Exercise 3

Distributed Software Systems — Prof. Paolo Ciancarini
Università di Bologna
A.Y. 2023/2024
Riccardo Scotti (0001060133)

Greeting server

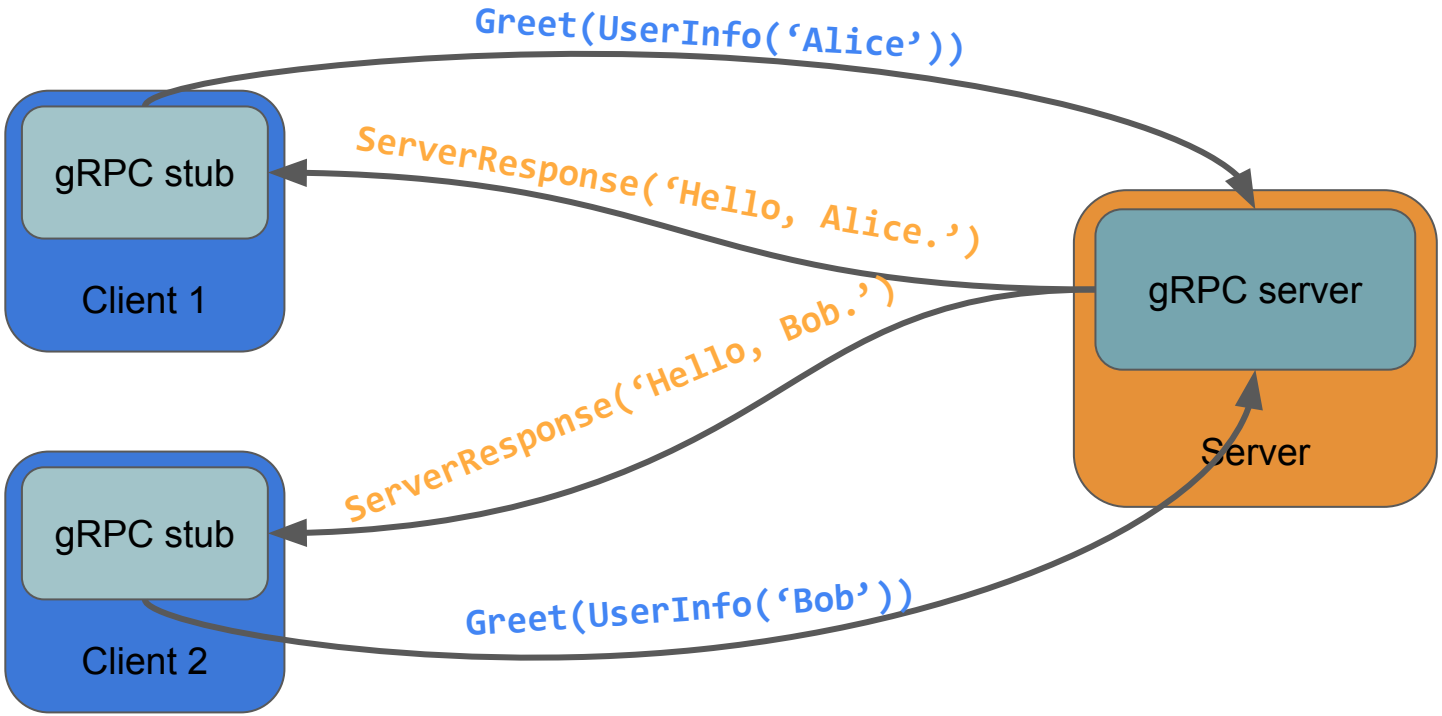
- We want to design a gRPC server that can receive a request from a client, containing its name, and respond to it with a personalized greeting.
- When launching the server, it will be possible to specify the port it will listen for requests.
- When launching the clients, it is possible to specify both their name and the port they will send requests to.
- The system will be tested with two clients, running on different threads.

Greeting server: protocol

In order to allow the communication between server and clients, we will define a gRPC protocol in a .proto file, which will then be compiled to a python API that will be used to perform Remote Procedure Calls.

In the protocol, we define message types **`UserInfo`**, containing a string with the name of the client, and **`ServerResponse`**, containing the greeting phrase. We also define the procedure **`Greet`**, which takes a **`UserInfo`** and returns a **`ServerResponse`**.

Greeting server



Greeting server

```
riccardo@riccardo:~/uni/distributed/lab3$ python3 server.py 55550
Server started, listening on 55550
[Server] Received request from user named 'Bob'
[Server] Received request from user named 'Alice'
```

```
riccardo@riccardo:~/uni/distributed/lab3$ python3 client.py 55550
Client 1 started
Client 2 started
[Bob] Message received from server: "Hello, Bob."
[Alice] Message received from server: "Hello, Alice."
riccardo@riccardo:~/uni/distributed/lab3$
```

