Java TCP and UDP programming

Link to the course

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Based on the original course by O. Liechti and J. Ehrensberger.

Objectives (1/2)

- Program your own TCP client/server applications in Java with the Socket API.
- Process data from TCP streams.
- Program your own UDP emitter/receiver applications in Java with the Socket API.
- Process data from UDP datagrams.

Objectives (2/2)

- Learn the different ways to send a UDP datagram to one or multiple clients.
- How UDP can be used for service discovery.
- Make usage of a REPL.

Your applications will be able to communicate over the network!

Part 1/2

In this first part, you will learn the basics of TCP and UDP programming in Java.

Explore the code examples

More details for this section in the course material. You can find other resources and alternatives as well.

Explore the code examples

Individually, or in pair/group, **take 15 minutes to explore and discuss the code examples (part 1/2)**.

Answer the questions available in the course material:

- How do the code examples work?
- What are the main takeaways of the code examples?
- What are the main differences between the code examples?

If needed, use the theoretical content to help you.

The Socket API

More details for this section in the course material. You can find other resources and alternatives as well.

The Socket API

- Java API to create TCP and UDP network applications.
- Described in the `java.net` package in the `java.base` module.
- Originally developed by Berkeley University for the Unix operating system.
- Ported to Java and many other languages.
- Provides classes and methods to create sockets, connect to remote hosts, send and receive data, etc.

TCP

More details for this section in the course material. You can find other resources and alternatives as well.

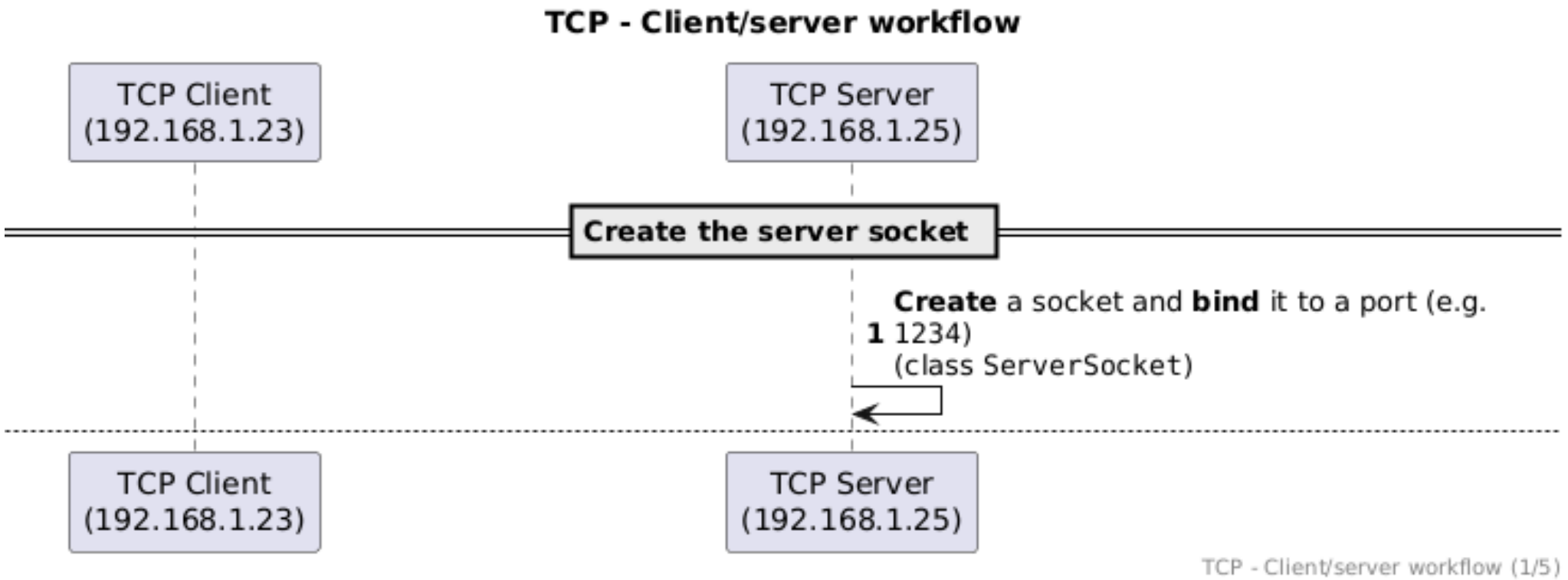
TCP

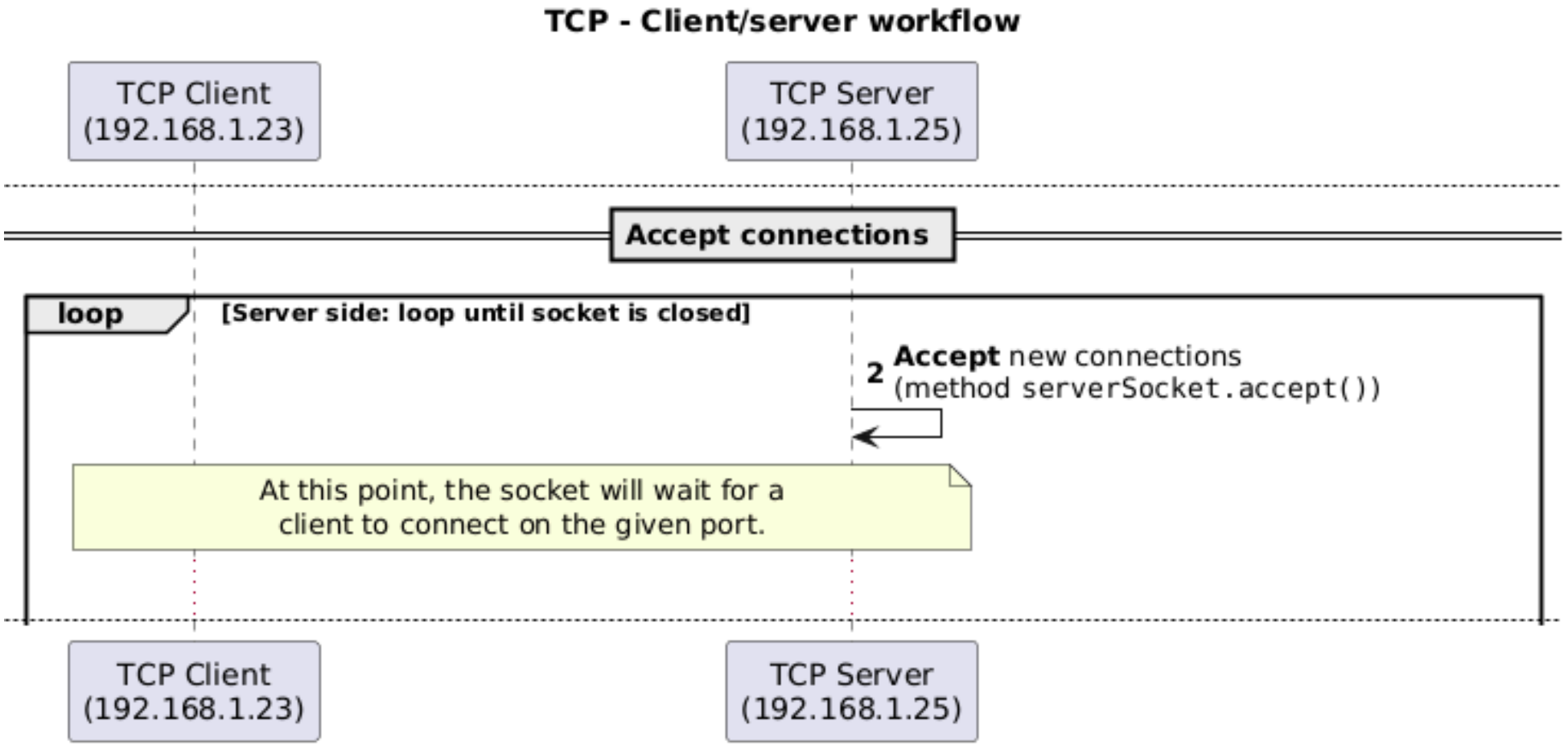
TCP is a transport protocol that is similar to a phone call:

1. A connection is established between two parties (unicast).
2. Data sent is guaranteed to arrive in the same order.
3. Data can be sent again if needed (retransmission).

