Client-Server communication in C#

AARHUS UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

MICHAEL LOFT
ML@ASE.AU.DK

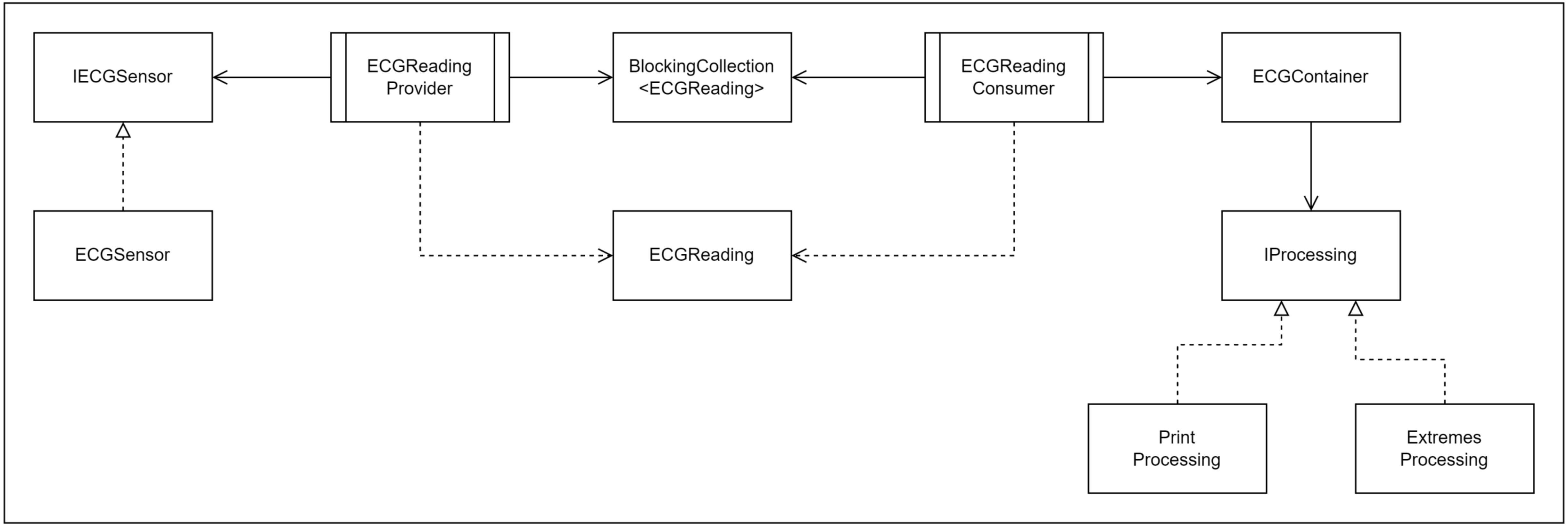
Agenda

The end goal

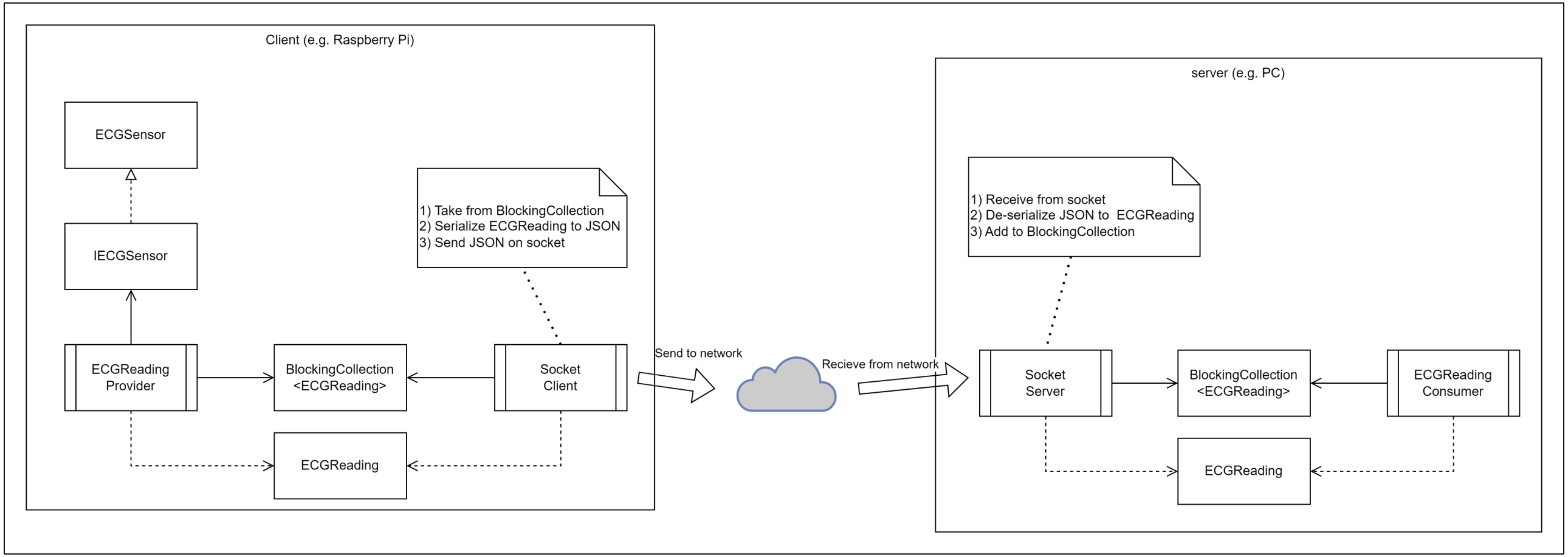
Client <-> Server socket communication