Challenge: add functions to clients to make them chat p2p

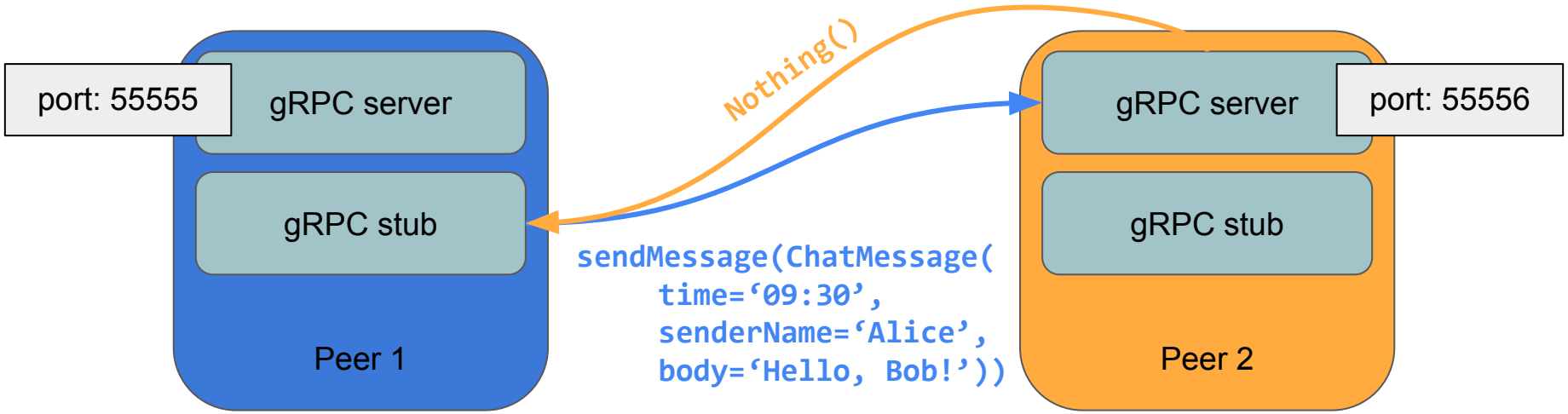
- a peer-to-peer chat can be seen as two processes acting both as a server and as a client at the same time
  - they listen for incoming requests (messages), like servers
  - but they can also send requests to other servers, like clients
- two new function to the clients defined before (from now on referred to as “peers”), one to listen for messages and the other to send them
- each peer listens to a unique port and sends messages on the other peer’s port

Peer to peer chat: protocol

In a new .proto file, we define message types **`ChatMessage`** and **`Nothing`**; the former contains a timestamp, the name of the sender and the body of the message; the latter is an empty message, used because rpc functions necessarily need to take and returns data, even when it is not required.

We also define two procedures: **SendMessage** and **GetInfo**; the former allows a peer to send a **ChatMessage** to another peer, while the other is used to establish connection between peers at the beginning.

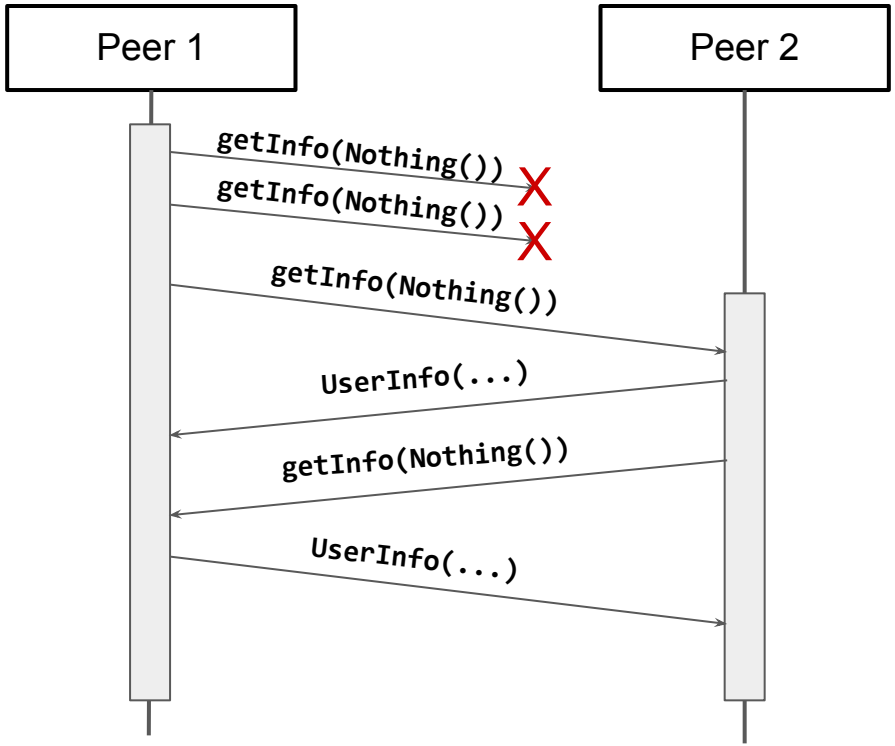
Peer to peer chat: protocol



Peer to peer chat: establishing connection

- A peer must know which port the other peer is listening on, and vice versa (space coupling)
- If the other peer is not running, all messages sent are lost (time coupling)
- When a peer starts running, it tries to call the **GetInfo** procedure;
  - if the other peer is not yet on, as per the gRPC API an exception is raised
  - as long as the exception is raised, the peer retries every n seconds
  - when it receives a response from the other peer, it is possible to start exchanging messages

Peer to peer chat: establishing connection



Peer to peer chat

```
riccardo@riccardo:~/uni/distributed/lab3$ python3 chat.py Alice
55555 55556
Waiting for other peer to connect...


Welcome, Alice. You are chatting with Bob.


[17:24] Alice> Hello, Bob!
[17:24] Bob> Hello, Alice!
```

```
riccardo@riccardo:~/uni/distributed/lab3$ python3 chat.py Bob 5
5556 55555
Waiting for other peer to connect...


Welcome, Bob. You are chatting with Alice.


[17:24] Alice> Hello, Bob!
[17:24] Bob> Hello, Alice!
```

Possible improvements

- Space coupling could be avoided, while still technically maintaining a direct, peer-to-peer communication, by using a naming server that peers would query only before starting a communication, and then proceeding again to send messages directly.
- Time coupling cannot be avoided, if we want that messages flow from one peer to another, without being stored somewhere else.
- The busy-wait a peer performs when waiting the other to connect could be avoided by using some sort of interruption or signal.