g. cliente do browser mostra a página web recebida do servidor, ...)
 - tipicamente, o cliente é que inicia o contacto com o servidor
- Comunicação deve de ser fiável (sem perda dados e com entrega ordenada de mensagens) → **canais TCP**

Paradigma cliente - servidor



- Servidor fica à espera de ligações num determinado porto;
- Quando o cliente se liga ao servidor é estabelecida uma nova conexão bidireccional;
- Socket representa um extremo de uma conexão.

- Classes e métodos relevantes em JAVA:
- (servidor) **java.net.ServerSocket**:
 - construtor: **ServerSocket(int port)**
 - outros métodos: **accept()**, **close()**
- (cliente) **java.net.Socket**:
 - construtor: **Socket(String host, int port)**
 - outros métodos: **connect()**, **getInputStream()**, **getOutputStream()**
- ler / escrever no socket: **BufferedReader**, **InputStreamReader** / **BufferedWriter**, **OutputStreamWriter**

Cliente

Esqueleto:

criar socket
e ligação com
o servidor

```
Socket socket = new Socket(address, port);
```

abrir canais
de escrita e
leitura no
socket

```
BufferedReader in = new BufferedReader(new  
InputStreamReader(socket.getInputStream()));
```

```
BufferedWriter out = new BufferedWriter(new  
OutputStreamWriter(socket.getOutputStream()));
```

ler e escrever
nos canais de
acordo com o
protocolo da
aplicação

```
while(...) {  
    out.write(...);  
    out.newLine();  
    out.flush();  
    ... = in.readLine();  
}
```

fechar socket
e respectivos
canais

```
socket.shutdownOutput();  
socket.shutdownInput();  
socket.close();
```

JAVA:

Servidor

Esqueleto:

criar novo
server socket
num dado porto

```
ServerSocket sSock = new ServerSocket(port);
```

aceitar conexões
indefinidamente

```
while (true)  
{
```

bloquear até que
uma conexão
seja estabelecida

```
Socket clSock = sSock.accept();
```

abrir canais
de escrita e
leitura no socket

```
BufferedReader in = new BufferedReader(new  
InputStreamReader(clSock.getInputStream()));
```

```
BufferedWriter out = new BufferedWriter(new  
OutputStreamWriter(clSock.getOutputStream()));
```

ler e escrever nos
canais de acordo
com o protocolo
da aplicação

```
while(...) {  
    ... = in.readLine();  
    out.write(...);  
    out.newLine();  
    out.flush();  
}
```

fechar socket
e respectivos
canais

```
clSock.shutdownOutput();  
clSock.shutdownInput();  
clSock.close();  
}
```

JAVA:

Exercícios

- 1) Implemente um servidor que aceite a ligação de um cliente de cada vez, e que devolva ao cliente cada linha de texto que este lhe envie até o cliente fechar a ligação.
- 2) Implemente um cliente para o servidor desenvolvido no exercício anterior.

- **Sugestão:**

- Usar endereço "127.0.0.1" e porto 12345 na criação dos sockets
- Fechar ligação quando se escrever "quit" no cliente

Exercícios

- 3) Implemente um servidor para efectuar somas, tratando um cliente de cada vez. O cliente poderá enviar números inteiros, na sua representação textual, um por linha. O servidor, deverá responder a cada número recebido, enviando a soma de todos os números até então recebidos desse cliente. Quando deixarem de chegar números (*end of file* no stream de leitura) o servidor deverá enviar ao cliente a média de todos os números enviados pelo cliente.
- 4) Implemente um cliente para o servidor de soma desenvolvido no exercício anterior.

● Sugestão:

- Usar endereço "127.0.0.1" e porto 12345 na criação dos sockets
- Fechar ligação quando se escrever "quit" no cliente

Exercícios

5) Modifique cada um dos servidores acima de modo a permitir tratar vários clientes concorrentemente, dedicando uma thread a cada cliente.