Where we want to end up

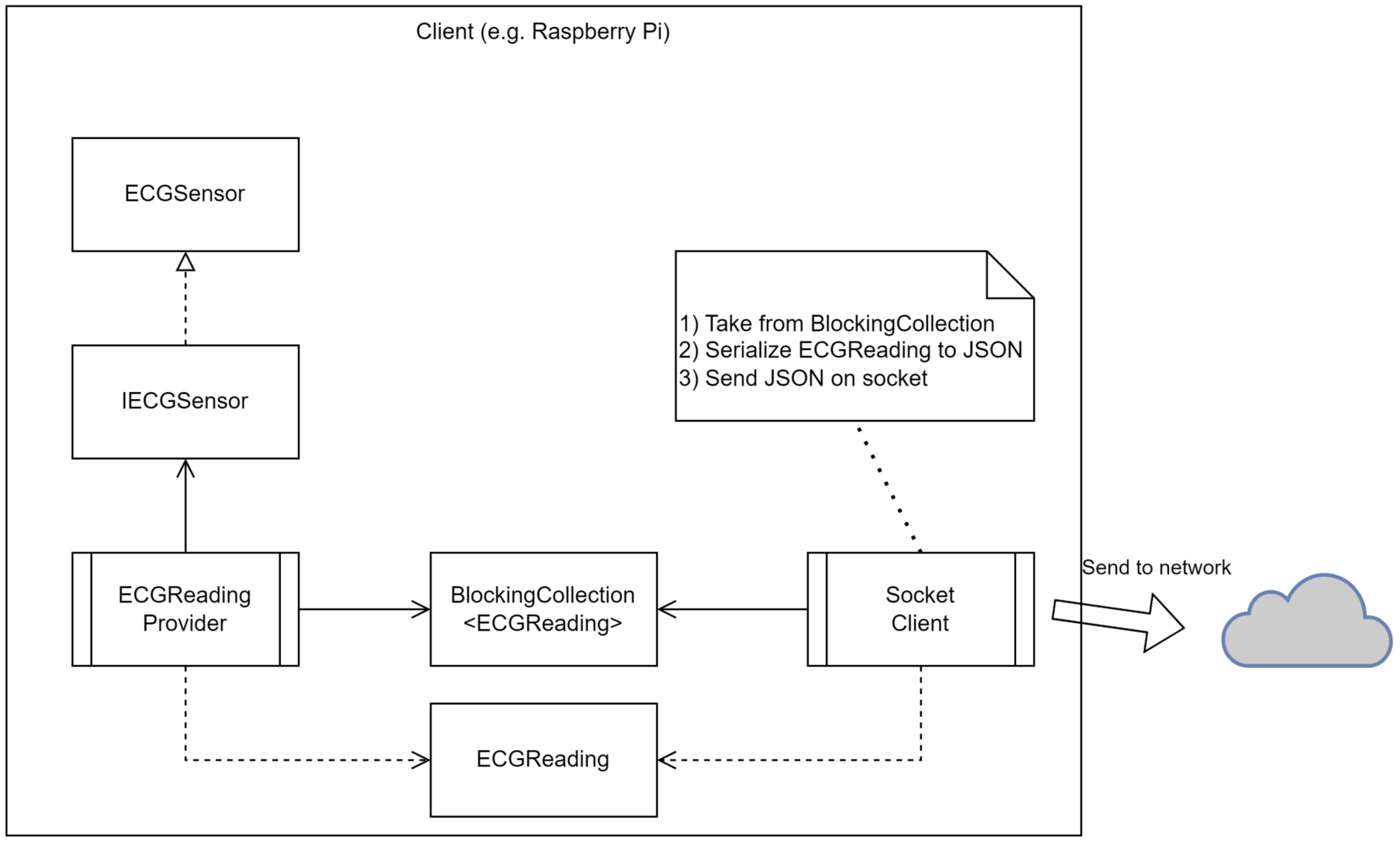
We want to go from this



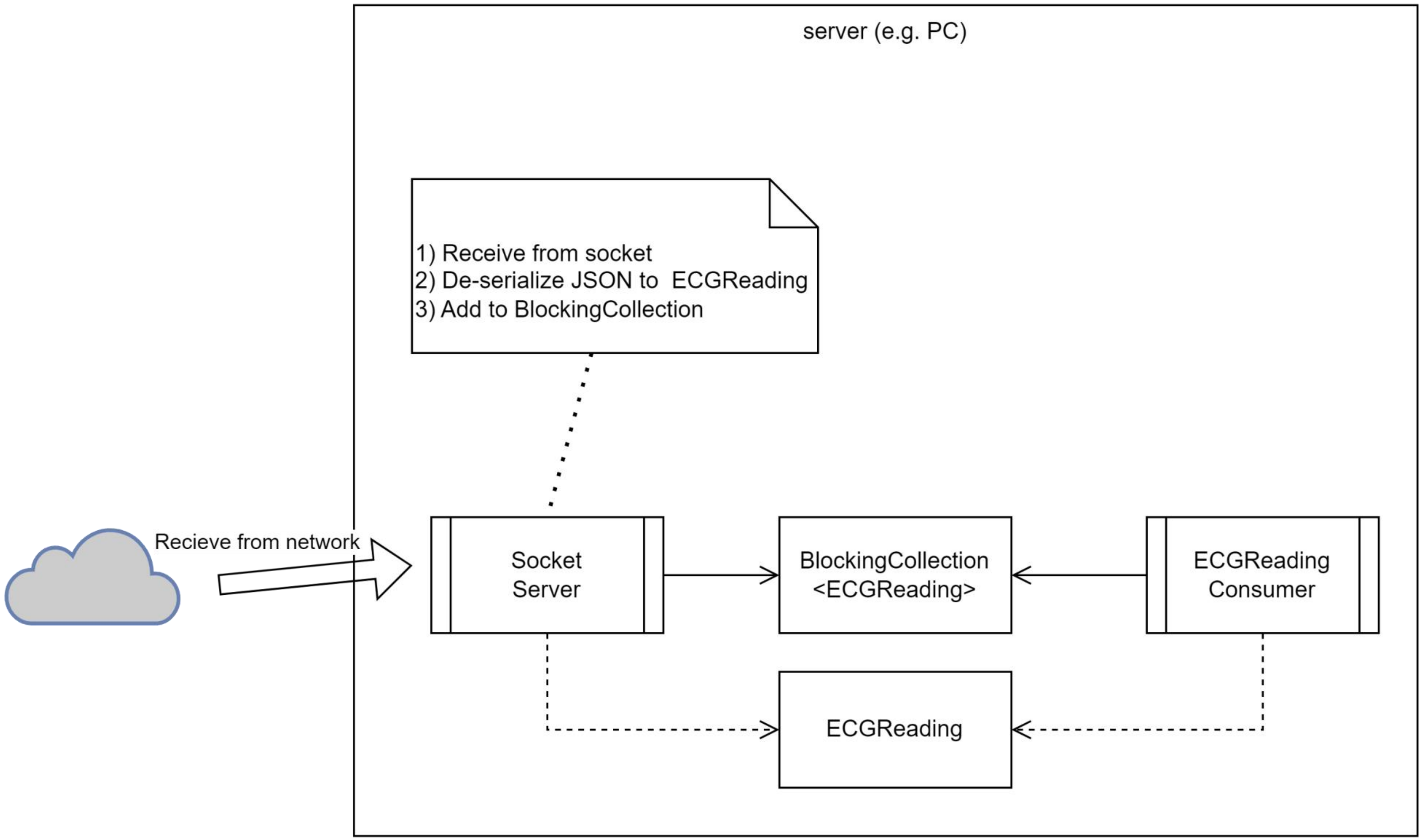
To this


Client (e.g. Raspberry Pi)
ECGSensor
IECGSensor
ECGReading Provider
BlockingCollection <ECGReading>
Socket Client
ECGReading
1) Take from BlockingCollection
2) Serialize ECGReading to JSON
3) Send JSON on socket
Send to network
Recieve from network
server (e.g. PC)
1) Receive from socket
2) De-serialize JSON to ECGReading
3) Add to BlockingCollection
Socket Server
BlockingCollection <ECGReading>
ECGReading Consumer
ECGReading