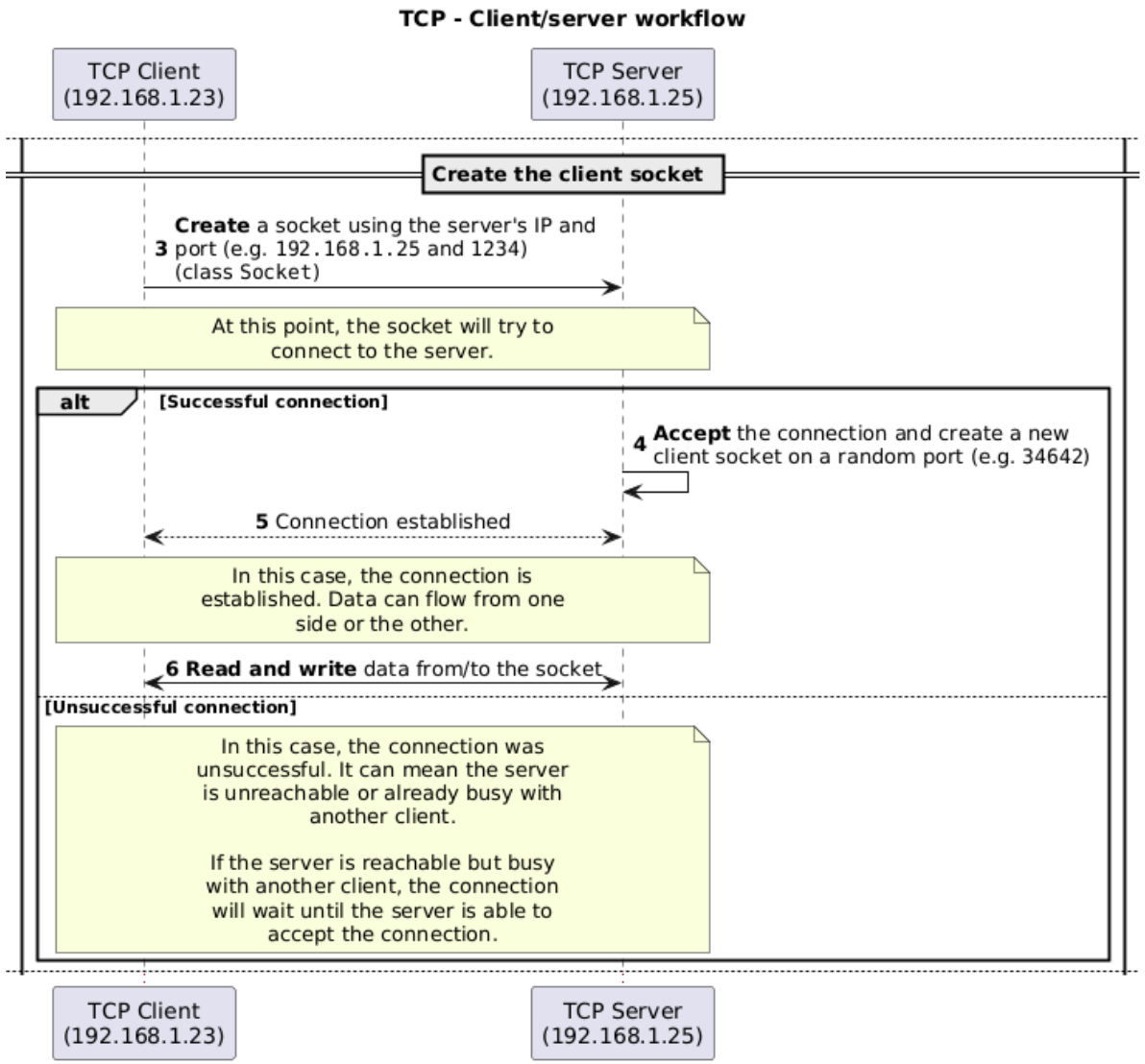
TCP in the Socket API

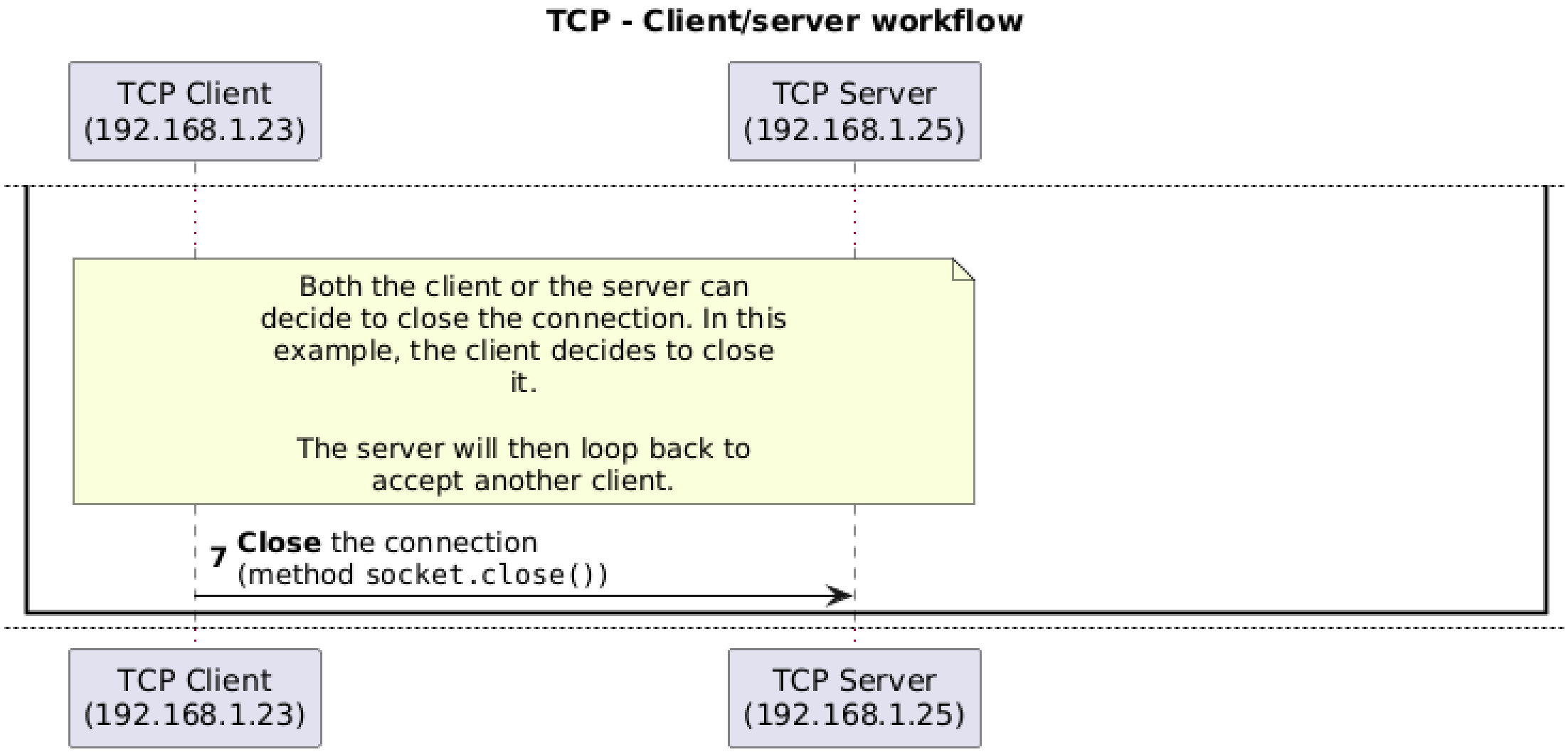
- The Socket API provides two classes to create TCP applications:
  - `ServerSocket` (server side).
  - `Socket` (client side).
- A TCP socket is a connection endpoint between two parties using a host and a port.
- A server socket listens for incoming connections on a specific port.
- A client socket connects to a server socket using the server's host and port.
- Once connected, data can be exchanged between the two sockets.