Client

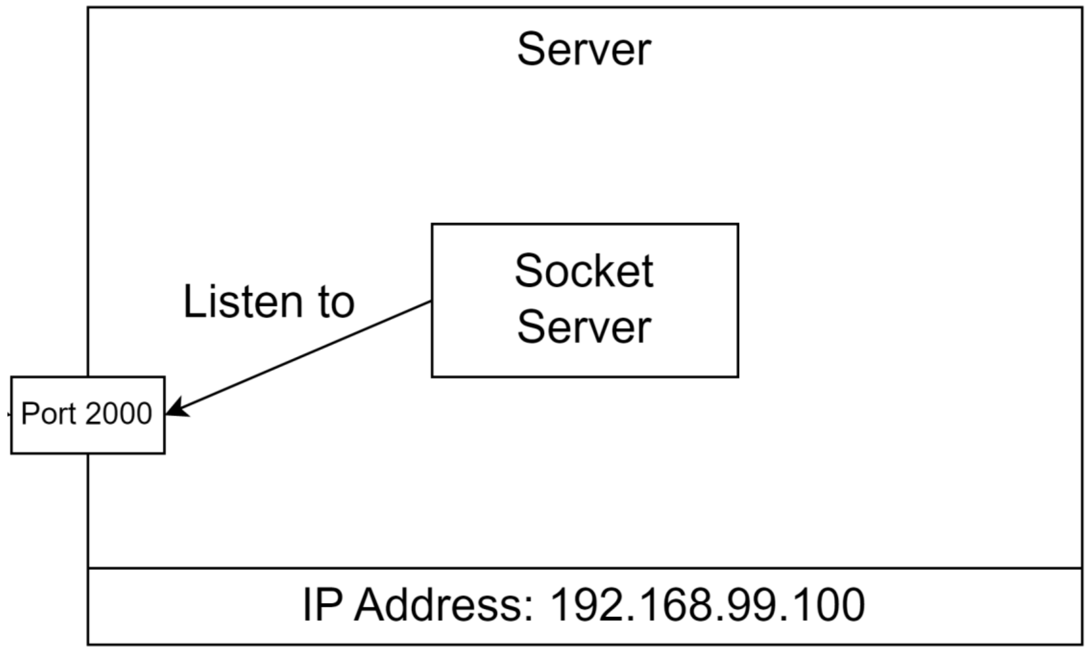

Client (e.g. Raspberry Pi)
ECGSensor
IECGSensor
ECGReading Provider
BlockingCollection <ECGReading>
Socket Client
ECGReading
1) Take from BlockingCollection
2) Serialize ECGReading to JSON
3) Send JSON on socket
Send to network

Server


server (e.g. PC)
1) Receive from socket
2) De-serialize JSON to ECGReading
3) Add to BlockingCollection
Recieve from network
Socket Server
BlockingCollection <ECGReading>
ECGReading Consumer
ECGReading

Socket communication

The server listens



The Server specifies where to listen:

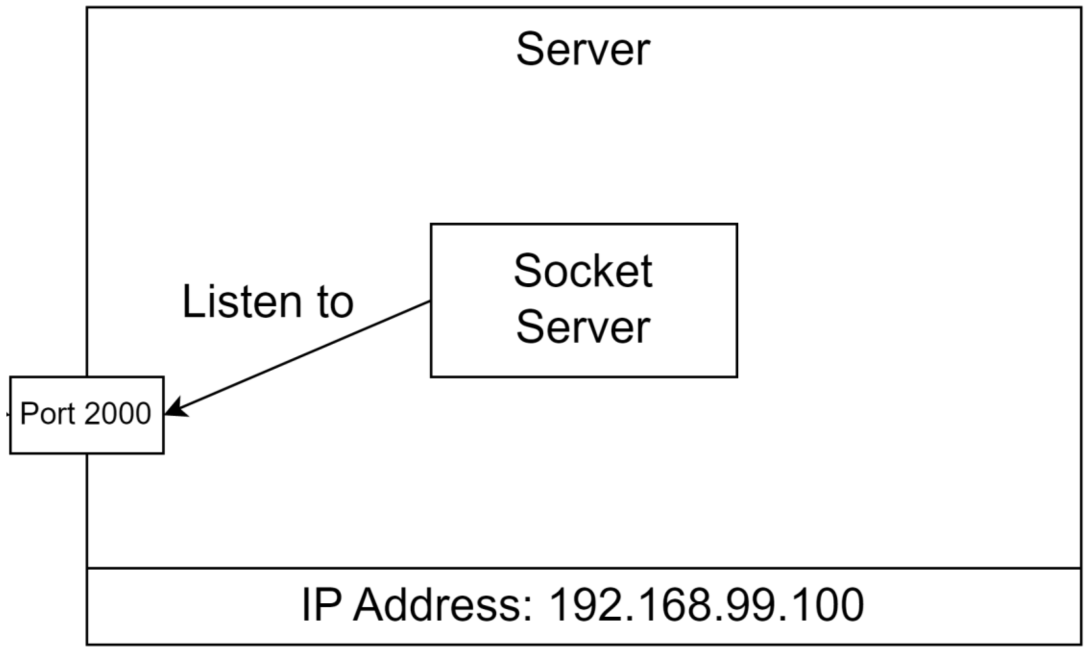
- IP Address
- Port number

The server listens

The server can have multiple network cards and more than one address.

IP Address to listen to can be:

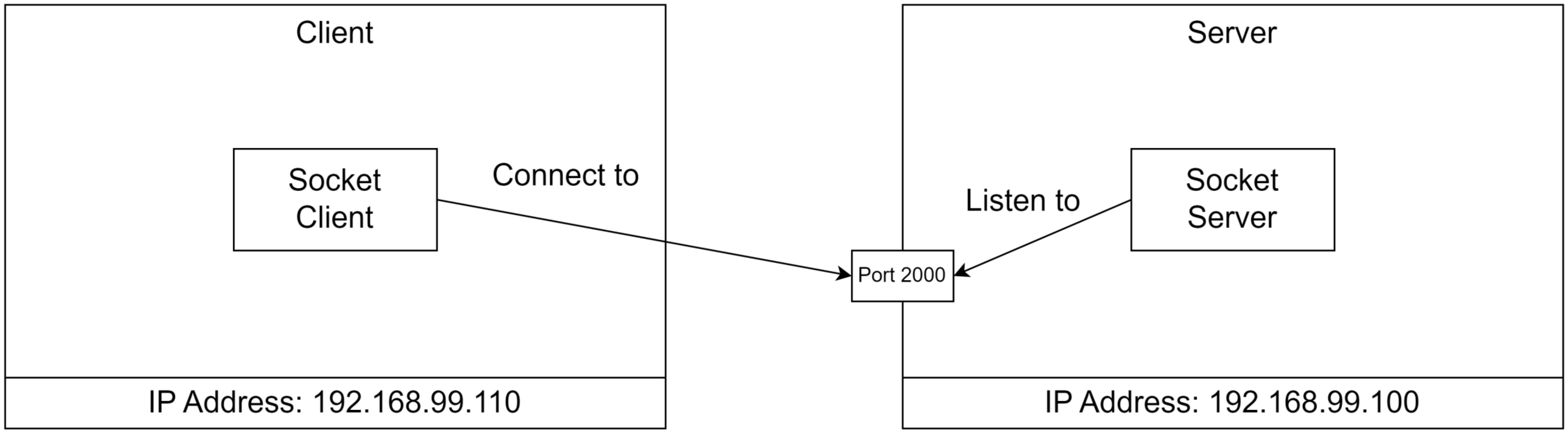
- **Any** <= all network interfaces
- **Loopback** <= 127.0.0.1
- A specific IP address



The Server specifies where to listen:

- IP Address
- Port number

The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketServer

```csharp
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

Create and IPEndPoint with Address = Any and Port = 2000.

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all networ[illegible]dresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPo[illegible](ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Create a new socket.

- 'AddressFamily' is the one for IP addresses – "InterNetwork".
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network add     ses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(i    dress, 2000);

        using Socket listener = new Socket(I   ndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Co  ole.WriteLine($"Listening on: {ipAddress}");

                                                    (buffer, SocketFlags.None);
                                                    ng(buffer, 0, numberOfBytesReceived);
                                                    edData}");

            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Create a new socket.

- ‘AddressFamily’ is the one for IP addresses – ”InterNetwork”.
- Socket type is Stream, which means a two-way data stream.
- The protocol type is Transmission Control Protocol - TCP.

The Socket implements the ‘IDisposable’ interface.

The ‘using’ keywork means, that the ‘listener’ object will be disposed correctly, even if any exceptions occur in the code.

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all networ[illegible]dresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPo[illegible](ipAddress, 2000);

        using Socket listener = new S[illegible]t(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Associate socket with local endpoint on the machine.

SocketServer

Start listening on the socket.

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all networ[illegible]dresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoi[illegible](ipAddress, 2000);

        using Socket listener = new S[illegible]et(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"[illegible]ening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network address   for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddr   , 2000);

        using Socket listener = new Socket(ipEndP   t.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on:   pAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Wait for a connection from a client.

SocketServer

Read bytes from the socket in to a buffer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily  SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

Convert to text (UTF-8)

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamil  SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receiv   uffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamil   SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receiv   uffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{receivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Read bytes from the socket in to a buffer

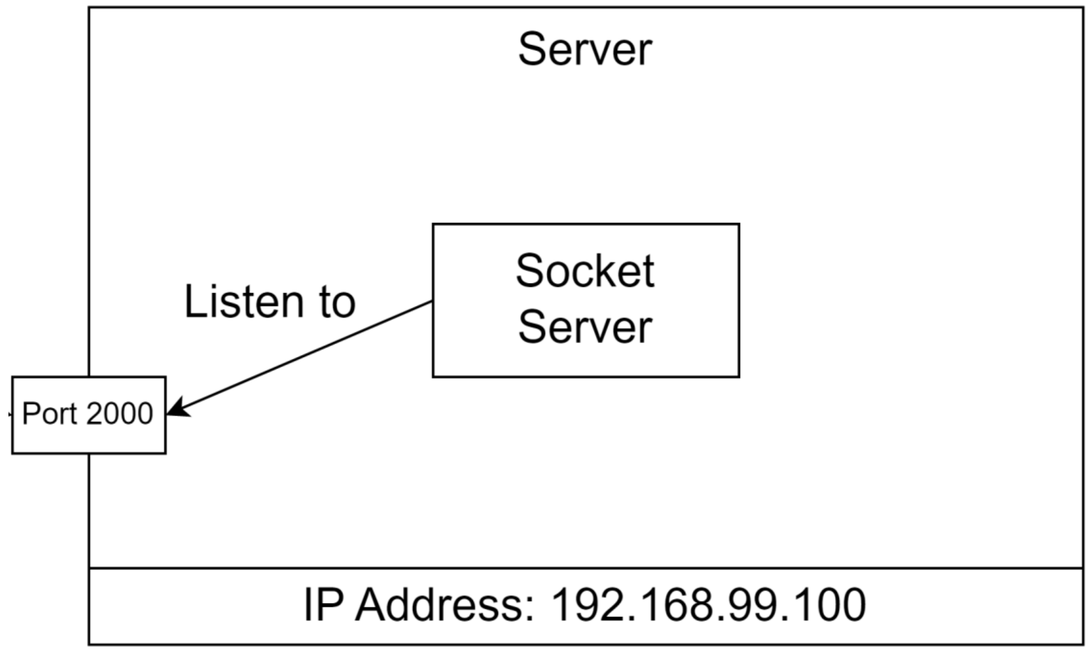
SocketServer

```
class SocketServer
{
    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for th
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily  SocketType.Stream, ProtocolType.Tcp);
        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (true)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Recei  buffer, SocketFlags.None);
            string receivedData = Encoding.UTF8.GetS   ng(buffer, 0, numberOfBytesReceived);
            Console.WriteLine($"Server received:{re   ivedData}");

            string reply = "ACK";
            byte[] replyBytes = Encoding.UTF8.GetBytes(reply);
            handler.Send(replyBytes, SocketFlags.None);
        }
    }
}
```

Send ”ACK” message back to the client.

NOTE:
There is no requirement to send anything back to the client. It is only included here to show two-way communication.

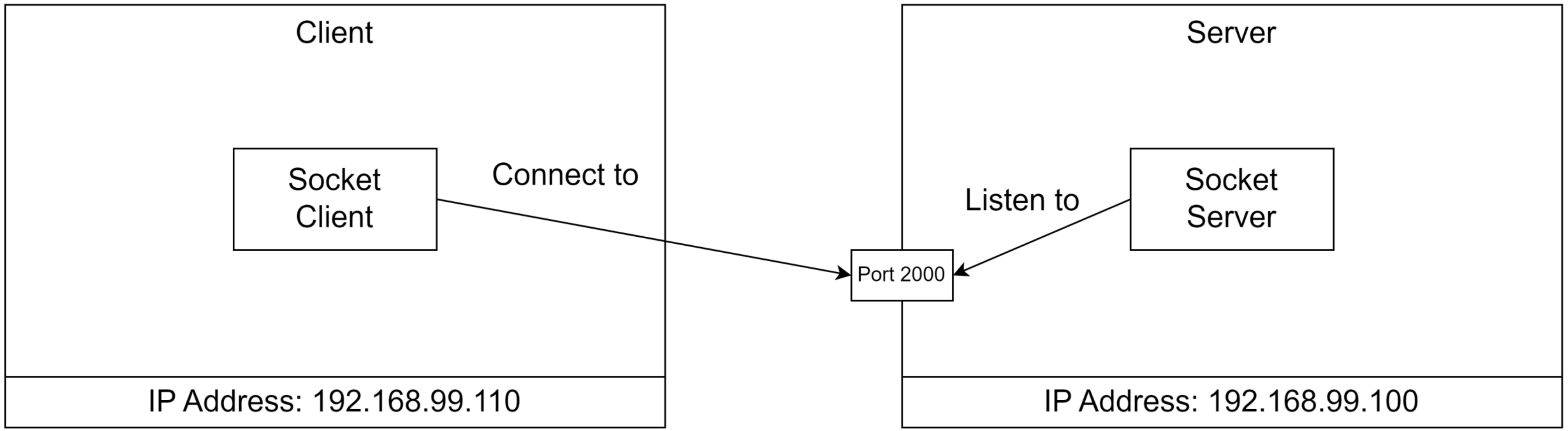
The server listens



The Server specifies where to listen:

- IP Address = Any
- Port number = 2000

The client connects to the server



The client connects to a server socket:

- IP Address
- Port number

SocketClient