TCP - Client/server workflow
TCP Client
(192.168.1.23)
TCP Server
(192.168.1.25)
Create the server socket
Create a socket and bind it to a port (e.g.
1 1234)
(class ServerSocket)
TCP Client
(192.168.1.23)
TCP Server
(192.168.1.25)
TCP - Client/server workflow (1/5)

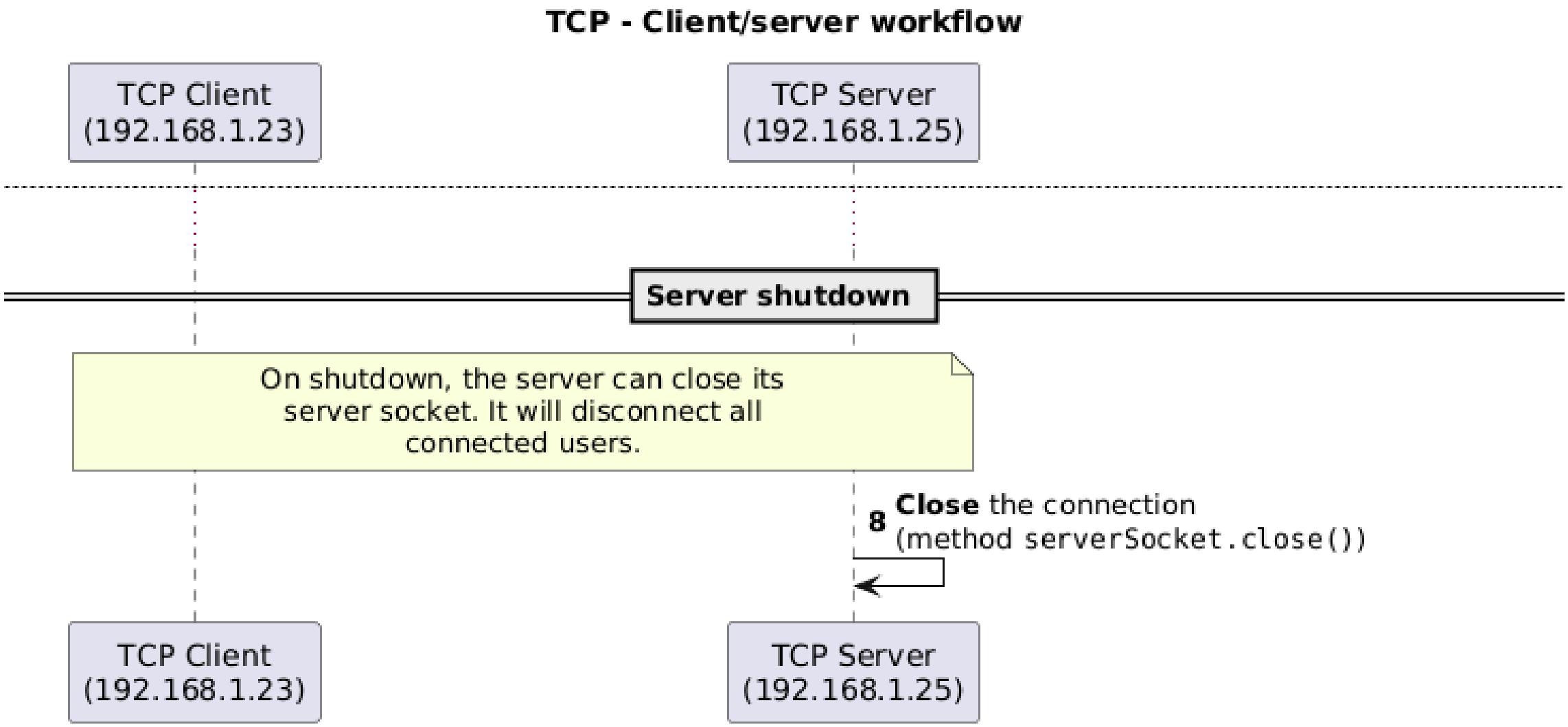


TCP - Client/server workflow (2/5)


TCP - Client/server workflow
TCP Client
(192.168.1.23)
TCP Server
(192.168.1.25)
Create the client socket
Create a socket using the server's IP and
3 port (e.g. 192.168.1.25 and 1234)
(class Socket)
At this point, the socket will try to
connect to the server.
alt
[Successful connection]
4 Accept the connection and create a new
client socket on a random port (e.g. 34642)
5 Connection established
In this case, the connection is
established. Data can flow from one
side or the other.
6 Read and write data from/to the socket
[Unsuccessful connection]
In this case, the connection was
unsuccessful. It can mean the server
is unreachable or already busy with
another client.
If the server is reachable but busy
with another client, the connection
will wait until the server is able to
accept the connection.
TCP Client
(192.168.1.23)
TCP Server
(192.168.1.25)
TCP - Client/server workflow (3/5)



TCP - Client/server workflow (4/5)



TCP - Client/server workflow (5/5)

Processing data from streams

- Sockets use data streams to send and receive data, just like files:
  - Get an input stream to read data from a socket.
  - Get an output stream to write data to a socket.

Get the streams from a socket:

```
// Get input stream
input = socket.getInputStream();

// Get output stream
output = socket.getOutputStream();
```

Read and write data from/to the streams as text:

```
// Get input stream as text
input = new InputStreamReader(socket.getInputStream(), StandardCharsets.UTF_8);

// Get output stream as text
output = new OutputStreamWriter(socket.getOutputStream(), StandardCharsets.UTF_8);
```

Improve performance with a buffer (with a binary stream):

```
// Get input stream as binary with buffer
input = new BufferedInputStream(socket.getInputStream());

// Get output stream as binary with buffer
output = new BufferedOutputStream(socket.getOutputStream());
```

Everything you have learned about Java IOs applies here!

UDP

More details for this section in the course material. You can find other resources and alternatives as well.

UDP

- A transport layer protocol just like TCP.
- Connectionless protocol - does not require to establish a connection before sending data.
- Unreliable protocol - does not guarantee delivery but is fast.
- Analogy: sending postcards through the postal service.

UDP datagrams

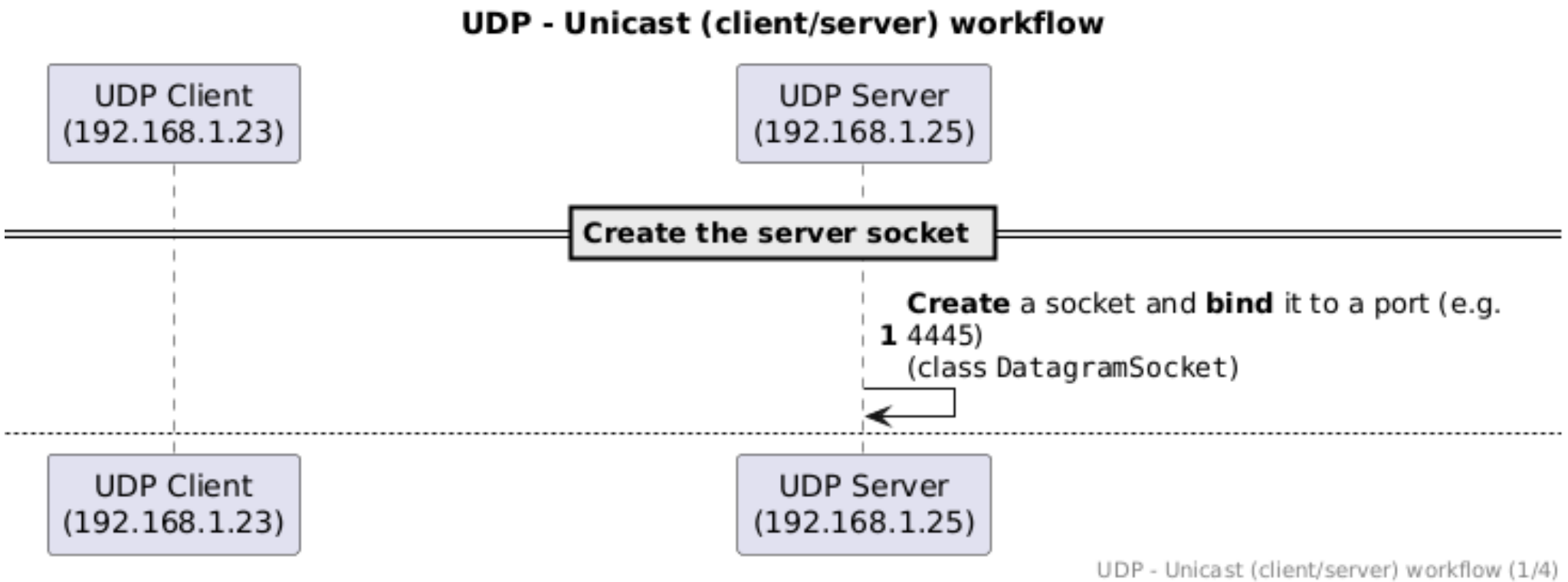
- Datagrams are discrete chunks of data (packets) sent over the network.
- Sent individually and independently, just as postcards.
- Each datagram is self-contained and has all the information needed for routing and delivery, thanks to its header.
- The header contains the source's IP, source and destination ports, length, checksum, etc. and a payload (data).
- Maximum size of a UDP datagram is 65,535 bytes (including header).