```csharp
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Create and IPEndPoint with
Address = 127.0.0.1 (localhost)
and
Port = 2000.

If the server is on a different machine, the Address shall be the IP address of that machine.

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("1[illegible].0.1");
        IPEndPoint ipEndPoint = new IPEndPo[illegible]ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Create the socket and connect to the IP Endpoint.

The connection attempt may time out if the endpoint does not exist, which will throw an exception.

SocketClient

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, [illegible]000);

        using Socket client = new Socket(ipEndPoint.A[illegible]essFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetBytes(message);
            client.Send(messageBytes, SocketFlags.None);
            Console.WriteLine($"Socket client sent message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Create the message to send.
Convert to bytes.
Send the bytes.

SocketClient

Receive "ACK" message from the server.

NOTE:
There is no requirement to send anything back to the client. It is only included here to show two-way communication.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2(   ));

        using Socket client = new Socket(ipEndPoint.Addres  amily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetByt  (message);
            client.Send(messageBytes, SocketFlags.  ne);
            Console.WriteLine($"Socket client sen  message: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buffer, SocketFlags.None);
            var response = Encoding.UTF8.GetString(buffer, 0, received);

            Console.WriteLine($"Client received: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

SocketClient

Shutdown both sending and receiving socket when done.

```
class SocketClient
{
    public void RunClient()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2(   ));

        using Socket client = new Socket(ipEndPoint.Addres   amily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        for (int i = 0; i < 10; i++)
        {
            // Send message.
            var message = "Hello " + i;
            var messageBytes = Encoding.UTF8.GetByt   (message);
            client.Send(messageBytes, SocketFlags.   ne);
            Console.WriteLine($"Socket client sen   essage: {message}");

            // Receive ack.
            var buffer = new byte[1024];
            var received = client.Receive(buf   , SocketFlags.None);
            var response = Encoding.UTF8.Get   ing(buffer, 0, received);

            Console.WriteLine($"Client rec   ed: {response}");
            Thread.Sleep(1000);
        }

        client.Shutdown(SocketShutdown.Both);
    }
```

Your turn

Solve the

”Exercise - Client-Server communication”

Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        while (!_ecgReadings.IsCompleted)
        {
            try
            {
                ECGReading ecgReading = _ecgReadings.Take();

                string objectAsJson = JsonSerializer.Serialize(ecgReading);
                var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {objectAsJson}");
            }
            catch (InvalidOperationException)
            {
                // IOE means that Take() was called on a completed collection.
            }
        }
        client.Shutdown(SocketShutdown.Both);
    }
}
```

Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);
        client.Connect(ipEndPoint);

        while (!_ecgReadings.IsCompleted)
        {
            try
            {
                ECGReading ecgReading = _ecgReadings.Take();

                string objectAsJson = JsonSerializer.Serialize(ecgReading);
                var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {objectAsJson}");
            }
            catch (InvalidOperationException)
            {
                // IOE means that Take() was called on a completed collection.
            }
        }
        client.Shutdown(SocketShutdown.Both);
    }
}
```