Reliability

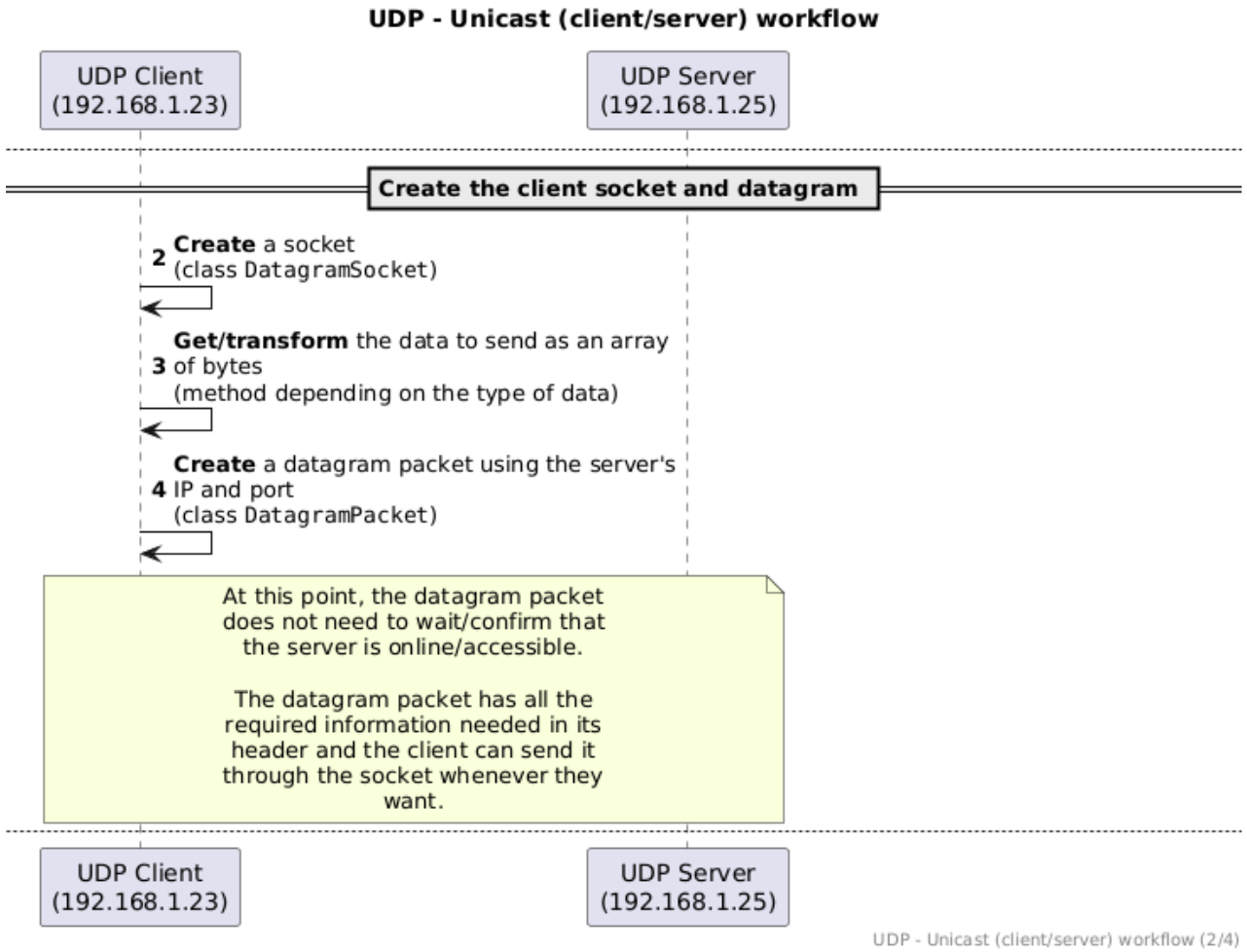
- UDP is unreliable (no guarantee of delivery, no guarantee of order).
- The application must implement its own reliability mechanism.
- In some cases, reliability is not needed (e.g. streaming).
- Handling reliability is complex - not covered in this course.

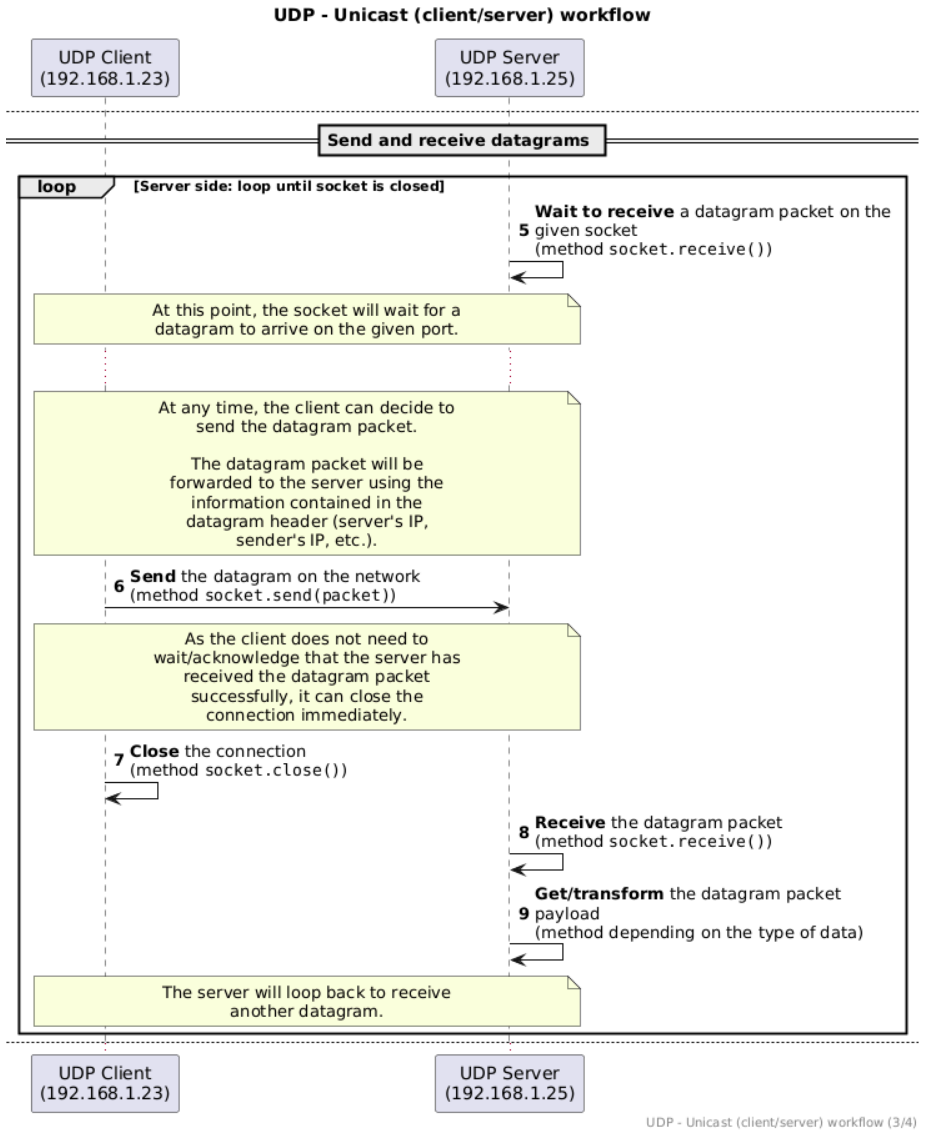
UDP in the Socket API

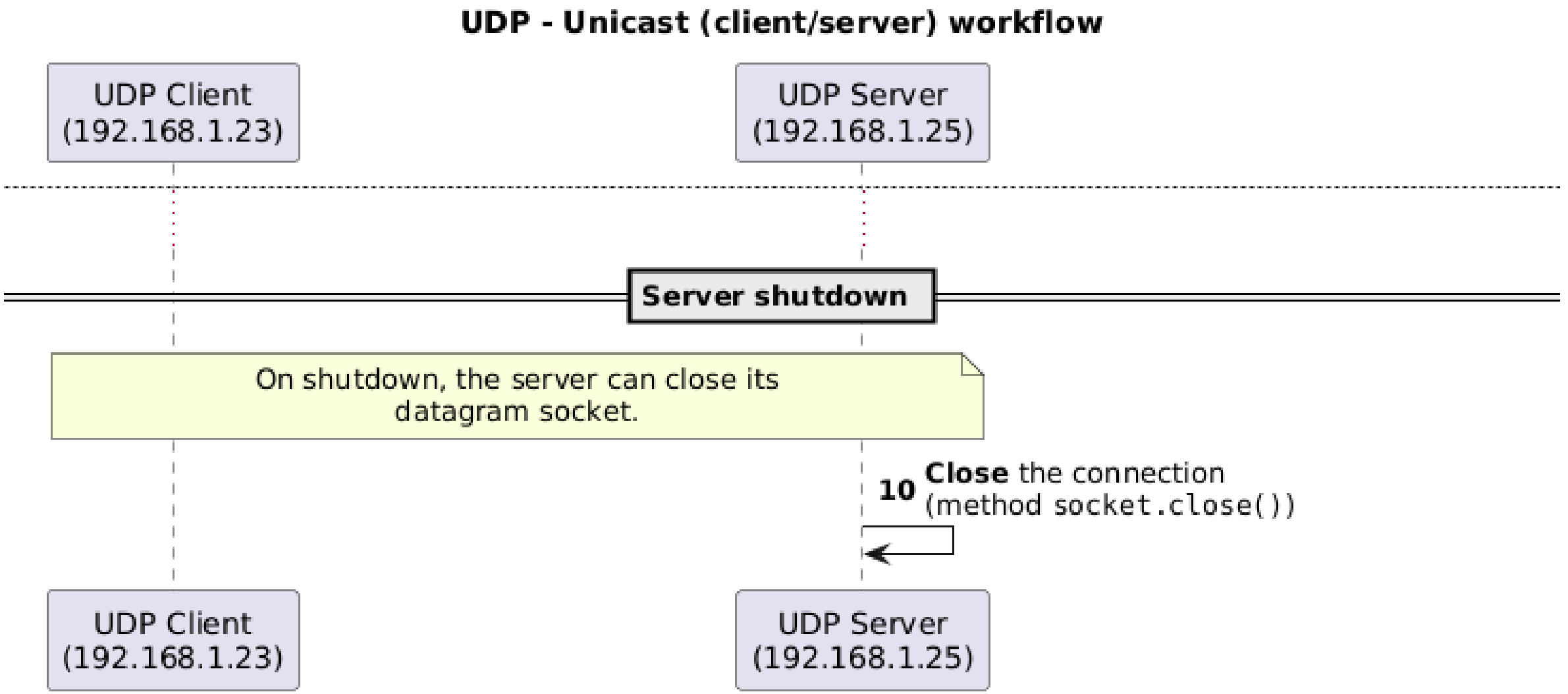
- `DatagramSocket` is used to send and receive datagrams.
- A datagram is created with the `DatagramPacket` class.
- A datagram socket can send and receive datagrams to/from any host and port.
- A datagram socket can be bound to a specific port to listen for incoming datagrams.
- A datagram socket can be used to send and receive datagrams without establishing a connection.



UDP - Unicast (client/server) workflow (1/4)


UDP - Unicast (client/server) workflow
UDP Client
(192.168.1.23)
UDP Server
(192.168.1.25)
Create the client socket and datagram
2 Create a socket
(class DatagramSocket)
3 Get/transform the data to send as an array
of bytes
(method depending on the type of data)
4 Create a datagram packet using the server's
IP and port
(class DatagramPacket)
At this point, the datagram packet
does not need to wait/confirm that
the server is online/accessible.
The datagram packet has all the
required information needed in its
header and the client can send it
through the socket whenever they
want.
UDP Client
(192.168.1.23)
UDP Server
(192.168.1.25)
UDP - Unicast (client/server) workflow (2/4)


UDP - Unicast (client/server) workflow
UDP Client (192.168.1.23)
UDP Server (192.168.1.25)
Send and receive datagrams
loop
[Server side: loop until socket is closed]
5 Wait to receive a datagram packet on the given socket (method socket.receive())
At this point, the socket will wait for a datagram to arrive on the given port.
At any time, the client can decide to send the datagram packet.
The datagram packet will be forwarded to the server using the information contained in the datagram header (server's IP, sender's IP, etc.).
6 Send the datagram on the network (method socket.send(packet))
As the client does not need to wait/acknowledge that the server has received the datagram packet successfully, it can close the connection immediately.
7 Close the connection (method socket.close())
8 Receive the datagram packet (method socket.receive())
9 Get/transform the datagram packet payload (method depending on the type of data)
The server will loop back to receive another datagram.
UDP Client (192.168.1.23)
UDP Server (192.168.1.25)
UDP - Unicast (client/server) workflow (3/4)



UDP - Unicast (client/server) workflow (4/4)

Processing data from datagrams (1/2)

A datagram contains a payload as a byte array.

You can create a datagram with a byte array as payload:

```
// Create a datagram with a payload and a destination address
byte[] buffer = "Hello, World!".getBytes(StandardCharsets.UTF_8);
DatagramPacket packet = new DatagramPacket(
  buffer,
  buffer.length,
  InetAddress.getByName("localhost"),
  1234
);
```

Processing data from datagrams (2/2)

You can get the payload of a received datagram as a byte array:

```
// Create a datagram to receive data
byte[] buffer = new byte[1024];
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

// Receive a datagram
socket.receive(packet);

// Get the payload of the datagram
byte[] data = packet.getData();
int length = packet.getLength();
String message = new String(data, 0, length, StandardCharsets.UTF_8);
```

End of part 1/2 - Questions

Do you have any questions?

Practical content (part 1/2)

What will you do?

- Learn to use the debugger.
- TCP & UDP - Execute the code examples.
- Explore the Java TCP programming template.
- Explore the Java UDP programming template.

Now it's your turn!

- Read the course material.
- Do the practical content.
- Ask questions if you have any.

➡️ Find the course on GitHub.

Do not hesitate to help each other! There's no need to rush!

Part 2/2

In this second part, you will learn more about UDP and some of its unique features compared to TCP.

Explore the code examples

More details for this section in the course material. You can find other resources and alternatives as well.

Explore the code examples

Individually, or in pair/group, **take 15 minutes to explore and discuss the code examples (part 2/2)**.

Answer the questions available in the course material:

- How do the code examples work?
- What are the main takeaways of the code examples?
- What are the main differences between the code examples?

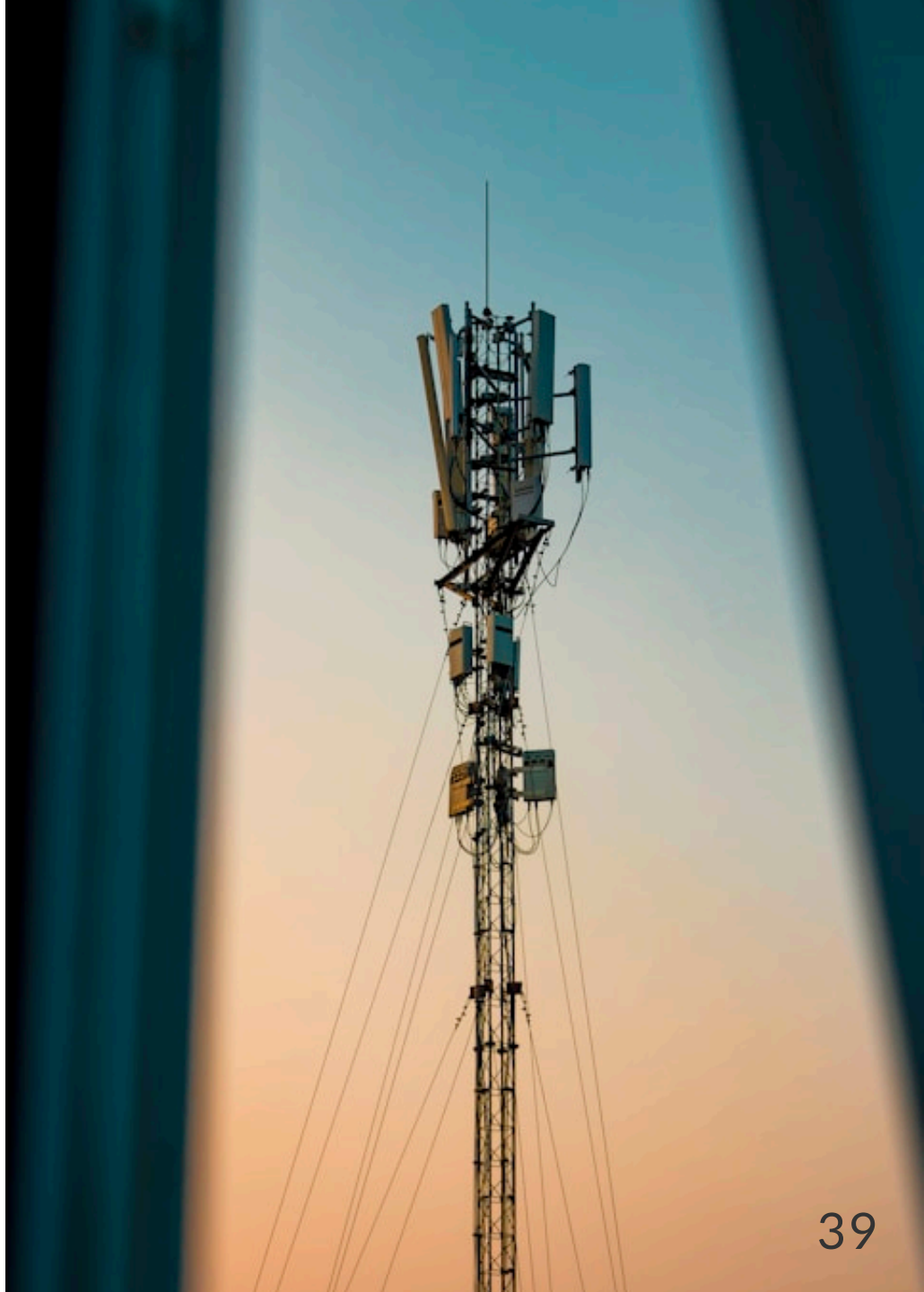
If needed, use the theoretical content to help you.