Receiving objects on the server

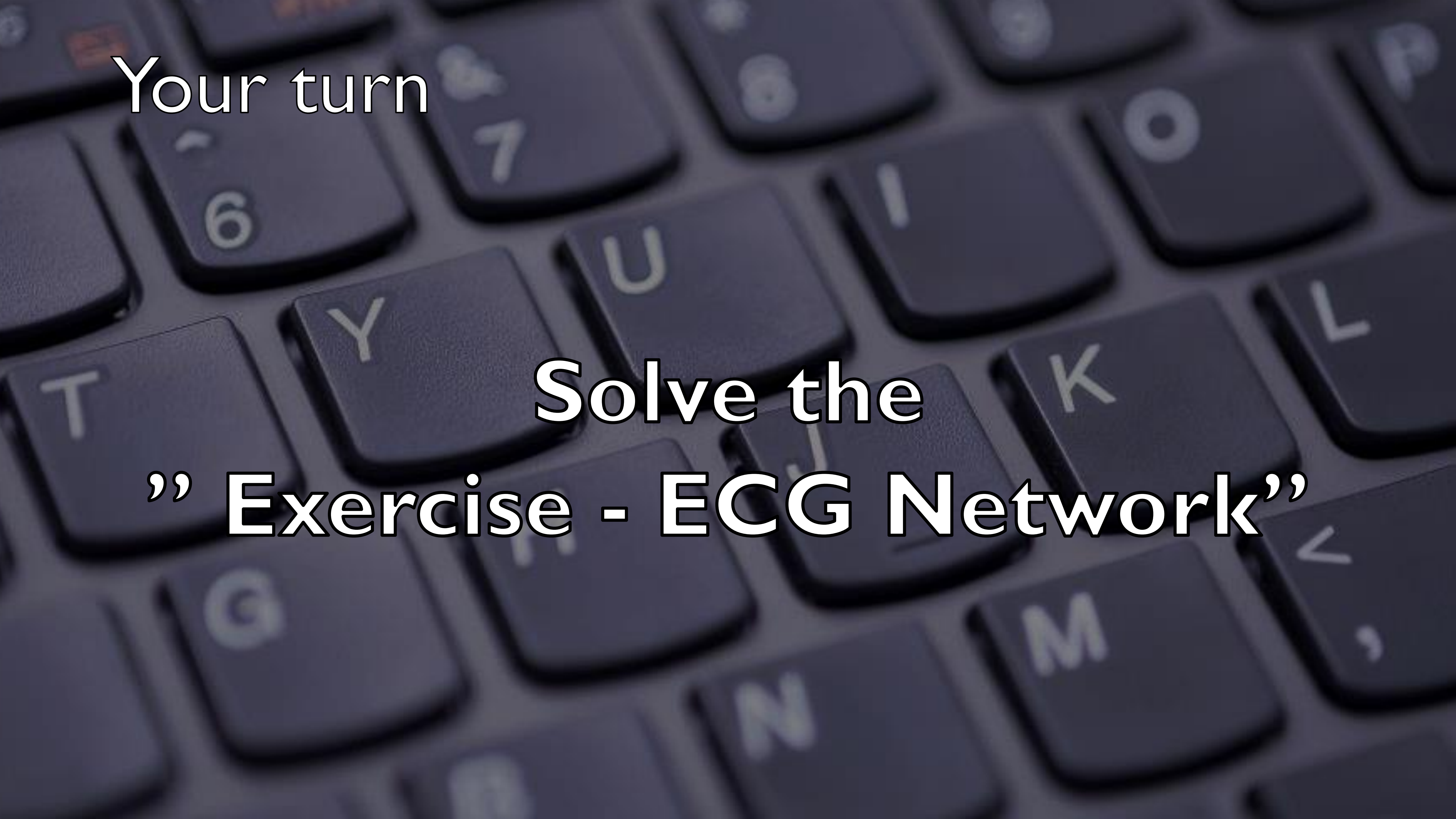
```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }
    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (!ShallStop)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            if (numberOfBytesReceived > 0)
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading = JsonSerializer.Deserialize<ECGReading>(receivedData);
                    if (ecgReading != null) _ecgReadings.Add(ecgReading);
                }
                catch (System.Text.Json.JsonException e)
                {
                    // log any parsing exceptions
                    Console.WriteLine(e);
                }
            }
        }
        listener.Close();
        _ecgReadings.CompleteAdding();
    }

    public bool ShallStop { get; set; } = false;
}
```

Receiving objects on the server

```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }
    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily,
        using Socket listener = new Socket(ipEndPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (!ShallStop)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            if (numberOfBytesReceived > 0)
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData);
```

```
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData);
                    if (ecgReading != null) _ecgReadings.Add(ecgReading);
                }
                catch (System.Text.Json.JsonException e)
                {
                    // log any parsing exceptions
                    Console.WriteLine(e);
                }
            }
        }
        listener.Close();
        _ecgReadings.CompleteAdding();
    }

    public bool ShallStop { get; set; } = false;
}
```

Your turn



Solve the

” Exercise - ECG Network”

References and image sources

AARHUS UNIVERSITY