Unicast, broadcast and multicast

More details for this section in the course material. You can find other resources and alternatives as well.

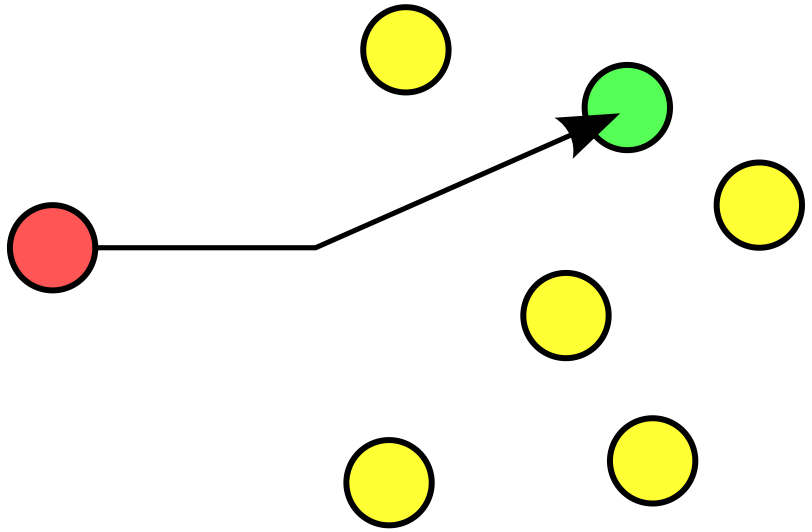
Unicast, broadcast and multicast

- Unicast, broadcast and multicast are ways to send data over the network.
- TCP is unicast only - one sender and one receiver.
- UDP can be unicast, broadcast or multicast.



Unicast

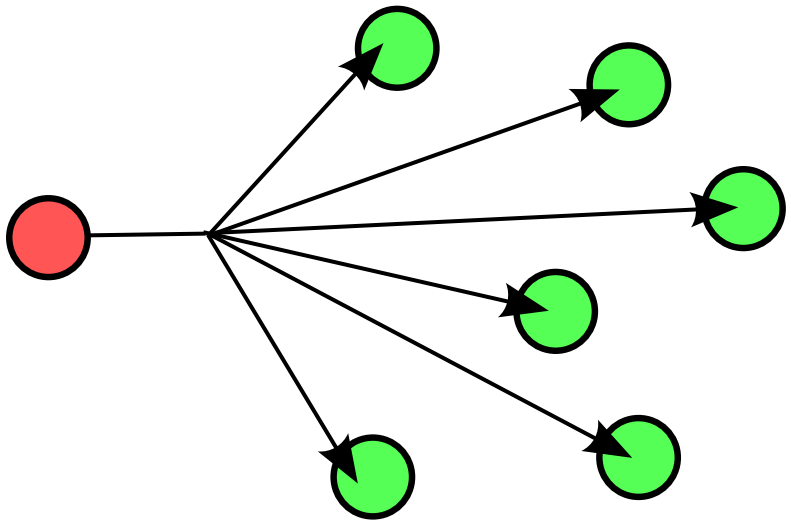
- One-to-one communication.
- One sender and one receiver.
- To send a datagram, the sender must know:
  - The IP address of the receiver.
  - The port of the receiver.



Think of it as a private conversation between two people.

Broadcast

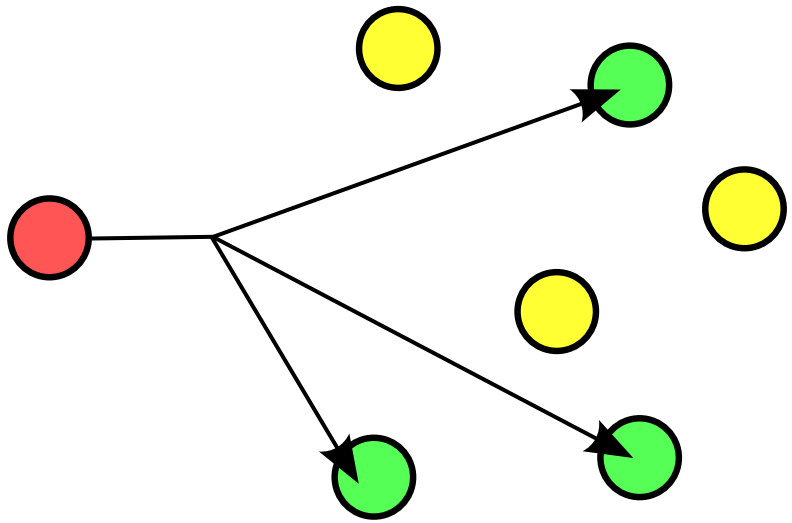
- One-to-all communication.
- One sender and multiple receivers.
- To send a datagram, the sender must know:
  - The network subnet (e.g. `192.168.255.255`).
  - The port.



Think of it as a public announcement.

Multicast (1/2)

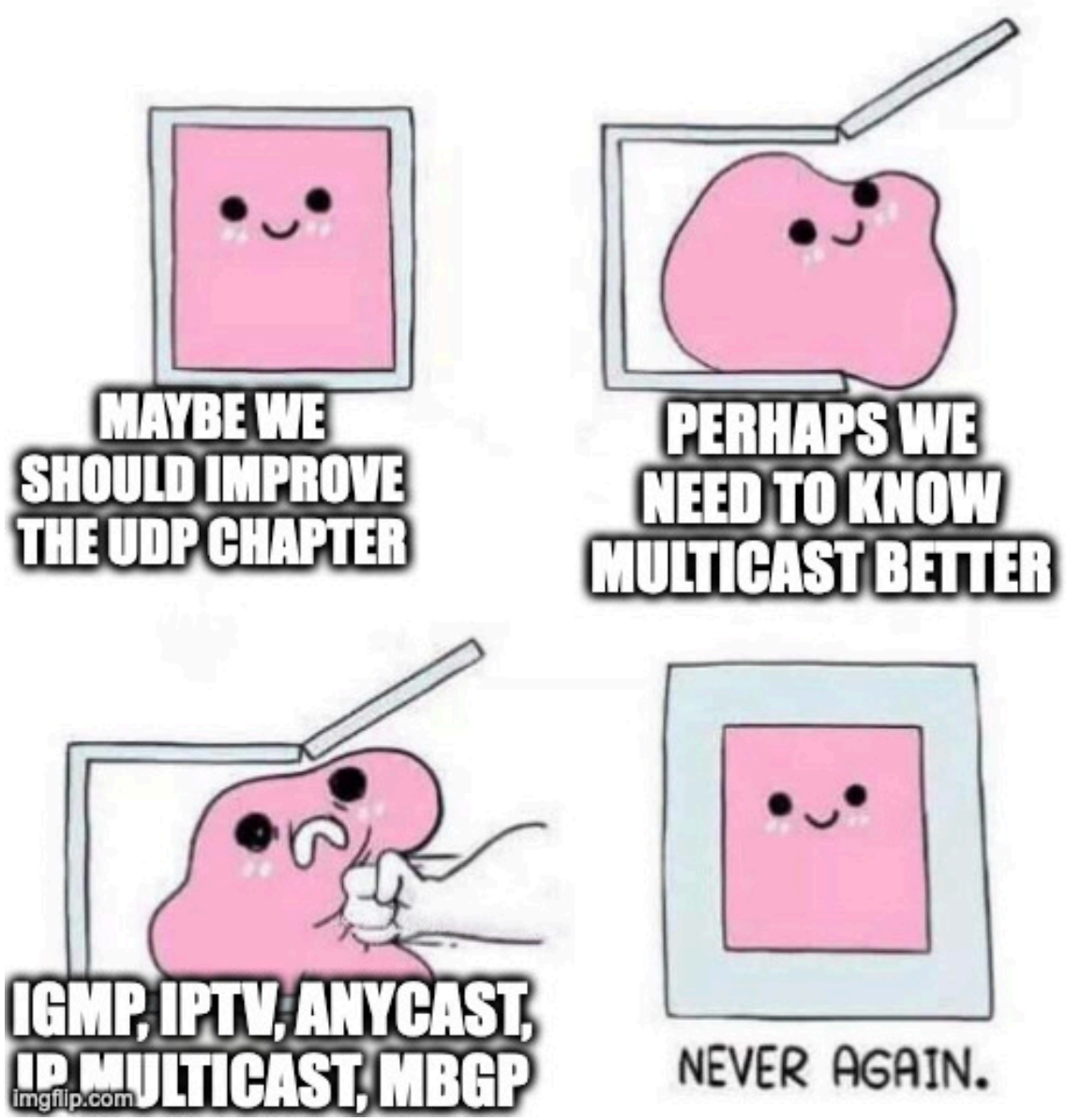
- One-to-many communication.
- One sender and some receivers.
- To send a datagram, the sender must know:
  - The multicast address (between `239.0.0.0` and `239.255.255.255`).
  - The port.



Think of it as a group conversation.

Multicast (2/2)

- Just as with broadcast, it can be blocked by routers.
- Broadcast/multicast are quite guaranteed **not** to work on the public Internet.
- Made for the local network.
- Multicast is a complex topic, not covered in depth in this teaching unit.



Messaging patterns

More details for this section in the course material. You can find other resources and alternatives as well.

Messaging patterns

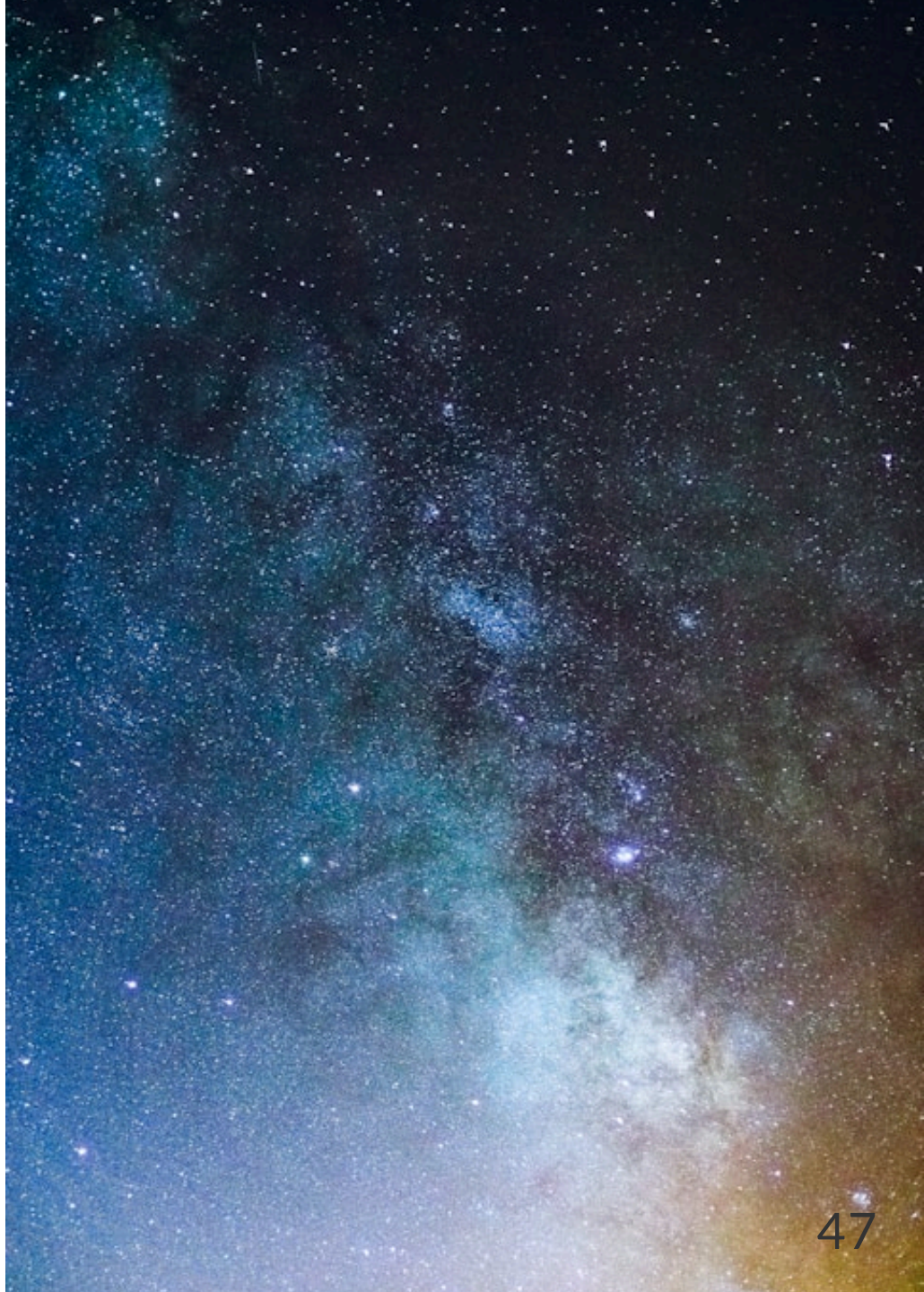
- Fire-and-forget:
  - One-way communication.
  - No response.
  - No guarantee of delivery.
- Request-response:
  - Two-way communication.
  - Response.
  - Guarantee of delivery (manual).

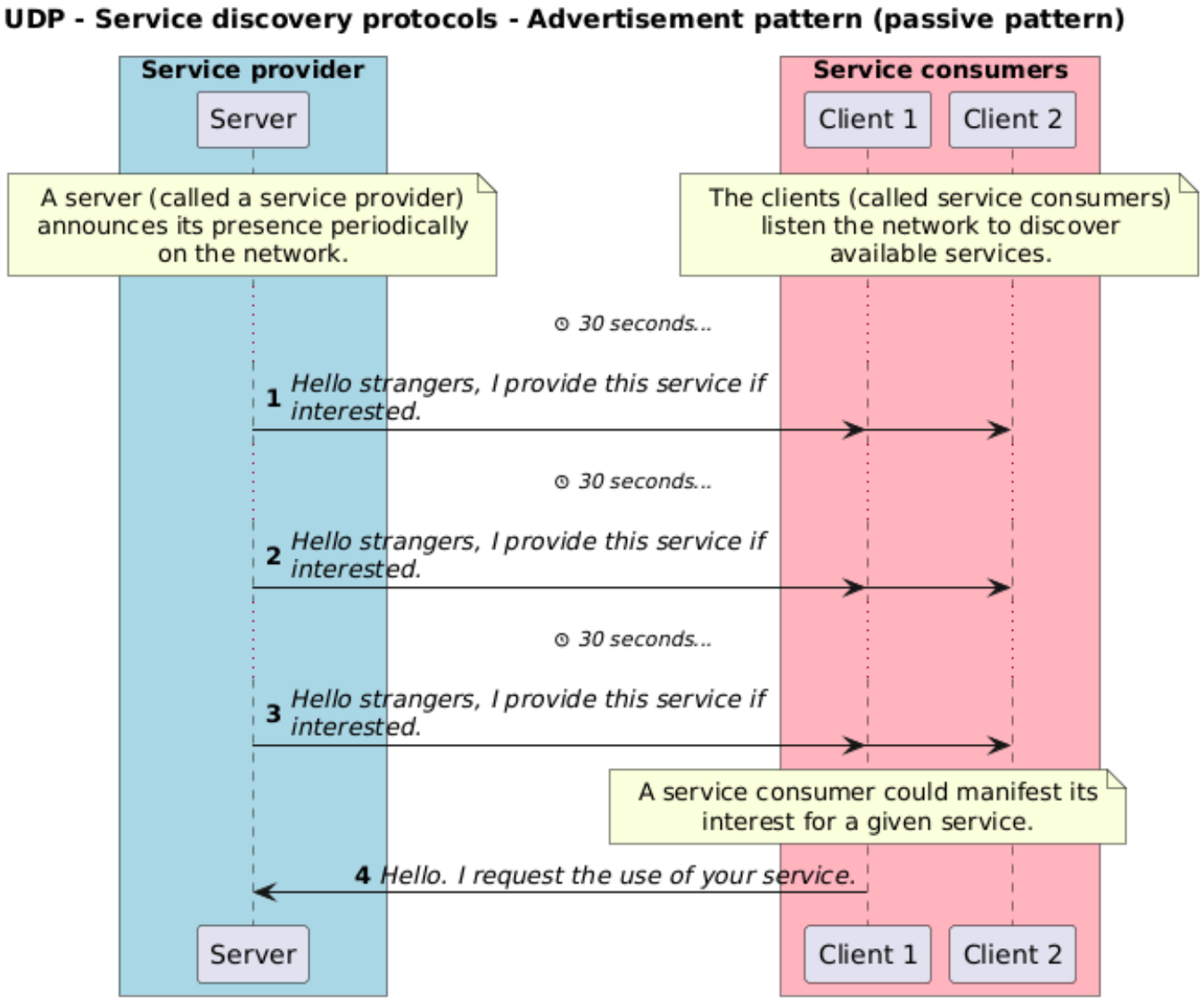
Service discovery protocols

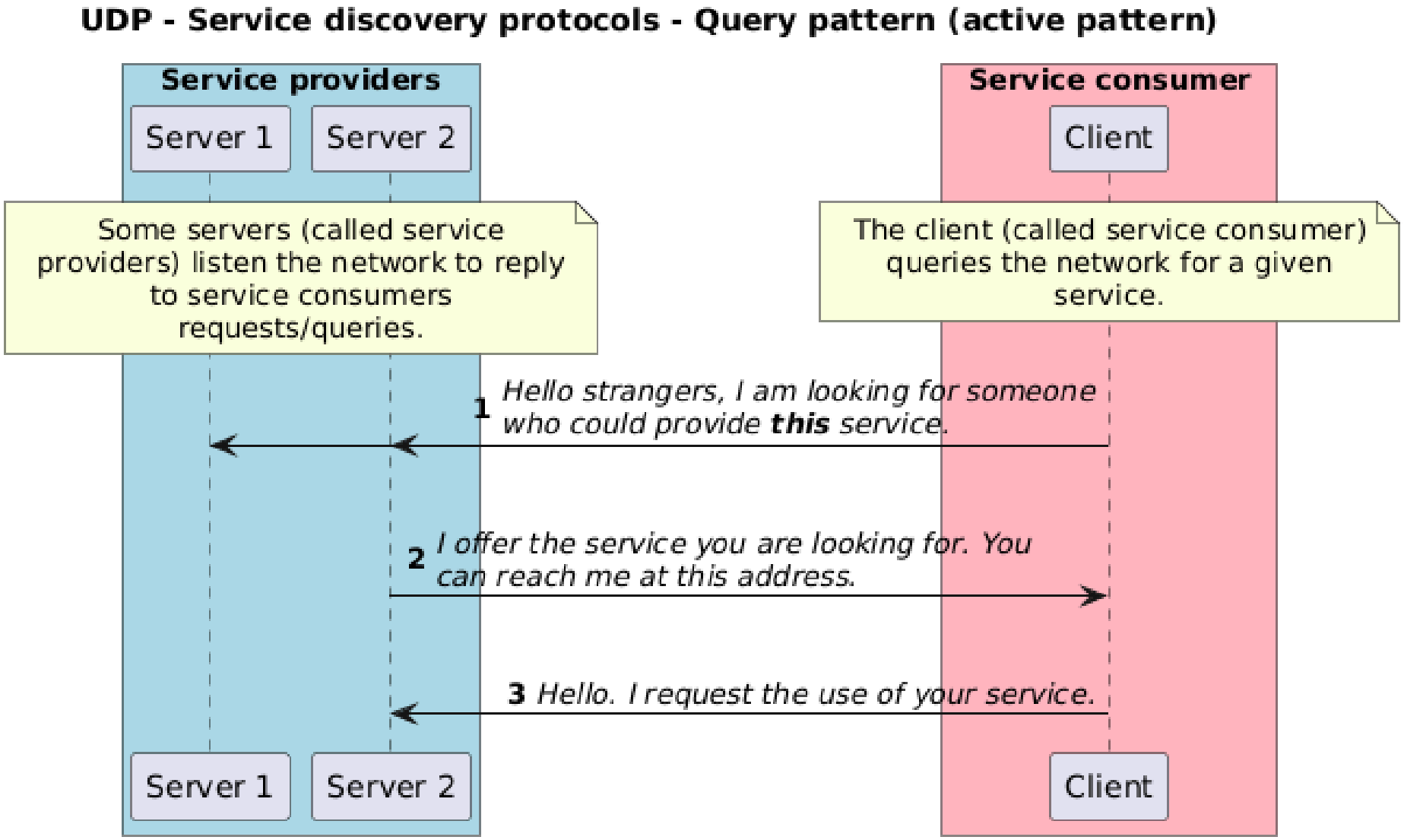
More details for this section in the course material. You can find other resources and alternatives as well.

Service discovery protocols

- Discover services on the network.
- Service discovery protocol patterns:
  - Advertisement (passive).
  - Query (active).




UDP - Service discovery protocols - Advertisement pattern (passive pattern)
Service provider
Server
Service consumers
Client 1
Client 2
A server (called a service provider) announces its presence periodically on the network.
The clients (called service consumers) listen the network to discover available services.
30 seconds...
1 Hello strangers, I provide this service if interested.
30 seconds...
2 Hello strangers, I provide this service if interested.
30 seconds...
3 Hello strangers, I provide this service if interested.
A service consumer could manifest its interest for a given service.
4 Hello. I request the use of your service.
Server
Client 1
Client 2


UDP - Service discovery protocols - Query pattern (active pattern)
Service providers
Server 1
Server 2
Service consumer
Client
Some servers (called service providers) listen the network to reply to service consumers requests/queries.
The client (called service consumer) queries the network for a given service.
1 Hello strangers, I am looking for someone who could provide this service.
2 I offer the service you are looking for. You can reach me at this address.
3 Hello. I request the use of your service.
Server 1
Server 2
Client

Read-Eval-Print Loop (REPL)

More details for this section in the course material. You can find other resources and alternatives as well.

Read-Eval-Print Loop (REPL)

A REPL is a concept that allows you to interact with a program by sending commands to it without restarting it.

In networking, it is used to send commands to a server that will read the commands, evaluate them, and send back the result.

All of this is done in a loop until the client or the server decides to close the connection.

Useful to keep a connection open and send multiple commands!

Summary of differences between TCP and UDP

More details for this section in the course material. You can find other resources and alternatives as well.

TCP	UDP
Connection-oriented	Connectionless
Reliable	Unreliable
Stream protocol	Datagram protocol
Unicast	Unicast, broadcast and multicast
Request-response	Fire-and-forget, request-response
-	Service discovery protocols
Used for FTP, HTTP, SMTP, SSH, etc.	Used for DNS, streaming, gaming, etc.

Questions

Do you have any questions?

Practical content (part 2/2)

What will you do?

- TCP & UDP - Execute the remaining code examples.
- Update the *"Guess the number"* and *"Temperature monitoring"* application protocols.
- TCP - Try to access the TCP server from multiple TCP clients at the same time.
- UDP - Try to emit from multiple UDP emitters at the same time.
- Implement and Dockerize the *"Guess the number"* game (optional).
- Implement and Dockerize the *"Temperature monitoring"* application (optional).

Now it's your turn!

- Read the course material.
- Do the practical content.
- Ask questions if you have any.

➡️ Find the course on GitHub.

Do not hesitate to help each other! There's no need to rush!

Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this course?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

➡️ GitHub Discussions

You can use reactions to express your opinion on a comment!

Sources (1/2)

- Main illustrations by Carl Nenzen Loven on Unsplash and Possessed Photography on Unsplash
- Illustration by Aline de Nadai on Unsplash
- Illustration by Alexander Andrews on Unsplash
- Illustration by Oleksandra Bardash on Unsplash
- Illustration by Becky Phan on Unsplash
- Illustration by Jackson Simmer on Unsplash
- Illustration by Sam Dan Truong on Unsplash

Sources (2/2)

- Illustration by Zuza Gałczyńska on Unsplash
- Illustration by Andy Holmes on Unsplash